A blackboard-based system for learning to identify images from feature data

A BLACKBOARD-BASED SYSTEM FOR LEARNING TO IDENTIFY IMAGES FROM FEATURE DATA

by

Margaret Norman, M.A., M.Sc.

A Doctoral Thesis
submitted in partial fulfilment of the requirements
for the award of Doctor of Philosophy
of the Loughborough University of Technology

January 1991

SUPERVISOR: Dr. C.J. Hinde
Department of Computer Studies

CONTENTS

ACKNOWLEDGEMENTS

I would like to express my thanks to the following people:

Dr. C.J. Hinde, my supervisor, for his patient and friendly assistance and help throughout this project, much good advice and loan of many useful books and journals.

Dr. P. Fretwell and Dr. R.W. Series at R.S.R.E. Malvern for their advice and help, supplying the car images used for the tests of the initial system and several useful references, and entertaining me so well on my visits to Malvern.

Professor E.A. Edmonds, my director of research, for overseeing the project so competently and thus helping to ensure that it was completed on schedule.

The many other members of staff at Loughborough who helped to solve various problems.

Simon Polovina for the illustration of a conceptual graph in Chapter 2.

Last, but by no means least, my husband Eddie and daughters Katherine, Elizabeth and Jennifer for all their support.

ABSTRACT

A blackboard-based system which learns recognition rules for objects from a set of training examples, and then identifies and locates these objects in test images, is presented. The system is designed to use data from a feature matcher developed at R.S.R.E. Malvern which finds the best matches for a set of feature patterns in an image. The feature patterns are selected to correspond to typical object parts which occur with relatively consistent spatial relationships and are sufficient to distinguish the objects to be identified from one another.

The learning element of the system develops two separate sets of rules, one to identify possible object instances and the other to attach probabilities to them. The search for possible object instances is exhaustive; its scale is not great enough for pruning to be necessary. Separate probabilities are established empirically for all combinations of features which could represent object instances. As accurate probabilities cannot be obtained from a set of preselected training examples, they are updated by feedback from the recognition process.

The incorporation of rule induction and feedback into the blackboard system is achieved by treating the induced rules as data to be held on a secondary blackboard. The single recognition knowledge source effectively contains empty rules which this data can be slotted into, allowing it to be used to recognise any number of objects - there is no need to develop a separate knowledge source for each object. Additional object-specific background information to aid identification can be added by the user in the form of background checks to be carried out on candidate objects.

The system has been tested using synthetic data, and successfully identified combinations of geometric shapes (squares, triangles etc.). Limited tests on photographs of vehicles travelling along a main road were also performed successfully.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1. BACKGROUND AND PROJECT AIMS.

The Royal Signals and Radar Establishment (R.S.R.E.) at Malvern have been engaged in a considerable amount of research in visual pattern recognition, much of which has been concerned with interpreting photographs of outdoor scenes. A particular area of study has been the location and identification of cars and other vehicles in black-and-white images of road scenes.

One technique which R.S.R.E. have been working on is feature matching. The intention of their project was to produce a program which would find the ten 'best matches' for a given pattern (e.g. part of a car - a wheel shape, or a window shape) in a line-edged picture, giving mean x and y co-ordinates and a rating or cost for each match found.

The initial motivation for this research was the desire to find some way to utilise the output from such a feature matcher in a system which would learn to locate and identify the vehicles in a road scene, or to perform other similar pattern recognition tasks.

1.2. DESCRIPTION OF THE FEATURE MATCHER.

The feature matcher is based on a modification of the connected word recognition algorithm used extensively in speech recognition (Holmes, '88). This algorithm matches acoustic patterns for all the words to be

recognised to the incoming speech signal. One problem with the matching is that the speed of speaking varies; there may be differences in time scale between two utterances of the same word. A mathematical technique known as dynamic programming (or dynamic time warping, when applied to speech recognition) is used to compensate for time scale variations. This technique finds the least-cost path which matches frames of the speech pattern to frames of a word pattern, with the constraint that paths may advance one frame along either or both of the patterns being matched at each step. If d(i,j) is a measure of the difference between frame i of the speech pattern and frame j of the template, D(i,j) is the accumulated cost of the best path from the start of the match to (i,j) and p is a penalty to be attached to a one-frame time distortion, then as (i,j) can be reached from (i-1,j), (i-1,j-1) or (i,j-1),

$$D(i,j) = \min\{D(i-1,j)+p, D(i-1,j-1), D(i,j-1)+p\} + d(i,j).$$

As all paths start from (1,1),

$$D(1,1) = d(1,1).$$

These formulae can be used to draw up an accumulated cost matrix and thus to establish the match cost.

The E.L.S. (Edge List Search) technique for feature matching (Varga et al., '89) uses dynamic programming, applying it to spatial distortion instead of temporal distortion. It is reference-data driven rather than observed-data driven: data is found to match a given reference shape rather than shapes being found to match given data, because the image data can be expected to contain a large amount of background information which does not require identification; observed-data driven techniques are more applicable to areas such as speech recognition where the data is one-dimensional rather than two-dimensional.

The first stage of the image processing involves using an operator based on the Sobel edge convolution to obtain a line-edge description. An algorithm which determines, normalises and thresholds the total change in line orientation within an eight-connected line segment is then applied to remove 'rough' lines representing vegetation in the background of the image. The resulting edge map is broken up into isolated line segments by removing points with more than two edge point neighbours and arbitrarily breaking closed shapes. Very short (one or two pixel) line segments are removed, as these generally represent noise. The outcome of the processing is a list of line segments, each of which is recorded in both forward and reverse directions in the form of a list specifying the horizontal and vertical position and orientation of each pixel.

A 'linkage' matrix which specifies allowed connectivities between segments is drawn up by applying a threshold to the Euclidian distance between segment ends. This matrix is used to reduce the size of the search space and thus speed up the search.

The reference model consists of a set of templates, or feature patterns, which are long unbroken edge segments. The system uses a one-pass dynamic programming technique to find sequences of observed segments which match each pattern, assigning a local match cost, which is a function of connectivity, segment shape and orientation, and an accumulated cost based on the best sequence of matches to date, to each match.

Segment transitions are recorded in a decision matrix or linked list. When the end of the template is reached, the lowest accumulated cost is selected as the starting point for a traceback to give the sequence of

segments which represent the optimal match. If identified segments are deleted after traceback, the process can be repeated to yield a complete set of graded matches.

1.3. CHARACTERISTICS AND LIMITATIONS OF THE MATCHER.

The patterns for which matches are to be sought obviously need to be selected with care if the output of the matcher is to be useful. The approach which was initially adopted was to select a clear edged picture of a typical car, edit it to remove irregularities and breaks in the lines, then to select segments of this picture corresponding to standard car parts for matching. The segments initially used were: car wheel, wheel arch, front window, rear window, roof/bonnet shape.

The matcher is intended to be sensitive to the exact shape of the pattern, so it is important that the patterns used should be as typical as possible of the car parts which they are intended to locate. There are considerable differences between the shapes of parts on different models of car; some experimentation may be necessary to decide which patterns will give the best matches on the widest possible range of models. As any one car is unlikely to be typical in all respects, it may be best to use a window shape from one picture, a roof shape from another etc. In order to match all examples adequately, it may be necessary to use more than one pattern for each part.

The size of the patterns is also significant; where, for example, the pattern being matched is a circle (car wheel shape), good match ratings will only be obtained from circles of the same size as the pattern. A

square of approximately the same size may produce a better rating than a circle which is too big or too small. If vehicle parts are to appear approximately the same size in all the pictures to be studied, it is important that all vehicles should be photographed from approximately the same distance - from a fixed position alongside a road, for example.

The variations in size and shape which inevitably occur in 'real' data mean that the matcher has to be fairly flexible if it is to pick up a useful proportion of the vehicle parts being sought, and this flexibility means that it will also inevitably detect a considerable number of 'false' features: windows in buildings in the background, small bushes etc. may be identified as possible car wheels or windows, and wheels and windows may be mistaken for one another or the same image feature may be identified as both. Initial trials suggested that the match ratings would not be a reliable indicator of the presence or absence of a car part, and that all matches whose ratings exceeded a threshold value (which would depend on the size and shape of the pattern) would have to be considered by the vehicle recognition process.

Another problem when using 'real' data is that the features being sought often do not show up well in photographs, and may be lost altogether during the initial processing of the image to extract edges. Wheels are particularly difficult to pick up, especially in poor lighting conditions. The body outline may also not be clear, particularly if a vehicle is dark in colour and is photographed against a dark background. The list of matches found may, therefore, include only a few vehicle parts even if a complete, unoccluded vehicle appears in the photograph.

The amount of information provided on each match is limited to the

pattern identifier, mean x and y co-ordinates and a rating (on which limited reliance can be placed). The task, therefore, is to develop a system which will learn to identify objects, given just the co-ordinates of a set of features of which a small proportion may belong to instances of one or more of the categories of object being sought.

1.4. APPROACH.

The learning and recognition problem outlined above may be split up into a number of sub-problems, which overlap one another to some extent. The first task is to select for each category of object to be identified a set of parts and corresponding feature patterns which should be sufficient to identify the object and to distinguish it from all the other possible objects. A set of pictures must then be obtained which show as wide a range of instances of the object as the system is intended to be able to identify, and training data must be obtained from these pictures for use in learning an object description. This training data should include feature match data on the pictures, and data giving the locations of all the relevant parts of each object instance.

The training data must be used to develop a set of rules which define the object in terms of the parts being matched and the relationships between these parts. The rules must contain sufficient information to allow the recogniser to identify and group together the features which could correspond to a single object instance.

The complete set of object parts may be sufficient to give a positive identification, but it must be borne in mind that when using real data, only

a few of the parts of any object instance may be detected. The identification rules must include some means of determining the likelihood that any incomplete part set represents an object instance. If the training data has been carefully preselected, it will not be possible to determine such likelihoods from this data; some form of continued learning, using feedback about the performance of the recogniser on real data, will have to be employed.

Finally, the recognisers for several different types of object must be linked together, along with a program for determining which data should be submitted to which recogniser, a means of resolving conflicts between the recognisers and a continued learning system for updating probabilities, to produce an integrated recognition system. Some means must be found of enabling the various elements of the system to interact and communicate with one another; a blackboard system would seem to offer the appropriate capabilities.

1.5. RESEARCH CONSIDERATIONS AND OBJECTIVES.

The main area of this research is pattern recognition, particularly the use of rule induction in developing pattern descriptions to enable effective visual identification of objects. The research objectives here are first, to study previous work in pattern recognition - both visual identification and pattern recognition in general - and to determine how to adapt existing methods to deal with the novel type of data being used here, and second, to look at rule induction techniques and determine how they can best be applied to the development of the appropriate rules from the training data available and how continued induction can be used to update the rules in

the light of the experience gained by the system in operation.

A secondary area of interest is the use of blackboard systems for controlling the interaction of diverse knowledge sources to solve complex problems. Blackboard systems have already been used for a number of applications in the pattern recognition field, but existing systems use knowledge sources which are independent of one another; the major concern here is to find a way of adapting the standard blackboard model to allow one knowledge source (the learner) to modify and update other knowledge sources (the recognisers).

The final topic for research is methods of handling uncertainty, both numerically and symbolically. Uncertainty arises here in the identification of image features, the selection of groups of features which represent different parts of a single object, the assignment of probabilities to feature set/object part-set pairs and the resolution of conflicts between recognisers; appropriate methods need to be identified to deal with each of these.

1.6. THESIS ORGANISATION.

The following four chapters of this thesis contain literature surveys covering the areas of research outlined in the previous section: pattern recognition in Chapter 2, rule induction, including its application to pattern recognition, in Chapter 3, blackboard systems in Chapter 4 and methods of handling uncertainty in Chapter 5.

The development and testing of the programs which have been

produced is covered by the next four chapters: Chapter 6 is on the design of a system which will deal with a single object, Chapter 7 describes the testing of this system, Chapter 8 explains how it was incorporated into a blackboard system to produce an image identifier capable of learning to recognise a range of different objects, and Chapter 9 describes the image identifier tests. Finally, Chapter 10 summarises what has been achieved and contains suggestions for further work. Appendix A contains a full listing of the image identifier system and Appendix B contains edited test runs.

CHAPTER 2

PATTERN RECOGNITION

2.1. WHAT IS PATTERN RECOGNITION?

Firstly, what is recognition? A dictionary definition of recognise is: 'to know again; to identify as known or experienced before; . . .'. Recognition depends first on knowledge; in order to be able to recognise something, you need to know what that something is, to have a definition that enables the 'something' to be distinguished from all other possible things.

A pattern is not just a single, unique thing but an exemplar of a group of things. Learning a pattern implies learning a description which enables members of this group to be identified. Unless the group concept is very tightly defined, learning an adequate description will involve collating knowledge gained from a number of examples. Pattern recognition, then, implies learning descriptions of conceptual groups from sets of examples, then applying these descriptions to enable the correct classification of new examples.

The pattern to be recognised can be a visual pattern, a sound pattern or a more abstract pattern such as a winning position in a game of chess, a type of mathematical equation or the pattern of characteristics that distinguish the members of a biological species. Computer pattern recognition has found numerous applications, including robot vision systems, speech recognition systems, theorem provers and medical diagnosis systems.

2.2. CATEGORISATION.

The learning and utilisation of categories is one of the basic processes of human thought. There are estimated to be more than seven million discernible colours; since it is obviously impractical to have this many different colour names, we group them into a very much smaller number of categories to enable us to refer to them (Bruner et al., '56). Similarly, we group people into the categories 'men' and 'women', 'adults' and 'children'; we group buildings into 'houses', 'shops', 'offices' etc. The categories we use depend on our language, our culture and our experience; they are to some extent individual, but if our category names are to be used for communication purposes there has to be some consensus about what defines a member of a particular category.

Categories can themselves be categorised, for example as affective, functional or formal (Bruner et al.). The members of an affective category evoke a common affective response. An old parlour game involves guessing the person someone is thinking of by asking them questions such as: "If this person were an animal, what animal would they be? If they were a colour, what colour would they be?" The person, the animal and the colour belong to the same affective category. Such categories are difficult to define, and frequently nameless. Functional or utilitarian categories are easier to define: they consist of objects which fulfil a specific task requirement. Formal categories are still more tightly defined, by a specific set of attributes which their members possess. There are close links between these types of category; an affective or functional category can be converted into a formal category by developing rules which specify what its members have in common.

What is needed for pattern recognition by computer is just such a set of rules; the pattern to be recognised must be developed into a formal category. Determining the form and content of the rule set is the first major problem facing the designer of a recognition system. Moreover, the pattern attributes contained in the rule specifications must be ones which can be assessed readily from the information input to the recognition system. Herein lies a second major problem: where the input is, for example, a visual image, the extraction of the required attributes may be a far from elementary task. A large amount of preprocessing may be necessary to obtain a description of the object to be recognised in an appropriate form for presentation to the recognition program.

2.3. DESIGN AND RECOGNITION - FORM AND FUNCTION.

Consider the problem of defining the category 'car'. What is a car? We cannot say that a car must have a roof, a certain number of windows and four wheels, as we can find examples of cars which do not have these attributes. Our description of a complex artifact like this is likely to be couched in terms of function rather than form.

Functional definitions of categories form one of the start points of the design process. The fundamental problem in design can be described from a functional viewpoint as:

Given: a specification of functions which are required and functions which can be provided

Find: a constructed structure from within the design task environment which satisfies the specification (and possibly certain restrictions).

(Freeman & Newell, '71), where the task environment consists of a set of structures and a set of functions such that:

1. Each structure provides certain functions.
2. For each function it provides, a structure requires certain other functions to be provided.
3. A functional connection can occur between two structures if one provides a function provided by the other.
4. A constructed structure consists of a set of structures and functional connections between them, and provides/requires the functions provided/required by its components and not involved in the functional connections.

The recognition problem can be seen as the reverse of this design problem: for recognition, the constructed structure is given and its functional specification is to be identified and categorised.

A purely functional approach to either design or recognition has several limitations. Not all objects can be described adequately in terms of function; in many domains a mixed approach, such as Essence descriptions which contain both functional and spatial information (Fretwell et al., '87) may be more appropriate. Where a functional description of an object can be given, it may not be possible to relate the function of the whole object to the functions of particular sub-parts, or to derive the object's function from the functions of sub-parts. It is also not easy to define a general-purpose set of basic functions from which descriptions of a wide range of objects could be built up - a necessity if methods are not to be restricted to use in very limited task domains (Di Manzo et al., '85).

If the recognition process begins with visual data, relationships between form and function need to be established. The mapping of form to function is many-to-many - most functions can be provided by more than one form, and most forms can perform more than one function, so the space of possible solutions to a problem will generally be large. Reasoning between structure and function in the domain of hand tools is described in (Brady et al., '84).

2.4. SIMPLIFYING THE PROBLEM: DISCRIMINANT DESCRIPTIONS.

The pattern recognition problem can be made simpler by considering discriminant descriptions rather than characteristic descriptions. A characteristic description is one which distinguished objects in a given category or class from all other possible objects; a discriminant description describes one class of objects in the context of a fixed set of other classes of object (Dietterich & Michalski, '79). Characteristic descriptions are used to answer questions such as "Is this a car?' or, more generally, "Is there a car in this scene?"; discriminant descriptions are used to answer questions such as "Is this car a Ford or a Vauxhall?".

When discriminant descriptions are used any attributes which are shared by all the categories under consideration can be ignored, so these descriptions are generally simpler than characteristic descriptions. The existence of a fixed range of options also allows the use of elimination techniques to aid identification, and of 'best fit' methods of categorisation.

A common way of simplifying the recognition problem is to consider

an artificial world, e.g. the Blocks world (Winston, '75), where there is only a limited range of possible objects which can be distinguished by the values of a small number of different attributes. Each combination of possible values of the attributes defines a pattern class; a concept is represented by one of these classes, or by a conjunction of several classes. Recognition here is effectively a discrimination problem.

Recognition in the Blocks world is an artificial problem, but there are many real-world problems which can be tackled by using discriminant descriptions, text recognition being an obvious example. The study of discrimination problems can also prove useful in the development and testing of techniques for later application to more difficult characterisation problems.

2.5. REPRESENTATION ISSUES - LAYERED DESCRIPTIONS.

One of the basic requirements of any rule-based pattern recognition system is a representation system, to be used for both the rules which define category membership and the descriptions of objects to be categorised. The representation system must incorporate terms to represent the (physical and/or functional) attributes of an object and, if appropriate, some say of representing structural information i.e. the relationship between the components of an object. Structural information can be incorporated in the vocabulary of the representation language, or be conveyed by its syntax.

Each term in a definition can itself be defined. A tetrahedron is a solid with four faces, each of which is a triangle. A triangle is a planar

figure with three sides, each of which is a straight line. A straight line is .... The definition process must, of course, stop somewhere - with a set of fundamental concepts, or primitives, whose definitions are assumed to be known (Sowa, '84). The vocabulary of the representation language could consist of just these primitives, more elaborate concepts being expressed in terms of these, but the use of such a restricted language would make descriptions of all but the simplest objects extremely long and complex. Incorporating terms representing higher-level concepts into the language will allow shorter definitions to be formulated, at the cost of increasing the size and complexity of the representation language itself.

One way of using high-level terms while keeping the representation language simple is to layer the representation. A bottom layer description is formulated in terms of primitive concepts; this is then used as the starting point for the formulation of another description in terms of rather more advanced concepts, which can be used in turn to develop yet another description, and so on. Each term in a description generally replaces a group of terms in the previous description, so the length of the description decreases as the complexity of the terms from which it is composed increases. As each layer uses its own vocabulary, which need not, and in general does not, include all the terms in the previous layer's vocabulary, the overall size of the vocabulary does not increase unduly as the process advances.

Layered techniques were not developed solely to allow the use of different representation systems within a single pattern recognition system; one of the early goals of Artificial Intelligence research was to find ways to replicate the working of the human brain by layers of neurons, which led to the development of neural nets and layered threshold

mechanisms with very similar representational mechanisms for all layers. (Ashby, '60; Arbib, '72).

A layered approach to the analysis of visual images was pioneered by Marr (Marr, '79; Garnham, '87). He carried out the computation of 3-D models from grey-level descriptions in three main stages, employing two intermediate levels of representation: the primal sketch, which represents the significant intensity changes in the image, with tokens standing for regions and their boundaries, and the $2^1/_2$ D sketch, which represents the orientation and approximate distance from the viewer of surfaces.

There is a difference between moving from descriptions of individual components to a description of a composite object, and moving from a general object description to a more specific object description: going from 'four straight lines' to 'quadrilateral' is not the same as going from 'quadrilateral' to 'square'. There are two different hierarchies in operation, characterised by the relations 'is-a-component-of' and 'is-a-kind-of'. The way in which these two hierarchies tie in with a layered approach to object description is not entirely straightforward. They can be depicted as trees, the 'is-a-kind-of' tree having a genus or supertype as its root, with branches leading to subtypes, and the 'is-a-component-of' tree having a composite object as its root, with branches leading to components. (See Figures 2.1, 2.2). A move up from one level of description to the next corresponds to a move down through the 'is-a-kind-of' tree, or a move up through the 'is-a-component-of' tree.

(Smith & Medin, '81) describes three ways of viewing concept descriptions. The classical viewpoint considers a concept to be defined by

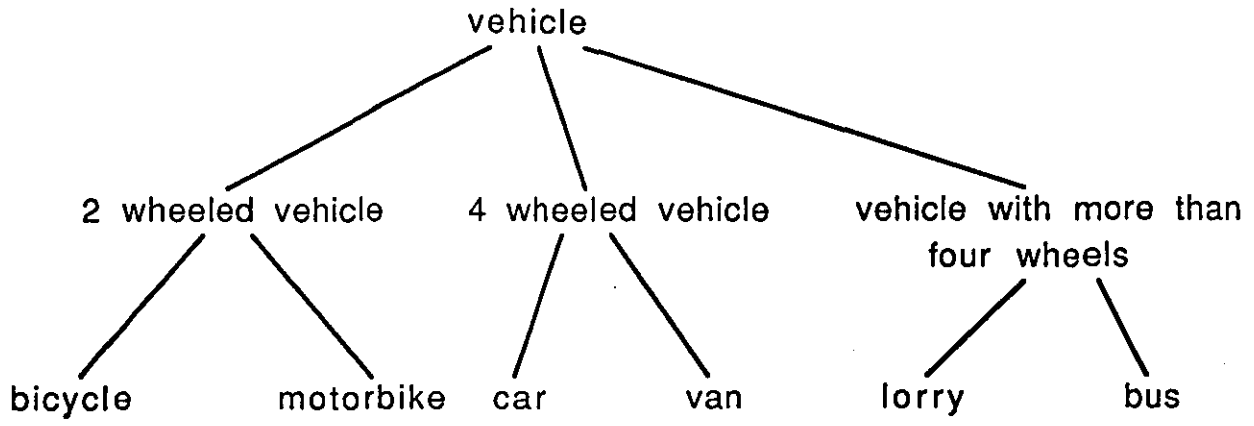


Figure 2.1. An 'is-a-kind-of' tree.

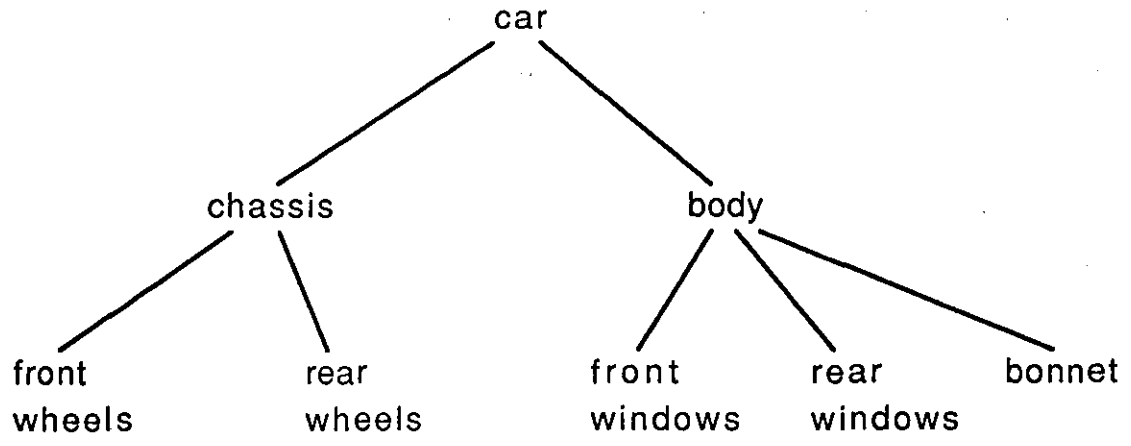


Figure 2.2. An 'is-a-component-of' tree.

its genus or supertype and a set of necessary and sufficient conditions that differentiate it from other species of the same genus; the probabilistic viewpoint is that a concept is defined by a collection of features, and everything that has a preponderance of those features is an instance of the concept; the prototype viewpoint considers that a concept is defined by a characteristic prototype, and an object is an instance of the concept whose prototype it resembles most closely. These views all seem to be concerned more with discriminant descriptions than with characteristic descriptions, and help to show how the distinction between

characteristic and discriminant descriptions relates to the layered approach to description. With discriminant descriptions, the top layer - the supertype - has already been fixed; the recognition task involves identifying the correct element of the layer below this.

Much of the information which is shown explicitly in low level descriptions is conveyed implicitly by higher level descriptions. Moving from inter-related components to a composite object is equivalent to making structural information implicit; moving from a supertype to a subtype means making non-structural information implicit. It may be advantageous for the description process to be halted at a stage where information which is important for recognition is still explicitly stated.

The information implicit in a given type can be regarded as a list of properties associated with the type. The idea of a hierarchy of types is one of the fundamental concepts underlying the design of the object-oriented programming language SMALLTALK (Goldberg & Robson, '89); every object in SMALLTALK is an example of a type which occupies a clearly defined position in the type hierarchy, and each of these types has a set of properties associated with it. SMALLTALK supports inheritance of properties, i.e. every subtype automatically has all the properties associated with its supertype, so when a new subtype is defined only those properties which are exclusive to it need be specified. The use of inheritance thus reduces the size of property lists. As a type is fully defined by its properties, the property list can be used for recognition purposes: an example of a supertype which has all the properties of a given subtype must be an example of that subtype.

2.6. REPRESENTATION SYSTEMS.

The task of selecting an appropriate representation system can be compared with the task of choosing a programming language for a particular application. In a sense all programming languages are equivalent, in that they all ultimately instruct the computer to carry out the same basic operations, but as different languages have been designed to suit different purposes the wrong choice can make the programmer's task infinitely harder. Similarly with representation systems; the critical factors to be taken into account when making a choice are different, but the same principles apply.

A wide variety of different representation systems have been used for pattern recognition and machine learning. These include predicate calculus, production rules, hierarchical descriptions, semantic nets, frames and scripts (Dietterich & Michalski, '83). Usually the same representation is used for both the input data and the rules, but this is not universal (Forsyth & Rada, '86). Those described here are all general-purpose; there are also a number of systems which have been developed for use in specific task domains, such as that developed by Buchanan for Meta-DENDRAL (Buchanan, Feigenbaum & Lederberg, '71).

2.6.1. Feature vectors.

The simplest type of representation for input data, the feature vector is an array of numbers which characterise the state of different attributes of an example. It can be used together with several different types of rule format; for example, ID3 (Quinlan, '82) uses feature vectors to induce a decision tree. It is only suitable for problems where examples can be

described fully by a limited set of attributes, each of which can take only a small number of different values. It is most commonly used where no structural information is involved, though it is possible to transform structural descriptions into feature vectors (Wysotzki, Kolbe & Selbig, '81).

2.6.2. Predicate calculus.

First-order predicate calculus (FOPC) is a logic system devised by the German mathematician Frege. Sentences, or well-formed formulae, in FOPC are made up from predicates which take one or more arguments, constant terms, variables, logical connectives (the Boolean operators) and universal and existential quantifiers, following clearly defined formation rules. Inference rules determine how one formula can be derived from others. (Garnham, '87).

Predicate calculus has several important advantages as a knowledge representation system: it is a language with machine-independent semantics, it forms the basis of the logic programming language Prolog, it enables the use of bottom-up and top-down problem solving (by performing resolutions on the left-hand and right-hand clauses in a formula) and it contains uniform proof procedures that can prove any true theorem in finite time. However, it also has some drawbacks: FOPC does not allow the quantification of predicates, and it does not support certain types of human reasoning such as the use of defaults (Gabrielides, '88). Consequently though some systems, for example Shapiro's Model Inference System (Shapiro, '82) employ pure FOPC, others such as Michalski's INDUCE (Dietterich & Michalski, '79) use extensions to it.

2.6.3. Conceptual graphs.

Conceptual graphs have evolved from the existential graphs which Pierce used as a notation for logic; they are a variant of first-order logic which enjoy the advantages of having a direct mapping to and from natural language and having direct extensions to modal logic and other forms of reasoning.

A conceptual graph is a finite, connected, bipartite graph composed of nodes linked by arcs. There are two different types of node: one type, normally drawn as a box, is used to represent concepts and the other, drawn as a circle, is used to represent conceptual relations. An example is shown in Figure 2.3.

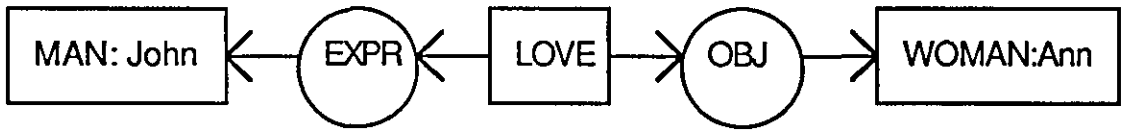


Figure 2.3. A conceptual graph.

Winston's concept learning system (Winston, '75) uses a graph representation where concepts are represented by circles and relationships are indicated by labelled arrows connecting these circles.

2.6.4. Frames.

A frame is a data structure which can be used to describe either classes or instances. It is a network of nodes and relations, which

represents things which are known to be true about the class and has terminals or slots to which assignments may be made. Markers may be used to specify conditions to be met by assignments, e.g. ranges of expected values, default values; procedures may also be attached to slots to drive the problem-solving behaviour of the system (Minsky, '75).

Collections of related frames may be linked together into a frame system, where the effects of actions are mirrored by transformations between the frames of the system, For visual scene analysis, for example, the different frames may represent a scene from different viewpoints. One image understanding system based on frames, FABIUS (Rosin, '88), represents objects and their subcomponents by hierarchies of frames.

2.7. ALTERNATIVE APPROACHES TO RECOGNITION.

Classical machine-vision techniques involve extracting progressively higher-level descriptions of the whole of a visual image. Only when a complete top-level description has been derived is recognition attempted. This bottom-up, breadth-first approach is computationally expensive and time-consuming, and its success depends on sufficient information being present in the original image. Studies of the way in which humans recognise objects and the types of information which they utilise have led to the development of alternative approaches which seem to be more suitable for many computer pattern recognition problems.

It is often possible to recognise an object from just a vague outline or a description of its general structure - most people would have no difficulty in identifying the objects in Figures 2.4 and 2.5 as a car and a

person respectively. With a more detailed image, a preliminary identification of an object can be effected by considering its outline or structure, then this identification can be confirmed by checking that the details are consistent with it. Alternatively, a tentative recognition could be based on the identification of a small detail - a particular configuration of lights on a car, or a manufacturer's logo - and the way in which this detail relates to the object as a whole could be studied to provide confirmation.

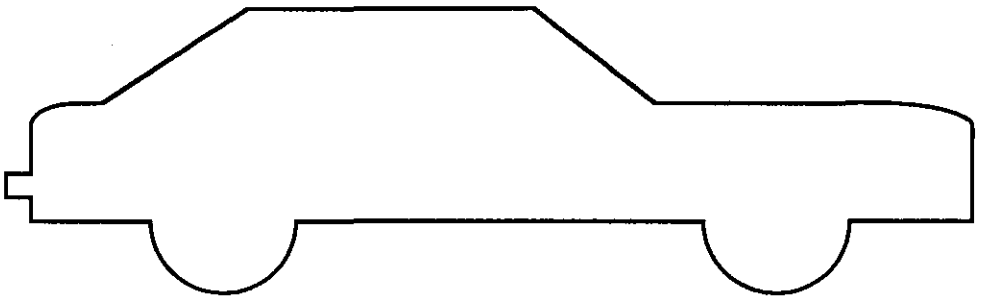


Figure 2.4. An image recognisable as a car.

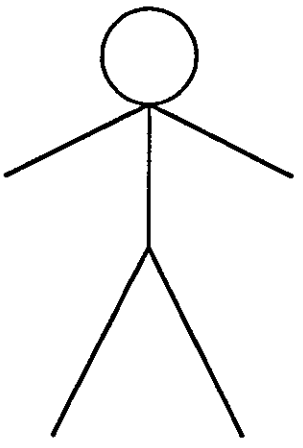


Figure 2.5. An image recognisable as a person.

Background knowledge about what objects are likely to be found in a given location, which objects tend to occur together, what spatial relationships can be expected to hold between them etc. can sometimes prove useful. For example, the small blurred shape shown in Figure 2.6 is

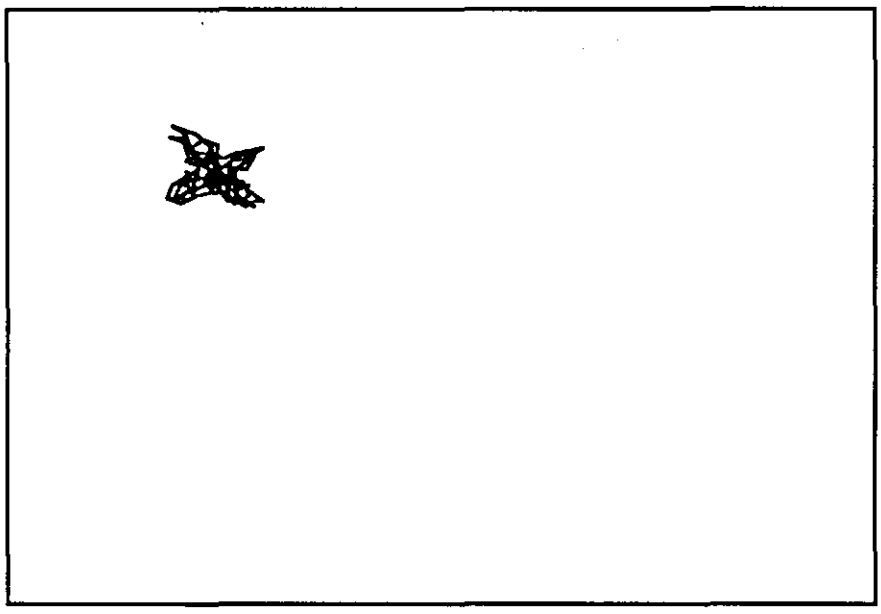


Figure 2.6. Recognition requiring background knowledge.

impossible to identify if considered in isolation, but if we are told that the surrounding area has been identified as sky, our knowledge of what objects are commonly observed in the sky allows us to propose two possible identifications: it could be a bird or an aeroplane.

Many systems use ideas like these as part of a combined bottom-up and top-down approach. Often the image is analysed bottom-up to a certain level, then a top-down technique is used to select promising areas for a more detailed depth-first search.

One vision system which uses a combination of bottom-up and

top-down techniques is Ohta's recogniser of outdoor colour scenes (Ohta, '85). Here the bottom-up process generates a plan of the large areas (patches) in the image. The top-down process fixes the interpretation of the large patches, then attempts to interpret smaller patches in the context of these. Whenever the top-down process makes a decision which may affect the interpretation of the whole scene, for example fixing the position of the horizon, the bottom-up process reevaluates its plan.

2.8. SUMMARY AND CONCLUSIONS.

Pattern recognition is in general a complex problem; if it is to be solved successfully it may need to be simplified and/or to be split up into a number of sub-problems.

The first step in the recognition process is the formal definition of the problem: the development of sets of rules which define membership of the pattern categories under consideration. A representation language must be selected for these rules. Ideally the hypotheses of the rules should be represented in the same language as the input data, so that the rules can be applied directly to the data, but often this will not be possible - one cannot, for example, define visual patterns such as the objects in a typical road scene in terms of the pixel values in which visual input data is normally supplied - so a layered approach will have to be adopted, using one or more intermediate levels of representation. Additional sets of rules will then be required to define the terms used in each representation layer with respect to the terms used in the preceding layer.

The development of rule sets may be time-consuming and difficult. The rules may be formulated by a human expert, or induced from examples of the behaviour required of them (see Chapter 3), or a combined approach may be adopted: formulate approximate rules, then refine them by testing their performance in classifying examples.

When the representation system(s) and rule set(s) have been determined, the recognition task will still not necessarily be straightforward. Ambiguities or inadequacies in the data or rules may make smooth progress from the input data through any intermediate levels of representation to a unique identification impossible. It may be necessary to employ background knowledge about the problem domain to guide the recognition process, to use interpolation or default values to deal with gaps in the data, and to use symbolic or numerical methods to handle the uncertainties which arise (see Chapter 5).

Despite the difficulties, a number of systems have been developed which work well in limited problem domains such as the identification of soybean diseases (Michalski, '78), the analysis of chess end-game positions (Quinlan, '82) and the understanding of speech using a limited vocabulary (Erman et al., '88). However, this is an area where there remains a great deal of scope for further research.

CHAPTER 3

RULE INDUCTION

3.1. MACHINE LEARNING - AN OVERVIEW.

Machine learning is one of the most important topics of modern Artificial Intelligence research. It has three main aims: to enable computers to perform intelligent tasks so that people need not do them, to get computers to do things that people cannot do (or cannot do fast enough), and to simulate human thought processes and thus help to increase our understanding of them (Simon, '83).

Learning can be defined as anything which produces adaptive changes in a system which enable it to perform the same task, or tasks drawn from the same population, more effectively or more efficiently next time. It is not just the acquisition of new knowledge; this definition of learning makes it clear that is not the mere possession of knowledge but the ability to apply it which is important.

Machine learning systems can be classified by the ways in which they represent knowledge, by their domains of application or by the learning strategies which they employ. The learning strategy depends on the amount of inference which has to be performed to transform the knowledge with which the system is supplied into a form in which it can be used effectively. At one extreme are systems which do not have to perform any inference because their input data has been carefully selected and organised by a teacher, and has only to be stored in memory and then recalled when required. At the other extreme are systems which infer all the knowledge they need to perform their task from raw, unprocessed data. Most systems fall somewhere between these two poles - they are supplied

with data which has been pre-processed to some extent, but which requires further modification before it can be applied effectively.

The tasks which a learning system has to perform may include: translation from an input language into an internal representation language, imposing a logical ordering on facts, integrating new data with existing knowledge, inducing rules, drawing analogies, interpolating data, generating queries and finding and resolving inconsistencies. The particular combination of tasks which a system performs is the basis of this classification of learning strategies (Carbonell, Michalski & Mitchell, '83):

- learning by rote - no significant inference is performed.
- learning by instruction - the learner may perform some translation, organisation and integration of knowledge.
- learning by analogy - facts and skills are transformed and/or augmented for application to new tasks.
- learning from examples - the learner induces concept descriptions from pre-classified examples.
- learning from observation and discovery - a general form of unsupervised inductive learning and theory formation.

The classification of strategies is not a rigid one, as some strategies combine elements of several different approaches to learning and the distinctions between different approaches are often somewhat blurred. To illustrate the fuzziness of the boundary between rote learning and rule induction, consider a pattern recognition system which stores descriptions of patterns and their classifications for use in classifying new patterns. Some logical ordering must be imposed on the stored information to enable facts to be retrieved from memory quickly and easily. A natural ordering would be the grouping together of descriptions which have common

elements. Where all the instances in a group have not only a common set of descriptors but also a common classification, it would seem reasonable to infer that this set of descriptors implies the classification. A rule could be formulated:

<descriptor set> implies <classification>

which could then be used to classify new examples. Rule induction can thus be seen as a natural extension of the imposition of a logical ordering on data; a rule set is, at its simplest, just a succinct way of representing information about a group of examples.

Some machine learning programs produce 'black-box' programs which perform the required task without making it clear how they are doing it, or what rules they are employing. They generally have a mathematical bias; their knowledge may be encoded in the form of a covariance matrix or an optimised set of coefficients, which will mean little or nothing even to an expert in the field in which they are designed to operate (Forsyth & Rada, '86). Black box systems can be very efficient, but they have the drawback that if errors creep in, for example because of some inadequacy in the set of training data, they are almost impossible to find. The fact that such systems are unable to offer any coherent explanation as to how their conclusions have been reached may also reduce the reliance which people are prepared to place on their results.

3.2. RULE INDUCTION IN PATTERN RECOGNITION.

We can all recognise a car when we see one, but would find it hard to codify the knowledge which enables us to do this. The difficulty of establishing rules for the solution of 'real world' problems is one of the main stumbling blocks in the development of pattern recognition programs

and Expert Systems; (Michie, '86) calls it 'the Feigenbaum bottleneck' after Edward Feigenbaum, one of the pioneers in this field (Buchanan,Feigenbaum & Lederberg, '71; Feigenbaum, '83). The problem of extracting rules from human experts can be bypassed if computers can be enabled to induce rules for themselves.

Induction can be regarded as the reverse of deduction: given a rule, deduction leads to the results of its application; given a set of results, induction leads to a rule which could have produced them. But induction is a less certain process than deduction; if a rule is known to be true then deductions made from it can be firmly relied upon, but an induced rule can be similarly relied upon only if the set of facts from which it was induced is complete, i.e. if it contains every fact which could be deduced from the rule. In pattern recognition, rules are generally induced from incomplete training sets, which is a powerful technique but one which needs to be handled with caution. The risks are particularly great when rules are induced from only positive examples of their behaviour; where there is nothing to indicate how a rule should not behave, i.e. which patterns do not belong to the category being defined, there is a clear danger that the induced rules will be excessively general.

On the positive side, the use of rule induction makes programs less domain specific, more easily adapted to other areas of application. If a program to recognise instances of a particular pattern has been based on a set of preformulated rules, developing another program to solve a different, but related, recognition problem will involve a great deal of rewriting; if, however, the original program has been based on an induction algorithm which will derive the rules from a set of examples, all that has to be done to convert the program is to replace the original example set with a new example set.

Although the main practical applications of rule induction systems are in real world situations, many algorithms have been developed through work on comparatively simple pattern recognition problems where the 'correct' rules can be determined easily. This has the advantage of allowing the efficiency of the algorithm to be assessed before it is applied to more complex tasks. Common simplifications include studying regular polyhedra (the 'Blocks World'), photographing objects against a plain background rather than in their normal settings, and restricting training examples to pictures of single unoccluded objects.

3.3. REQUIREMENTS FOR RULE INDUCTION.

(Michalski, '83) gives the following general paradigm for inductive inference:

Given: (i) observational statements that represent specific knowledge about some objects or processes

(ii) a tentative inductive assertion (which may be null)

(iii) background knowledge that defines assumptions and constraints, and any relevant problem domain knowledge including preference criteria for solutions

Find: an inductive assertion that weakly implies the observational statements and satisfies the background knowledge.

When the inductive assertion being sought is a set of concept descriptions, the observational statements required will take the form of a training set of instance descriptions together with their classifications. Where a single concept description is being sought, the classifications will be 'positive' and 'negative' if the training set includes both examples of the concept, and counter-examples; if only positive examples are used,

the classification will be implicit.

The task of finding the required concept description(s) can be viewed as a search through a space of possible descriptions, and the presence or absence of an initial tentative description helps to determine the direction of the search: top-down or bottom-up.

Background knowledge about the problem and problem domain may be used explicitly during the induction process to guide the search for rules and prune the search tree, or may be employed only at the setting-up stage to assist in the selection of an appropriate training set, representation language(s) for the training set and rules, and induction algorithm.

A wide range of different rule induction algorithms have been derived, for use in a range of different task domains. The selection of a suitable algorithm for a particular task will depend on a number of different factors; (Ross, '89) gives the following list, which is by no means exhaustive:

- are all the examples available immediately, or is the learning to be done incrementally?
- is the order of the training examples significant?
- how reliable are the example classifications?
- are all the attributes known to be relevant?
- is anything known about the relationships between any attribute values?
- is there any significance to correlations between the values of different attributes?
- can the program generate and test new examples?

To these, one could add:

- does the example set consist entirely of positive examples, or of positive and negative examples, or positive examples and 'near misses'?

- are the rules to be induced known to be conjunctive, or might they be disjunctive?

The choice of algorithm will clearly be less restricted if the system designer can select the composition, ordering and representation of the training set than if any of these aspects of the set are fixed in advance.

3.4. EVALUATION CRITERIA FOR INDUCTIVE ALGORITHMS.

As inductive algorithms have been developed to solve a very wide range of different problems, it is not possible to give a direct comparison of their efficiency by showing how well each of them can solve one particular problem. Most comparative studies have concentrated instead on specific issues - for example, (Dietterich & Michalski, '83) compared five different systems for determining a characteristic description of a single concept using the following evaluation criteria:

1. Adequacy of the representation language used.
2. Rules of generalisation implemented.
3. Computational efficiency (estimated from hand simulations of the methods on a very simple problem).
4. Flexibility and extensibility.

(Bundy, Silver & Plummer, '85) used a rather different approach, analysing seven different rule- and concept-learning programs with the aim of extracting and explaining the techniques used, identifying the range of applicability of each technique and establishing the relationship between different techniques designed to accomplish the same task. Other authors have simply described a range of different systems, leaving the reader to

draw comparisons between them (Quinlan, '82; Gabrielides, '88). The approach adopted here is to examine the field subject-by-subject rather than system-by-system.

3.4.1. Fields of application.

The fields in which learning systems have been used include:

- agriculture
- physics and chemistry
- cognitive modelling
- computer programming, expert systems
- education
- game playing
- image recognition and speech recognition
- mathematics
- medical diagnosis
- music
- natural language processing
- physical object characterisation
- planning and problem-solving
- robotics
- sequence prediction.

(Carbonell, Michalski & Mitchell, '83). Some systems are very domain-specific, in that the learning algorithms they use incorporate in their design a great deal of background knowledge about the problem domain; others are more general-purpose, using either very little domain knowledge or knowledge which can be separated easily from the learning algorithm and substituted with knowledge about a different domain if required. The systems which are most worth studying are obviously those which are either specific to the domain under consideration (in this case,

visual pattern recognition) or general-purpose; however, a brief study of systems specific to other domains is also worthwhile as these may contain some interesting new techniques which could be adapted for use elsewhere.

A classic example of a domain-specific program is Meta-DENDRAL (Buchanan, Feigenbaum & Lederberg, '71), which infers rules for the analysis of chemical data from a mass spectrometer. This program uses a three-stage process, firstly explaining the experimental data from each substance, then generalising the results, and finally organising the generalisations into a unified theory. The detailed implementation employs a great deal of domain knowledge, but the same overall plan could be used in other fields where considerable amounts of data from different problem instances require analysis.

Shapiro's Model Inference System (MIS) (Shapiro, '82) began life as a debugger for Prolog computer programs, but was developed into an automatic program synthesizer. This is much less domain-specific than it may at first appear, as programs could be synthesised for use in a range of different domains. One of the examples given of its use, however, the inference of a context-free grammar, employs domain-specific knowledge in the design of a specialist refinement operator.

The idea of allowing packages of domain-specific knowledge to be incorporated into a general-purpose system is exemplified by the STAR methodology (Michalski, '83). This uses as its description language a form of annotated predicate calculus, where each descriptor is assigned an annotation containing relevant background knowledge such as its domain and type, its relationship with other descriptors, the type of objects with which it can be used and the operators which are applicable to it. The

method allows the use of task-specific generalisation rules and preference criteria.

An example of a general-purpose system is ID3 (Quinlan, '82), which produces a decision tree for classifying sets of instances described by feature vectors, using either a cost basis or an information-theoretic approach to select attributes on which to partition the training set. ID3 was originally used for the analysis of chess end-games, but uses no knowledge specific to this problem domain. However, the approach used places fairly severe restrictions on the types of problem to which it can be applied: it has problems in dealing with incomplete and/or uncertain data (Hart, '86), and all tests have to be in the form of a comparison between a single variable and a constant (Forsyth & Rada, '86). Descendants of ID3 such as NEDDIE (Kodratoff et al., '88) incorporate such improvements as the use of a chi-squared test of attribute reliability, and termination if the overall reliability falls below a threshold, to circumvent some of the limitations of the original program.

3.4.2. Sources of input data.

Two main types of inductive learning can be distinguished: learning from examples, or concept acquisition, which aims at producing descriptions for classifying objects on the basis of their attributes or properties, and learning by observation, or descriptive generalisation, where the goal is to determine a general description characterising a collection of objects or observations. Concept acquisition includes the learning of both characteristic and discriminant descriptions of classes of objects, and the inference of sequence extrapolation rules; descriptive generalisation covers such problems as theory formation, discovering patterns in observational data and the determination of taxonomic

descriptions (Michalski, '83).

The distinction is really between learning to distinguish concepts which are known to the teacher, from preselected examples, and discovering concepts which have not been identified in advance. Generally the machine learning aspects of pattern recognition can be regarded as concept acquisition as the aim is to find some practical way of distinguishing examples of known classes.

AM, a system which develops mathematical concepts (Lenat, '83) is a good illustration of descriptive generalisation. This system consists of a set of primitive mathematical concepts together with a large number of domain-specific heuristic rules which guide it in deciding which areas to explore, defining new concepts, recognising simple relationships between concepts and estimating how interesting each concept is. It has no 'target' concepts; it simply aims at maximising the interest ratings of the concepts it discovers.

Learning from examples can be subdivided into different categories according to the source of the examples: a teacher, the external environment or, in some cases, the system itself. Systems designed to use examples supplied by a teacher typically incorporate these examples one at a time into a rule set or concept description which becomes progressively more accurate as the number of examples used increases. Winston's concept learner (Winston, '75) operates in this way: it uses a positive example of the concept to be learnt to form an initial model which is then modified through comparison with each new example or near-miss. Hayes-Roth's SPROUTER (Dietterich & Michalski, '83) follows a similar pattern.

The performance of systems which build up concepts incrementally by considering examples one at a time may be substantially affected by the order in which the examples are presented (McGregor, '88), so where it is impossible or impracticable for the examples to be ordered by a teacher, for example because data is being obtained directly from the external environment, a system which deals with a large set of examples at once may be more appropriate. This set of examples can then be ordered or grouped as required by the system itself.

Loisell and Kodratoff describe one method of grouping examples according to resemblances between them; they classify differences revealed by comparison of examples as highly ambiguous, ambiguous or discriminant near misses, then use these to rate the examples as highly comparable, comparable or separable, This produces a way of partitioning the example set so that examples not in the same subset show a maximum of differences. (Loisel & Kodratoff, '81).

Mitchell's Version Space strategy (Mitchell, '79) works by representing the set of all concept descriptions consistent with the observed training examples, using one example at a time to reduce the size of this version space. This system can select the next example to consider from a set of possibilities by considering which example comes closest to matching half of the descriptions in the version space, thus imposing its own ordering on the example set.

Instead of merely imposing their own ordering on an existing set of training examples, some systems have the ability to generate examples for themselves to resolve any ambiguities they discover. This method can, of course, only be used if a teacher or oracle is available to provide the correct classifications for the examples generated. Shapiro's MIS has this

facility, as does Sammut's concept learner (Sammut, '81), which starts with a positive instance of the concept to be learnt, generalises this and then tests the validity of its generalisation by generating new examples which satisfy it.

3.4.3. Search Patterns.

Rule induction can be regarded as a search problem; the induction program must search through a space of possible rules in order to find the correct one(s). The rule space is partially ordered by the 'more-specific-than' relation, which imposes a tree structure on it (Mitchell, '79). The search can be conducted in several different ways: depth first, breadth first, specific-to-general, general-to-specific or a combination of these.

Systems which employ a depth-first search strategy maintain a single current hypothesis, modifying or replacing this if a contradiction is discovered i.e. they test all the training examples against each rule in turn until a satisfactory rule set has been discovered. Breadth-first strategies, on the other hand, maintain a set of hypotheses, eliminating elements of this set when contradictions arise i.e. they test all the rules against each training example in turn. A breadth-first search has the advantage that training examples need not be retained once they have been examined; the drawback, though, is that for any reasonably complex problem the set of possible rules will be vast. The impracticability of formulating and examining every possible rule explains why the majority of systems, including Shapiro's MIS, Quinlan's ID3 and Winston's concept learner, operate depth-first.

A compromise approach is the beam search where the nodes at each

level of the search tree are pruned, only a limited number being retained for use in propagating new nodes. Dietterich & Michalski's Induce 1.2 uses this technique: it starts by selecting a random subset of the training examples to form its initial set of concept descriptions, generalises each description in this set by a single application of each of its generalisation rules in turn, then prunes the set of generalisations to a predetermined size, retaining those descriptions which are least complex and cover most examples. Any descriptions which cover enough of the examples are entered in the final rule set; the remaining elements of the pruned set are further generalised, until enough rules have been found. This system is quite efficient but not optimal, as generalisations which would lead to good descriptions may be pruned. (Forsyth & Rada, '86).

Specific-to general search strategies typically start with an initial hypothesis formulated from a single positive example of the concept being learnt, then generalise this to cover further positive examples. General-to-specific strategies start with a very general rule, then restrict it to eliminate negative examples. The majority of systems use training sets containing both positive and negative examples, and so use a combined approach: if the current rule set correctly classifies an example, no alteration is made to the rules; if the system fails to classify a positive example, or incorrectly classifies it as negative, the rules are made more general; if the system incorrectly classifies a negative example as positive, the rules are made more specific.

Rather than modifying the same working hypothesis in two different directions, depending on the types of error which are encountered, Mitchell's version space strategy (Mitchell, '79) and Young, Plotkin & Linz's focussing technique (Bundy et al., '83) both maintain two separate hypotheses, one overly general and the other overly specific. Positive

examples are used to make the specific hypothesis more general, and negative examples to make the general hypothesis more specific, until the two coincide. This bidirectional approach is an interesting idea, but unfortunately it can only be applied breadth-first (if a depth-first search were to be used, the two hypotheses would not necessarily coincide eventually, but could well bypass one another), which severely restricts the range of problems for which it can be used.

3.4.4. Rule Modification Techniques.

Most rule induction programs use the following main control loop:
Until the rules are satisfactory:

1. Identify a fault with a rule
2. Modify the rule to remove the fault.

The part of the system which identifies faulty rules is the critic; the part which modifies the rules is the modifier (Bundy et al., '83).

The critic will be activated when the rule set fails to classify a training example correctly, i.e. when a wrong classification has been obtained or when the program has failed to produce any classification. Where examples are classified by the application of a single rule, identifying the faulty rule poses no problems; where a sequence of rules have been applied, it will be necessary to examine a trace of the classification process in order to locate the fault.

The problem of identifying faulty rules has been analysed by Shapiro. He identifies three types of fault in a Prolog rule set: termination with incorrect output, termination with missing output and non-termination. His MIS deals with the first of these, incorrect output, by single-stepping through the trace, checking each rule until the fault is discovered, or by a

divide-and-query algorithm: check the rule which will divide the program trace into halves, then select the appropriate half on which to iterate the procedure. The algorithm which deals with missing output finds a goal which is uncovered, i.e. cannot be unified with the head of any clause. In cases of non-termination, the trace is examined to find two consecutive calls which breach the well-founded ordering determined by the maximum depth of any computation of a procedure (Shapiro, '82). The application of these critic algorithms requires the system to have access to an oracle or database from which the answers to queries can be obtained.

The methods implemented by the modifier fall into two main groups: generalisation or de-refinement techniques, and specialisation or refinement techniques. Specialisation is the opposite of generalisation, so the specialisation methods tend to be generalisation methods applied in reverse; for example, a rule can be generalised by dropping a condition from its hypothesis and specialised by adding a condition to it.

(Michalski, '83) distinguishes between two different types of generalisation rule - selective rules, where every descriptor in the generalisation is used in the initial example descriptions, and constructive rules, where new descriptors are used. His selective rules are:

- dropping a condition
- adding an alternative to a condition
- extending the range of values of a descriptor - for example, closing an interval
- climbing the generalisation tree, i.e. replacing a set of values of a structured descriptor with a value which is the lowest parent node of all the values in the set
- turning a constant into a variable
- turning a conjunction into a disjunction

- extending the domain of a quantifier e.g. replacing an existential quantifier with a universal quantifier

- inductive resolution

- the extension against rule: produces the most general statement consistent with a positive example and a negative example of a concept.

Not all of these rules are applicable to all problems. For example, the climbing the generalisation tree rule can only be used with structured descriptors, and the range of values of a descriptor can be extended only if the values are ordered in some way. The rules implemented by a particular system tend, therefore, to depend on the application for which it is being used.

Constructive generalisation techniques are even more application dependent as a considerable amount of background knowledge has to be used in order to derive new descriptors which are relevant to the problem. They typically involve identifying relationships and interdependence between descriptors, or common groupings of descriptors which can be replaced by a new combination descriptor. Few systems for learning from examples employ constructive generalisation; it is commoner in systems which learn by observation, such as the BACON system (Langley, Bradshaw & Simon, '83).

Specialisation methods are similarly application-dependent to some extent. Shapiro's MIS, which homes in on the correct rules by taking large steps in the specific-to-general direction, using the single rather crude generalisation technique of dropping the last conjunct from the hypothesis of a rule, then taking small steps back in the general-to-specific direction, uses a specific refinement algorithm for the inference of definite clause grammars. Its general refinement algorithm implements

these techniques:

- closing a clause
- instantiating variables
- unifying two input variables
- adding an output producing goal
- adding a test predicate.

3.4.5. Types of output: conjunctive and disjunctive rules.

The rule or set of rules which is the output of a rule learning program can be represented in a wide variety of different ways which are all logically equivalent to:

<Hypothesis> implies <Conclusion>.

In the case of single concept learning programs, the conclusion - membership of the concept class - is usually implicit. The hypothesis is the concept description.

The hypothesis may be conjunctive or disjunctive. A conjunctive hypothesis is of the form:

Condition1 and Condition2 and . . . . .and ConditionN.

Disjunctive rules can be written as a disjunct of conjuncts:

(Cond1a and Cond2a and . .) or (Cond1b and .Cond2b and . .) or . . .

or as a set of conjunctive rules, since the rule

A or B implies C

is equivalent to the rules

A implies C and

B implies C.

It is not true, however, to say that all rule sets with more than one element are essentially disjunctive. A set of (conjunctive) rules is conjunctive if all the rules in it have different conclusions, disjunctive

only if there exist two rules in the set with different hypotheses but the same conclusion.

Specialising disjunctive rules presents no particular problems; where a disjunctive rule incorrectly classifies a negative example as positive, the disjunct(s) requiring specialisation will be the one(s) which the example erroneously satisfies. When generalising a disjunctive rule, however, the situation is more complex. A positive example which has not been correctly classified by a rule could be covered by applying a rule of generalisation to any of the existing disjuncts in the hypothesis, or by creating an entirely new disjunct. Search spaces of possible disjunctive rules tend, therefore, to be substantially larger than search spaces of possible conjunctive rules.

The majority of rule induction systems develop conjunctive rule sets; there are few examples of effective disjunctive systems. One simple disjunctive algorithm, designed to handle correctly classified examples presented as sets of ordered attribute values, was developed by Ross:

> Select a positive example to form the initial version of the concept.
>
> Given a new positive example:
>
> > form the meet of one component of the concept disjunct and the new example, check this for validity against all negative examples. If valid, replace the component with the meet; if invalid, repeat with another component or if no more components remain to be checked, form a new component.
>
> Given a new negative example:
>
> > check against all components of the concept; eliminate any components which are invalidated and reprocess the positive examples which supported them.

(Ross, '89). Another disjunctive technique is 'refocussing', a modification

of Mitchell's focussing technique (Bundy et al., '85).

3.5. SUMMARY AND CONCLUSIONS.

Attempts to solve a wide variety of learning problems in many different fields have led to the development of a range of systems which have some common characteristics, but have been tailored to cope with different types of input and to provide different types of output.

The identification or development of a system to handle a particular task should be begun by analysing the possible starting points for the induction, and defining the output which is required in as much detail as possible. The starting points will be determined by the type, quantity and quality of data which is available, and the types of preprocessing which could be applied to it; the types of rule which are to be induced will depend on the task which the rules are to perform, i.e. the kinds of questions to which they will be expected to provide answers.

Specification of the input and output will place significant constraints on the choice of learning method, but there may be additional constraints as well: the type of equipment which is available and the speed with which the system is required to operate, for example. The large number of factors to be taken into account in selecting an appropriate system suggests that despite the wide choice of systems available, it will often be necessary to substantially modify an existing system or to develop a completely new one to deal with a new application.

CHAPTER 4

BLACKBOARD SYSTEMS

4.1. THE BLACKBOARD CONCEPT: HISTORICAL BACKGROUND.

The blackboard system is an attempt to model the behaviour of a group of experts collaborating to solve a problem; they cluster around a blackboard on which information about the problem is written, and each one selects the information which they can use, synthesises new information from it and places this on the blackboard to be used in turn by the others.

The basic idea is similar to that of Selfridge's Pandemonium model of human pattern recognition, developed in 1959 (Lindsay and Norman, '77). This feature analysis model employs several sets of demons: image demons record the initial signal image, feature demons search the image for particular characteristics, then cognitive demons, each of which is responsible for recognising one particular pattern, scan the output of the feature demons and start yelling if they find any of the characteristics they require. The more relevant characteristics they find, the louder they yell. A decision demon listens to the resulting 'pandemonium' and selects the pattern associated with the cognitive demon which is yelling loudest as the most likely explanation of the signal. The demons thus collaborate to solve the recognition problem; substitute writing on a blackboard for yelling and you have a blackboard model.

The term 'blackboard' was first introduced by Newell, who used this idea in the development of the production system (Newell and Simon, '72). The first of the modern generation of blackboard systems was used to control the Hearsay-II speech understanding system (Erman et al., '80).

This has been followed by a host of other systems, used for a wide range of applications including signal interpretation, 3-D molecular structure modelling and planning. There have also been some attempts to produce general-purpose blackboard systems which can be tailored to specific applications, for example the Hearsay-III system (Erman, London and Fickas, '88), and tools which knowledge engineers can use to help design systems, such as AGE (Nii and Aiello, '88).

4.2. BLACKBOARD ARCHITECTURE.

A blackboard system has two main components: the knowledge sources, independent sources of problem-specific knowledge to be used to solve the problem (the system equivalent of the human experts in the original model), and the blackboard data structure, which holds the problem data and through which the knowledge sources communicate and interact with one another (Engelmore, Morgan and Nii, '88). (See Figure 4.1). There must also be a control component, which is generally

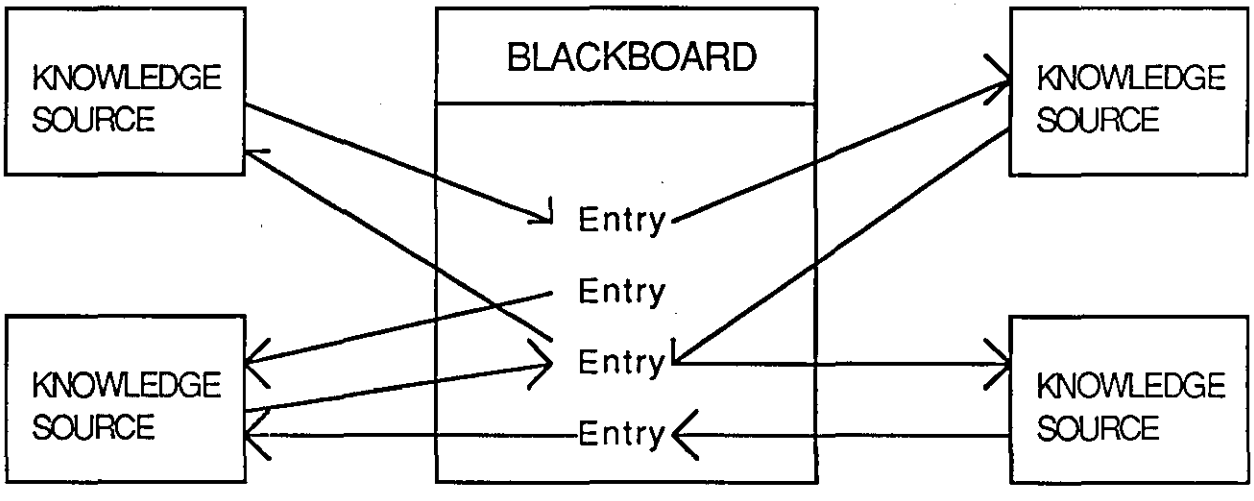


Figure 4.1. Structure of a Blackboard System.

application-specific. Control information may be held on the blackboard or in a separate module.

The blackboard data structure is generally hierarchical - knowledge is represented at several different levels, each of which has its own vocabulary. The blackboard may be partitioned into several different hierarchies. This is done for efficiency, so that knowledge sources do not consider data which cannot be relevant to them.

The knowledge sources are mainly responsible for taking information from one or more parts (sections or levels) of the blackboard and transforming it into information to be entered into the same or other parts of the board. There may also be special knowledge sources which perform tasks such as scheduling operations and checking to see if the termination conditions are met.

The normal cycle of activity is as follows: the system controller determines which of the knowledge sources are capable of utilising the information currently held on the blackboard and determines which of the possible operations should be executed first, then the selected operation is carried out. This produces changes in the information on the blackboard, so the controller is reactivated to assess the new situation and the cycle is repeated. The pre-conditions for each knowledge source, i.e. the information which they require, must obviously be specified so that the controller can determine which operations could be performed; some means must also be provided for assigning an order of priority to the operations, for example by assessing the usefulness and/or reliability of the potential output. (Jones and Millington, '86).

4.3. ADVANTAGES AND DISADVANTAGES.

The main advantage of blackboard systems is that they allow difficult problems to be solved in stages using a variety of methods, without the need to specify in advance the order in which the methods are to be applied. They have a modular structure, so knowledge sources can be developed, implemented and tested independently and new knowledge can be incorporated easily into an existing system when it becomes available.

The blackboard structure is particularly well suited to experimentation. Simply by adjusting the control module and varying the priorities attached to different knowledge sources, alternative configurations can be evaluated and compared for efficiency and accuracy in finding correct solutions to a problem.

Where the most efficient solution method for a particular problem has already been established, however, the use of a blackboard is usually less efficient than direct implementation of the algorithm. Scheduling the operations to be carried out can take up a good deal of processing time; if the appropriate order of operations for part or all of the solution process is already clear, this order should be specified and the operations carried out without reference to the blackboard control module.

If it is clear which knowledge source information is intended for, it is obviously faster and more efficient to pass the information to that knowledge source directly instead of via the blackboard. It is also undesirable to use the blackboard at intermediate stages within the operation of a single knowledge source; where knowledge sources are carrying out multi-stage operations, they need to be provided with facilities for storing intermediate results using their own internal data

structures so that data is not translated from one format to another and back again unnecessarily. (Erman et al., '88).

4.4. APPLICATION-SPECIFIC AND GENERAL-PURPOSE SYSTEMS.

The blackboard concept was originally developed in response to the need to find ways of tackling the problem of speech understanding by computer. It was soon realised that the ideas which had been used here could be applied to problems in other fields as well, and a number of other applications were developed which, though differing in detail, all shared the same general approach.

Some of the variations between early systems reflected the fact that different problems required different solution methods; others arose because lack of detailed information about existing systems meant that many teams of designers had no other option than to start from scratch in producing their own versions of standard blackboard components. To help avoid such duplication of effort, several attempts were made to produce multi-purpose systems or blackboard shells which could be tailored to suit different applications.

Attempts to use blackboard shells in more problem fields inevitably revealed some limitations; new ideas had to be introduced, which in turn could be abstracted and implemented in another generation of shells, leading on to yet more applications. The general trend might be expected to be for each new generation of systems and shells to be more elaborate and complex than the last, but this has not always been the case. As new ideas have been incorporated into systems, some existing ideas have been dropped, either because they have been superseded or because they have

been shown to be of limited applicability. Some designers have also deliberately sought to produce systems which are easy to understand and use, sacrificing flexibility where necessary for the sake of simplicity.

4.5. EXAMPLES OF APPLICATION-SPECIFIC SYSTEMS.

The systems which are described here have been selected to indicate the types of application for which blackboards have been used, and to give some impression of the wide variations in the way in which the basic concepts have been implemented.

4.5.1. Hearsay-II (Erman et al., '80).

The first application of the blackboard principle was to speech recognition; the Hearsay-II system was developed to recognise sentences constructed from a 100 word vocabulary, and was very successful, producing correct interpretations in approximately 90% of cases.

The system, written in SAIL, an Algol-60 dialect, uses six different levels of representation: segments of digitised speech, syllables, words, word-sequences, phrases and sentences. It includes thirteen knowledge sources, which create hypotheses at each level from information on the level below, control the number of hypotheses generated at each level, check for consistency and credibility of hypotheses, parse word sequences, predict words to follow or precede a phrase, check for termination and generate an interpretation of the final sentence to be passed to an information retrieval system.

Approximate knowledge is used in places for efficiency and

processing speed; for example, the full word-sequence parser is slow to run, so possible word sequences are checked first using an approximate parser which rapidly weeds out many unacceptable sequences, leaving only a small number to be checked by the full parser.

The control system allocates resources to the most promising actions by estimating the probable effects of an action, deducing its global significance and comparing it with other potential actions. Each knowledge source has a stimulus frame (a set of hypotheses which satisfy its pre-conditions) and a response frame (a stylised description of its actions) which are used together with global state information to calculate priorities.

Processing was at first opportunistic and data-directed, that is, promising hypotheses generated at one level were immediately followed up at other levels, but this was found to be inefficient at lower levels because of the inaccuracy of the credibility ratings. Completely processing one level before starting on the next proved more effective up to word-sequence level.

4.5.2. HASP (Nii et al., '88).

The HASP system was developed to interpret continuous sonar signals from hydrophone arrays used to monitor areas of ocean. The original intention was to produce a DENDRAL-like expert system, but this was shown to be impractical and a blackboard model was adopted instead. The main difference between the HASP problem and the Hearsay-II problem is that Hearsay was required to interpret independent utterances whereas HASP was interpreting a continuous signal, considering the current data in conjunction with its analyses of previous data.

HASP, like Hearsay-II, uses several different levels of representation - in this case, sonogram lines, harmonics, sources, vessels and at the top level a situation board which contains the current model of the ocean scene. The knowledge sources in HASP are also organised in a hierarchy, with sources called specialists, whose task is to put inferences on the current best hypotheses, on the bottom level, activators, which know when to use the various specialists, on the next level and a strategy knowledge source on top.

The specialist knowledge sources contain both 'textbook' knowledge and heuristics obtained from human experts. As the system maintains a top-level current best hypothesis from which new hypotheses are evolved, top-down, model-driven techniques are used as well as bottom-up, data-driven techniques. Extracting useful information from large amounts of data with a poor signal-to-noise ratio is computationally very expensive; using a top-down approach to 'tune' the search by generating signal expectations can reduce the problem to more manageable proportions.

The analysis carried out is time-dependent, and this aspect is handled by a mechanism called clock events. Specialist knowledge sources can put requests for recall at specific times, to review information and hypothesis elements, on a clock-event list.

The system produces explanations of its hypotheses in a similar way to expert systems, but as many of the inference steps in the development of a hypothesis are of limited interest, a special knowledge source is used to identify and present to the user the most important events.

4.5.3. UMass Schema System (Draper et al., '88).

Most vision systems are designed to perform specific tasks and are difficult to adapt to other uses because of the amount of domain-specific knowledge which they employ. This system arose from an attempt to produce a general-purpose vision system, by linking together many special-purpose ones. It was intended to be able to cope with such problems as the absence of experts able to introspect about their vision expertise, the difficulty of indexing into a potentially vast knowledge base of objects, the degree of uncertainty inherent in vision data and the vast quantity of data which is usually involved.

The system contains a number of single-object vision systems - schema - which co-operate and compete to arrive at a consistent interpretation of the image. It is modular: no schema depends on the internal details of any other schema, so new schemas can be added and existing ones modified easily, and a blackboard is used for all communication between schemas. However, the schemas are not all uniquely constructed; the system provides a set of knowledge sources and representations which are useful for any knowledge recognition, and the schemas contain information about which of this knowledge is relevant and when and how it should be applied. This information comprises an object-specific problem space definition, control knowledge for traversing the problem space and a function to translate evidence from knowledge sources into a degree of confidence in the presence of an object.

Instantiated copies of a schema, called schema instances, are invoked to identify instances of the schema's object class. These instances can be invoked by the user, but are more often invoked by each other, to gain support for hypotheses, to account for inexplicable data or to predict the

existence of objects which can be expected to occur with a currently-believed object. The system starts by invoking an instance of a general scene schema such as road-scene or house-scene, which then invokes instances of schemas for objects likely to be found within the scene, then these in turn invoke more schemas until the analysis is complete.

The Schema system has been designed to run in a parallel environment, which removes many of the scheduling problems which sequential blackboard systems suffer from: instead of having to decide which operation has the highest priority and so should be executed first, many operations are executed at once.

4.6. EXAMPLES OF GENERAL-PURPOSE SYSTEMS.

4.6.1. AGE (Nii and Aiello, '88).

AGE (Attempt to GEneralise) was conceived as a set of building-block programs covering common artificial intelligence techniques, together with an intelligent front end which would assist a user in constructing a blackboard system from them. Written in INTERLISP, the functions it provides include a user tutorial, debugging facilities, automatic generation of a system reference manual and a graphic interface.

The AGE blackboard contains hierarchically structured hypothesis elements, integrated by links representing support from above (expectation-links) or from below (reduction-links). The structure of the hierarchy may be simple or complex. Hypothesis elements can be generated by inference rules in the knowledge sources, and can also be generated and

named in advance by the user. They may include credibility ratings, calculated using preprogrammed procedures or user-provided algorithms.

The domain-specific knowledge needed by systems developed using AGE can be contained in one or more knowledge sources. These knowledge sources are used to create and modify hypothesis elements and relationships between elements. Each of them has associated with it lists of preconditions for its invocation, pairs of hypothesis levels that it spans and links it generates, a hit strategy to be used for the rules and a facility for binding variables. There may also be higher-level knowledge sources which manipulate the domain-specific knowledge sources.

Control components are needed to specify the input data format, initialisation function, processing method, rules for determining the next step in the processing and selecting the relevant knowledge source to carry it out, termination conditions and post-processing functions. Standard control components are provided, but user-written procedures may be substituted where required.

The AGE system has been widely distributed and used for applications ranging from a small system to solve cryptogram problems to TRICERO, a multisensor data-fusion system using separate blackboard subsystems (Williams, '88).

4.6.2. The Edinburgh Prolog Blackboard Shell (Jones et al., '88).

The Edinburgh system has been designed as an experimental tool, using the pattern-matching facilities built in to Prolog to simplify the implementation of the blackboard concept.

The blackboard structure is not defined explicitly; the user assigns indexes to entries, the choices of which determine the regions and levels into which the data will be organised. Blackboard data entries are Prolog unit clauses of the form:

bb(Tag,Status,Index,Fact,ConfidenceFactor)

where Tag is a system-supplied identifier, Status is a system-supplied indicator of whether the entry is current or defunct and ConfidenceFactor is a user-defined term or system default representing a degree of belief. Relations between entries are shown explicitly by Prolog clauses of the form:

supports(SupportingTag,SupportedTag).

The knowledge sources which contain the domain-specific knowledge required by the system must be supplied by the user as Prolog rules:

if Condition then Body to Effect est Est

where Condition is a test for the presence or absence of some combination of blackboard entries, Body and Effect specify the action of the rule and Est is a rating which is used by the scheduler.

The blackboard holds the agenda of tasks to be performed, in the form of knowledge source activation records (KSARs). Placing the agenda on the blackboard makes it possible for the scheduler to be implemented as a knowledge source which manipulates the agenda.

As a small-scale experimental tool, the Edinburgh shell has been tested on small applications such as the modelling of users of command-driven systems.

4.6.3. Hearsay-III (Erman et al., '88).

The ideas implemented in the speech-understanding system Hearsay-II have been abstracted, extended and generalised to produce Hearsay-III, a domain-independent framework for knowledge-based expert systems which is based on a relational database written in AP3 with control facilities in INTERLISP.

The designers aimed to provide the following facilities:

- support for the codification of diverse sources of knowledge
- support for the application and co-operation of these knowledge sources
- the ability to represent and manipulate competing solutions
- the ability to reason about partial solutions
- facilities for describing and applying consistency constraints
- support for long-term development of large systems, allowing experimentation with various knowledge sources and application schemes.

As Hearsay-III is intended to be used for large-scale applications, the scheduling problem is expected to be complex and is itself handled by a blackboard approach, using scheduling knowledge sources to handle such tasks as the assignment of priorities to knowledge source activation records. The blackboard is therefore split into two parts, which can be further subdivided by the user if required: the domain blackboard, used to hold a domain model and partial solutions, and the scheduling blackboard, used for performance reasoning.

The system was tested on small applications such as the solution of cryptarithmetic problems; full-scale applications include a system for constructing formal specifications of programs from informal specifications.

4.7. SUMMARY AND CONCLUSIONS.

Blackboard systems are clearly a very useful tool for coordinating the use of a variety of different sources of knowledge to derive a solution to a problem incrementally. They are particularly well suited to use in problem domains which lend themselves to hierarchical structuring, to the development of partial solutions to problems which cannot be solved completely because of inadequate or uncertain data, and to developing methods for tackling complex problems where a solution strategy cannot be specified completely in advance.

Pattern recognition, in which hierarchically layered representations, uncertainties in the data and multi-directional approaches to finding solutions are commonplace, is an ideal field in which to use a blackboard approach, so it is unsurprising that many of the systems which have been developed fall within this field.

Rule induction has an important role to play in pattern recognition, as was indicated in Chapter 3; some of the knowledge sources required for a pattern recognition blackboard system could be induced from sets of training examples. These knowledge sources could be developed independently and then incorporated into the blackboard system, or alternatively the induction process could form an integral part of the system: solutions verified by a high-level knowledge source, or by the system user, could be fed back to be used in modifying and updating the rules, producing a recognition system which learns from the experience gained by solving problems.

A rule induction or feedback module which developed and updated knowledge sources would not, strictly speaking, be a knowledge source

itself, but a meta-knowledge source, as the data it operated on would be not the problem data held on the blackboard but the knowledge sources themselves. One would effectively have a two-tier blackboard system, with the knowledge sources which made use of one blackboard being held themselves on another blackboard to be accessed by the meta-knowledge sources. Such a system would have to be written in a language such as Prolog which does not make a clear distinction between program and data. The Edinburgh Prolog blackboard shell (Jones et al., '88) would therefore appear to be a suitable starting point for the development of such a system.

CHAPTER 5

DEALING WITH UNCERTAINTY

5.1. SOURCES AND TYPES OF UNCERTAINTY.

The problems with which Artificial Intelligence is concerned are inherently uncertain - it is the lack of certainty, the need to make sense of incoherent or incomplete information, which gives rise to the need for 'intelligent' problem-solving behaviour. (Hinde, '85;'86).

Uncertainty can arise from a variety of sources; it can manifest itself in the problem data, in facts and in rules (Fox, '86). (Kodratoff et al., '88) describes these sources and types of uncertainty:

- unreliability of data due to symbolic noise (vagueness or ambiguity in the meaning of a term) or uncertainty in the measure of an attribute
- human-induced errors: assigning wrong values to attributes, misclassifying examples or giving too many or too few descriptors
- omission of necessary examples from a training set
- deficiencies in the description language used
- uncertainty in the problem domain
- noise in background knowledge.

(Schutzer, '87) gives some further examples: uncertainty about interactions between plan steps in a planning problem, uncertainty about the actions or intentions of an opponent in strategic planning, and problems with input data, including inaccuracy, incompleteness, disparity of sources, asynchronicity, inconsistency, variations in granularity and difficulties in data extraction, which mean that the data 'represents a kind of bounded ignorance'.

The uncertainty which is inherent in a system can be distinguished from the uncertainty introduced when modelling it using a particular representation system, which arises from vagueness in our perception and judgment of it. These distinct types of uncertainty may be best handled by different methods, numeric methods being more appropriate for the former, and symbolic methods for the latter. (Wise, '86).

5.2. APPROACHES TO HANDLING UNCERTAINTY.

The designers of A.I. systems can adopt two different approaches to the modelling of intelligent (human) behaviour: the understanding-oriented approach, aimed at duplicating the way in which humans operate, and the performance-oriented approach, aimed at producing the same results as a human would produce by whatever method seems most effective. (Spiegelhalter, '86). The various approaches which have been developed for dealing with uncertainty reflect this division as well as the differences between the types of uncertainty which arise in different problem domains.

Humans often use vague, ill-defined terms when describing their reasoning processes; the difficulties involved in translating vague expressions into numeric terms without introducing an unjustifiable level of precision can be circumvented by using a symbolic approach. The use of symbols allows one not only to reason under uncertainty, as with a numeric approach, but also to reason with or about uncertainty (Fox, '86; Hinde,'86).

Expert Systems often employ IF...THEN rules obtained from human experts, with associated certainty factors which may show various forms

of bias: people's estimates of probabilities tend to be influenced by such factors as the ease with which they can recall or imagine an event (which leads to a bias towards specifics rather than generalities) and the degree of 'representativeness' which an event appears to display (which means, for example, that if a coin is to be tossed six times, 'HHTHTH' will be judged a more probable outcome than 'HHHHHH'). (Wise, '86). If the biases can be recognised, it should be possible to remove or reduce their effects; the results obtained will then be more accurate, but less 'human'. The main advantage of using such rule-based systems is the ease with which their conclusions can be explained to the user.

Performance-oriented approaches are frequently based on probability theory or an extension, simplification or adaptation of it. Probability theory is the oldest and most widely used method of handling uncertainty, and is derived from a formal description of rational behaviour. Probabilities are a function of two things: the proposition under consideration, and the evidence at hand. Their precise magnitude is usually less important than the reasoning behind it, the context in which it applies and the sources of information which would cause it to change. Probability theory is unique in its ability to process context-sensitive beliefs, and it has been shown that for any reasonable scoring rule, any scalar measure of uncertainty is either worse than or equivalent to it. (Pearl, '88; Wise, '86). However, its use does present some problems: there may be insufficient data available to allow a full probability distribution to be specified accurately, with traditional probability theory ignorance cannot be distinguished from uncertainty, the computational cost may be excessive, and if approximations and simplifications have to be made the results obtained may not be accurate.

The need to express ignorance, as opposed to uncertainty, has led to the development of methods based on intervals: the range of probabilities which could be assigned to a hypothesis is given, with the lower limit of the interval based on the weight of the evidence supporting the hypothesis, and the upper limit calculated from the weight of evidence against it, or the support for its negation. The width of the interval represents the degree of ignorance, or lack of evidence.

There is a clear difference between the concept of probability and the concept of truth. A probability of 0.5 attached to a hypothesis does not mean that it is half-true; hypotheses are either true or false, and probabilities can be regarded merely an estimate of the relative likelihoods of these two alternatives. The idea of reasoning with truth rather than with probability - or with belief, as the truth or falsehood of hypotheses will, in general, not be known - has led to the development of truth maintenance systems, which are used to establish sets of mutually consistent hypotheses. Truth maintenance can be linked with probabilistic methods: the use of a preference ordering of assumptions will ensure that the 'most probable' solutions to a problem are explored first. (Hinde et al., '89).

(Pearl, '88) gives the following classification of methods for handling uncertainty:

logicist - uses nonnumerical techniques, primarily non-monotonic logic.

neo-calculist - uses mathematical representations, with new calculi to circumvent the perceived deficiencies in traditional probability calculus (Dempster-Shafer calculus, fuzzy logic etc.)

neo-probabilist - remains within the framework of traditional

probability theory

heuristic - uncertainties are not made explicit, but are embedded in domain-specific procedures and data structures.

Here, methods are reviewed under four different headings: numeric methods, which covers traditional probability theory and related methods such as Dempster-Shafer theory and the use of certainty factors, methods derived from fuzzy set theory, truth maintenance, and other methods (mainly symbolic).

5.3. NUMERIC METHODS.

5.3.1. Probability Theory and Bayesian Inference.

Mathematical probability theory was developed to help explain random physical phenomena, where the frequencies of occurrence of each of a set of mutually exclusive states which an object can take on tend to approach stable values as the number of independent trials increases (Schutzer, '87). The classical definition of probability is:

If a single trial of a chance situation can have one of N exhaustive, mutually exclusive and equally likely outcomes and if f of those N possibilities are favourable to an event A, then the probability of A, p(A) is equal to f/N.

The conditional probability of an event A given an event B, p(A|B), is defined by:

$$p(A|B) = \frac{p(A \text{ and } B)}{p(B)} \quad (\text{if } p(B) > 0)$$

If A and B are independent events, then p(A|B) = p(A) and

$$p(A \text{ and } B) = p(A).p(B).$$

(Kotz and Stroup, '83).

Bayesians see the conditional relationship p(A|B) as more basic than that of joint events, so belief in joint events is computed from conditional relationships:

$$p(A \text{ and } B) = p(A|B).p(B).$$
$$= p(B|A).p(A).$$

This leads to the inversion formula:

$$p(H|e) = \frac{p(e|H).p(H)}{p(e)}$$

where p(H) is the prior probability of H and p(H|e) is the posterior probability of H given evidence e. The inversion formula can be used to update beliefs in response to evidence. The formula can also be given in an odds-likelihood form:

$$O(H|e) = L(e|H).O(H)$$

where O(H|e) is the posterior odds p(H|e)/p(notH|e), L(e|H) is the likelihood ratio p(e|H)/p(e|notH) and O(H) is the prior odds on H, p(H)/p(notH). (Pearl, '88).

Two events A and B are said to be conditionally independent given C if:

$$p(A \text{ and } B|C) = p(A|C).p(B|C)$$

If A and B are conditionally independent given C or notC, then the formula:

$$\frac{p(C|A \text{ and } B)}{p(\text{notC}|A \text{ and } B)} = \frac{p(A|C)}{p(A|\text{notC})} \cdot \frac{p(B|C)}{p(B|\text{notC})} \cdot \frac{p(C)}{p(\text{notC})}$$

can be derived from Bayes' rule. The assumption of conditional independence allows a large set of events to be split into a network of local event groups (LEGs) to simplify calculations. (Wise, '86).

It has been shown (Johnson, '86) that the conditional independence assumptions which would permit updating of the probabilities of hypotheses on the basis of multiple items of new evidence cannot hold; however, methods based on these assumptions, though not theoretically justifiable, may produce useful results (see Section 5.3.4).

A probabilistic model is normally specified by a joint distribution function, which assigns a non-negative weight to every elementary event (every conjunction in which each atomic proposition occurs once) such that the weights add up to 1. The distribution function may be specified by an algebraic expression (in the case of continuous random variables), or indirect methods such as network representations may be used.

The assumptions on which the theory is based should not raise any insuperable difficulties: a 'catch-all' can be included if necessary to ensure exhaustiveness, the hypothesis space can be refined beyond binary propositions to form multi-valued variables which each reflect a set of mutually exclusive hypotheses, and intermediate variables can be introduced to induce conditional independence - for example, medical symptoms which are linked together can be identified as a 'syndrome'. (Pearl, '88; Shepherd & Hinde, '89).

The specification of an accurate, coherent set of prior probabilities is often a major problem. If only some of the required parameters can be obtained, the maximum entropy principle - which states that as much uncertainty should be retained (as few hidden assumptions should be made) as possible - can be used to estimate the rest. If the specified parameters are non-coherent, the model may need to be adjusted.

Probabilities can be regarded as unknown parameters which have distributions; if a probability is considered as a proportion in a (possibly imaginary) sample, then a 95% confidence interval can be established around it using binomial sampling theory (Spiegelhalter, '86). Vectors of 0's and 1's can be used to represent probabilities, each bit position representing a possible state of the world, so each vector is a Monte Carlo sampling of possible states (Wise, '86).

5.3.2. Dempster-Shafer Theory.

The Dempster-Shafer theory of evidence was designed to handle cases where the probability distribution is incompletely known; it has the ability (which traditional probability theory lacks) to distinguish between uncertainty and ignorance. It has an underlying logical semantics so it can be implemented in a propositional logic system in a straightforward manner (Provan, '90), and it has the ability to model the narrowing of the hypothesis set with accumulation of evidence (Gordon & Shortliffe, '85).

The frame of discernment, T, is an exhaustive set of mutually exclusive hypotheses. Evidence disconfirming an element of T can be interpreted as support for the remaining elements, but there is nothing to indicate how the support should be divided between them. Instead of enforcing an arbitrary division, D-S theory allots belief not just to single elements, but to all subsets of T. The impact of each piece of evidence is represented by a basic probability assignment (bpa) which assigns a number in [0,1] to every subset of T such that the numbers sum to 1. As the elements of T are exhaustive the empty set, Ø, must be assigned a belief of 0. The belief assigned to a subset A is denoted m(A). Any uncommitted belief is assigned to T.

Belief in a proper subset of the frame of discernment entails belief in supersets which contain it, so another function must be used to specify the total belief in a subset: given a bpa, m, the corresponding belief function Bel assigns to every subset A of T the sum of beliefs committed to every subset of A by m. For a single-element subset A, Bel(A) = m(A); as the beliefs assigned by m must sum to 1, Bel(T) = 1.

Bel(A) represents the necessary support for a subset A; (1-Bel(A')) represents the possible support for A, or the plausibility of A. The belief interval for A is [Bel(A),(1-Bel(A'))]. The width of this interval can be regarded as the amount of uncertainty with respect to a hypothesis, given the evidence.

To determine the combined effect of two pieces of evidence, their belief functions must be combined. Given two belief functions Bel_1 and Bel_2 with corresponding bpa's m_1 and m_2, the combined belief function $Bel_1 + Bel_2$ can be determined by first calculating the combined bpa, m_1+m_2. Dempster's rule defines $m_1+m_2(A)$ to be the sum of all products of the form $m_1(X).m_2(Y)$, where A is the intersection of X and Y. This rule may assign a non-zero belief to the empty set, so the bpa has to be normalised: if the value initially assigned to Ø is k, then $m_1+m_2(Ø)$ is set to 0 and the values assigned to all non-empty subsets of T are divided by (1-k).

One problem with Dempster-Shafer theory (Gordon & Shortliffe, '85) is that as a set with n elements has 2^n subsets, if the frame of

discernment is large the number of computations required will be vast. Barnett proposed a method for reducing the computations from exponential time to polynomial time, based on the assumptions that evidence will apply to single elements of the frame of discernment or their negations and that evidences can be reordered. His method involves combining all the evidence applying to any one singleton or negation, then combining pairwise the resulting bpa's for each singleton and its negation, then finally combining the resulting n bpa's.

(Gordon & Shortliffe, '85) suggest a computationally tractable approach for applying the theory in a hierarchical hypothesis space, which involves pruning the network of subsets of T to a tree in which each node below T has a unique parent by removing subsets of no semantic interest. Generally the negations of hypotheses in the tree will not themselves be in the tree, so disconfirming evidence must be associated directly with the disconfirmed hypothesis; an approximation is used to combine disconfirming evidence, displacing belief upwards towards T to avoid consideration of subsets not in the tree.

(Provan, '90) describes how VICTORS (Provan, '87), an ATMS-based high level vision system (described in section 5.5.5), was extended using Dempster-Shafer theory to test how assigning weights, calculated by determining how far constraints are satisfied, to assumptions would affect its performance. Heuristic approximation algorithms were used to simplify the computations.

The theory shows a discontinuous sensitivity to small probabilities; ignoring 'negligible events' can have radical effects, so caution must be used when making simplifying assumptions. (Wise, '86).

5.3.3. MYCIN's Certainty Factors.

The Certainty Factors method developed for use in MYCIN (Buchanan & Shortliffe, '84), an Expert System for medical diagnosis, and since used in several other systems, is an approximation to probability theory which simplifies the computations required. With this method, the degree of belief in a proposition is represented by two numbers, the measure of belief (MB) and the measure of disbelief (MD), both of which vary between 0 and 1; the certainty factor, cf is equal to MB-MD and varies between -1 and +1. (Wise, '86).

The certainty factor attached to an if *evidence* then *hypothesis* rule determines to what extent the evidence should change the degree of belief in the hypothesis. Certainty factors can be defined in terms of prior and posterior probabilities:

$$cf(H|e) = \begin{cases} \dfrac{p(H|e) - p(H)}{1 - p(H)} & \text{if } p(H|e) > p(H) \\ \dfrac{p(H|e) - p(H)}{p(H)} & \text{if } p(H) > p(H|e) \end{cases}$$

where p(H) is the prior probability of H and p(H|e) is the posterior probability of H given e.

Where two items of evidence bear on the same hypothesis, the certainty factor z resulting from the combination of certainty factors x and y is defined by:

$$z = \begin{cases} x + y - x.y & (x,y \geq 0) \\ \dfrac{x + y}{1 - \min(|x|,|y|)} & (x,y \text{ of opposite sign}) \end{cases}$$

$x + y + xy \qquad (x\,y < 0)$

This combination function is not derived from the definition given above and is not consistent with it, but was proposed as a commutative, associative approximation (Heckerman, '86).

BACH (Sobolevitch, '85) is a production-based expert system which uses certainty factors, together with a truth maintenance system (see section 5.5) to produce a consistent view of activity in an area under surveillance from military intelligence reports. It uses a frame-like system to represent units; the frames have slots which include the unit internal ID, the certainty factor, and an assumption list and justification (used by the TMS). Reports are assumed to be independent of each other but units are not, so if a new unit's antecedents include a new report, its certainty factor is calculated using a MYCIN-type formula, but if a rule concerns units only the maximum of their certainty factors is assigned to its conclusion.

5.3.4. Prospector (Gaschnig, '82).

Three different types of relation specifying how a change in the probability of one assertion affects the probability of other assertions are used in Prospector, an Expert System intended to help geologists in exploring for hard-rock mineral deposits.

The Prospector knowledge base contains models of certain classes of ore deposits, each encoded as an independent, hierarchically structured inference network. The terminal nodes in the networks represent field evidence; other nodes represent hypotheses. The system operates by matching field data supplied by the user against the models; it requests

additional information when necessary, informing the user of the geographical rationale for its questions, and provides a summary of its findings.

Logical relations are employed where the truth value of a hypothesis is completely determined by the truth values of the assertions that define it. Conjunction (AND), disjunction (OR) and negation (NOT) operations are used. Where a hypothesis is defined by the conjunction of several pieces of evidence, the probability assigned to the hypothesis is the minimum of the evidence probabilities; for disjunction, the maximum is taken. These computations correspond to the standard fuzzy set union and intersection formulas (see Section 5.4).

With second type of relations, plausible relations, the assertions are related to the hypotheses by rules with associated rule strengths which measure the degree to which a change in the probability of the assertion changes the probability of the hypothesis. The odds-likelihood form of Bayes' rule is used to compute the hypothesis probability. (The theoretical background of this rule is somewhat shaky - see Section 5.3.1. - but it still produces satisfactory results). Certainties are expressed on a -5 to +5 scale, with linear interpolation between these extremes; the system translates these certainty values into probabilities (odds) to perform its calculations, then translates these back into certainty values when communicating with the user.

When assertions must be considered in a particular sequence the third type of relations, contextual relations, are used. Contexts specify conditions that must be met before an assertion can be used in the reasoning process.

5.4. FUZZY SETS.

The observation that attempts to model inexact concepts by formal systems of increasing precision lead to decreasing validity and relevance led Zadeh to propose the use of fuzzy set theory, a generalisation of traditional set theory which has found applications in numerous fields, including:

- pattern recognition
- clustering
- political geography
- decision-making
- robot planning
- chromosome classification
- medical diagnosis
- engineering design
- systems modelling
- process control
- social interaction systems
- structural semantics

(Gaines, '76).

Fuzzy sets are based on the idea of continuously graded degrees of membership of sets; the characteristic function of an ordinary set

$\mu_A(x)$: U -> {0.1} where $\mu(x) = 0$ x in A

$\mu(x) = 1$ x not in A

is replaced, for a fuzzy set, with a characteristic function of the form

$\mu_A(x)$: U -> [0,1]

which specifies the 'degree of membership of x in A'. With this definition 'crisp' concepts can still be represented adequately, but there is no necessity to assign artificial boundaries to concepts which are inherently vague.

The standard set operations - union, intersection, complementation - can be defined for fuzzy sets in several different ways. The definitions which are most commonly used are:

A union B = {max(a(x),b(x))/x|x is an element of U}

A intersection B = {min(a(x),b(x))/x|x is an element of U}

A' = {(1-a(x))/x|x is an element of U}

It can be shown that these definitions of union and intersection are the only ones which are consistent with the requirements that the operations should reduce to the normal set operations for degrees of membership of 0 and 1, that they should be order-preserving and continuous, and that the normal associativity, commutativity, distributivity and idempotence rules should be obeyed. If the distributivity and idempotence requirements are dropped, which may be considered desirable for reflecting natural language usage, then Zadeh's alternative definitions can be used:

A union B = {(a(x) + b(x) - a(x).b(x))/x|x is an element of U}

A intersection B = {(a(x).b(x))/x|x is an element of U}

(Gaines, '76). As well as the standard set operations, there is a range of operations which are specific to fuzzy sets, for example concentration, which reduces the degree of membership of elements which are 'only partly' in the set, normalisation, which adjusts the degrees of membership so that at least one element is 'totally' in the set, intensification and fuzzification.

Imprecise statements can be modelled as fuzzy sets using linguistic variables - variables whose values are natural language expressions referring to some quantity of interest. These expressions can be represented by fuzzy sets composed of the possible values that the quantity of interest can assume. For example, if the quantity of interest could assume an integer value between 1 and 10, the expression 'few' could be represented by

{0.4/1,0.8/2,1/3,0.4/4}

The natural language expressions normally form a structured finite set, with syntactic rules for generating expressions and semantic rules for associating fuzzy sets with them. Primary terms are modelled by fuzzy sets, and hedges (very, quite etc.) are modelled by fuzzy set operations. (Schmucker, '84).

Fuzzy set theory can be used to extend classical logic to produce a fuzzy logic in which the constraint that every statement must be either absolutely true or absolutely false no longer applies. The compositional rule of inference, which states that if R is a fuzzy relation from U to V and X is a fuzzy subset of U, the fuzzy subset of V which is induced by X is given by the composition of R and X, can be used when variables range over finite sets. It includes as a special case a generalisation of modus ponens:

If X is B then Y is C

X is A

Y is D

where X and Y are variables in universes U and V respectively, A and B are fuzzy subsets of U, and C and D are fuzzy subsets of V.

An alternative approach to approximate reasoning is truth value restriction: the degree to which the actual value of a variable agrees with its antecedent value in a production can be represented as a fuzzy subset of a truth space and used in a fuzzy deduction process to determine the corresponding restriction on the truth value of the right hand side of the production (Nafarieh, '88).

The concept of fuzziness can be extended to mathematical structures, replacing the concept of the value of a variable with 'the degree of membership of a value', as a result of which values seem to play the role of functions and non-fuzzy functions become functionals (Gaines, '76), and to the domains of interest of sets: operations which map a fuzzy set and domain of interest into a new fuzzy set and new domain of interest can be used, for example, to fuzzify algorithms for manipulating black-and-white images for application to grey-scale images (Edmonds, '81).

One problem with fuzzy set theory is that there is no proof that it models perception or judgment, and no clearly defined way of determining if a given membership function is 'right' (Wise, '86). The theory assumes that grades of membership of property categories may be expressed by functions, the values of which submit to the conventional arithmetic operations, and if unary operations such as the transformation of fuzzy sets with hedges are to be meaningful, a ratio scale must be used for subjective measurements. Franksen's investigation of the empirical justification of the assumptions made shows that for fuzzy sets representing a large variety of psychophysical continua and corporate utility under risk, a power function is an appropriate form of membership grading (Franksen, '78); little other research has been conducted in this area.

Other problems with Zadeh's fuzzy logic include extreme vagueness of results in fuzzy conditional propositions, and weaknesses in the ways in which chain reasoning, conjunctive fuzzy conditional propositions and combination of evidence are dealt with (Nafarieh, '88).

5.4.1. FRIL (Baldwin et al., '88).

FRIL (Fuzzy Relational Inference Language) is a logic programming language which extends Prolog to represent doubt and uncertainty associated with both facts and rules, using support pairs which define intervals containing point value probabilities. The use of support pairs is a compromise between using single probability values and using fuzzy sets - the intention is to avoid introducing an unjustifiable degree of precision, while keeping the computational burden to a minimum. The underlying Support Logic programming theory is more strongly related to a generalisation of probabilistic reasoning than to fuzzy reasoning, but FRIL includes a mechanism for representing and reasoning with fuzzy sets.

The first number in each support pair, the lower limit, represents necessary support; the second represents possible support. A total lack of evidence will be represented by the support pair [0,1]. With FRIL, a lack of evidence supporting a proposition is not interpreted as support for its negation; the following relationships apply:

Necessary support for P + Necessary support against P ≤ 1

Necessary support for P = 1 - Possible support against P

Necessary support against P = 1 - Possible support for P.

The probability of a proposition is evaluated by:

1. Determining a proof path for the proposition, ignoring any assigned probabilities.
2. Evaluating the probability associated with the proposition using this proof path.
3. Repeating steps 1 and 2 for all other possible proof paths.
4. Combining the results into a final probability interval.

When evaluating the probability associated with a proof path, FRIL assumes independence unless told otherwise. Conditional probabilities can be entered in the knowledge base if they are known. When combining the support from different proof paths, the default method assumes that the unknown probability lies within the intervals defined by each of the support pairs being combined, and so uses an intersection rule, which can yield a fairly narrow support pair interval even when the intervals contributing to it are quite wide. This method assumes that there is a degree of dependence between the inferences from different support paths. An alternative method, the Dempster rule, can be specified where the sources of inference are independent and conflicts may occur.

The standard Prolog unification algorithm is extended in FRIL to include a form of semantic unification. Semantic terms such as 'tall', 'average_height', 'short' can be defined using fuzzy sets such that there is a partial match between them; the unification process allows, for example, support for a person being tall to be deduced from the fact that they are known to be of average height.

FRIL has been used for numerous AI applications, including probabilistic reasoning in scene analysis, radioactive waste safety assessment, experience bases using conceptual graphs, software

dependability modelling and Expert Systems for the effect of stress on operator performance, aircraft design and chemical plant control.

5.4.2. A Fuzzy Rule-Based Production System (Nafarieh, '88).

The fuzzy logic incorporated in this system, which was developed and tested on target detection and recognition from temporal sequences of forward-looking infra-red (FLIR) and TV images, was intended not only to cope with the complexity of the problem and the uncertainty in the data, but also to facilitate the provision of a natural language interface to mid- and high-level subsystems, to increase the believability of results by relaxing perceived precision and, using contextual knowledge, to enable conflicting interpretations to be resolved and the initial analysis of the system to be refined.

The system has three phases: prescreening, scene recognition and contextual knowledge-based validation. The first phase uses rules such as:

If: range is long

Then: prescreened window size is small

where the terms 'long', 'small' are defined by fuzzy sets. The mapping of measurements to linguistic terms employs the definition of the 'Hamming distance' between two fuzzy sets:

$$d(A,B) = \sum_i |\mu_A(x_i) - \mu_B(x_i)| \qquad \text{(for finite sets)}$$

The scene recognition stage segments, recognises and labels scene components, distinguishing between various man-made objects (armoured personnel carriers, tanks) and natural objects (the sky, fields, trees). The

final stage resolves any conflicts in the results. A fuzzy k-nearest neighbour algorithm is used to produce class memberships for test vectors, based on class memberships of the training data and distances of the test data from the training data; these memberships are mapped to linguistic confidence values.

The system was compared with a rule-based system which used Dempster-Shafer theory, selectively extracting groups of four features at a time, generating confidences and combining these with previously generated confidences. The fuzzy system produced better results.

5.5. TRUTH MAINTENANCE.

5.5.1. The Origins of Truth Maintenance.

Truth maintenance systems (TMSs) were developed to support the use of non-monotonic reasoning in problem solving. This type of reasoning may be appropriate when knowledge of a problem is incomplete and default assumptions must be made to enable a solution to be found, when the universe of discourse is changing or when temporary assumptions are used to test a possible solution (Frost, '86). The truth maintenance concept is based on the use of belief values which, unlike truth values, are subject to alteration and revision in the light of new evidence; TMSs are designed to be used by deductive systems to maintain logical relations among beliefs, to modify the belief structure when premises are changed and to use the logical relations to trace the source of contradictions or failures, leading to more efficient backtracking (McAllester, '78).

The development of TMSs stemmed from Stallman and Sussman's work, (Stallman & Sussman,'77), which aimed at improving the behaviour of chronological backtracking in combinatorial search problems such as electronic circuit analysis by recording dependencies as the search progressed - dependency directed backtracking (DDB). (Shanahan & Southwick, '89).

There are two main types of TMS. The earlier type, justification-based systems (JTMSs) such as those produced by Doyle (Doyle, '79b) and McAllester (McAllester, '78), store as fundamental data the immediate justifications for inferences, maintaining a single consistent hypothesis and using DDB to restore consistency by rejecting an assumption when contradictions are discovered. These systems have several limitations:

- only one solution can be considered at a time, alternative solutions cannot be compared
- the current choice set can only be changed by introducing a contradiction which cannot be removed later, so switching states is difficult
- their machinery is cumbersome
- if some but not all of the inferences based on an assumption set have been derived when a contradiction is found, the work may have to be repeated later if the complete set of inferences is required.

(de Kleer, '84).

The later assumption-based systems (ATMSs), which were developed by de Kleer in an attempt to solve these problems, record the fundamental assumptions on which inferences rest, maintaining multiple self-consistent but mutually inconsistent sets of hypotheses or contexts.

(Shanahan & Southwick, '89). However, they too have limitations:

- if only one solution is required they are hopelessly inefficient
- they may search regions of the solution space which DDB would avoid
- debugging is difficult; intermediate states represent pieces of many solutions, and it can be hard to tell which is causing problems.

(de Kleer & Williams, '86).

The development of a combined system which was intended to have the advantages of both types and the disadvantages of neither, using DDB to provide the search strategy with a coarse focus and to handle control assumptions, and an ATMS to provide an additional level of discrimination and to handle non-control assumptions, is described in (de Kleer & Williams, '86). ATMSs have also been implemented with some form of rating system to ensure that the most promising solutions are investigated first (Hinde et al., '89; Provan, '90).

5.5.2. Justification-Based Truth Maintenance Systems.

The JTMS developed by Doyle is generally considered to be the first true TMS. It operates by keeping track of which statements, assumptions and hypotheses are currently believed (IN) and which are not currently believed (OUT). (Doyle, '79b).

Doyle's JTMS employs two data structures: nodes, which represent beliefs, and justifications, which represent reasons for beliefs. Each node has one or more justifications associated with it. A node is IN if and only if at least one of its justifications is valid. There are two different types of justification, support-list justifications and conditional-proof

justifications. Support list justifications have two parts: an in-list containing nodes used in the derivation of the belief, all of which must be IN for the justification to be valid, and an out-list, all the nodes in which must be OUT for validity. The out-list is used to allow assumptions to be retracted; if the out-list of an assumption A contains the node notA, the assumption will be retracted automatically if it leads to a contradiction. (Norman, '87). Conditional-proof justifications are used when the status of the node depends on the validity of a hypothetical argument; they have three parts, a consequent, an in-list and an out-list, and are valid if the consequent is IN whenever each node in the in-list is IN and each node in the out-list is OUT.

The JTMS maintains a single consistent context (the current set of IN nodes) by using DDB to restore consistency when a contradiction arises. The nodes which contribute to the contradiction are found by tracing through the dependency structure, one of them is chosen as the culprit and rejected, and all justifications which depend on this node are checked for validity. (Shanahan and Southwick, '89).

A simplified JTMS was developed by McAllester. His system allows propositions to have one of three truth values, true, false or unknown, and represents all logical relations between propositions as disjunctive clauses; this representation makes no distinction between antecedents and consequents, which simplifies the backtracking process. (McAllester, '78).

5.5.3. Assumption-Based Truth Maintenance Systems.

The ATMS described in (de Kleer, '86a) was designed to allow a problem database to contain unresolved inconsistencies, so that the

problem solver could follow more than one search path through the solution space at once and compare alternative solutions with one another. It was also intended to increase the ease with which results obtained in one region of the space could be carried over into other regions, by recording derivations in the most general way possible.

ATMS nodes have a label, supplied by the ATMS, which determines the environments or contexts in which the datum holds by specifying the minimal sets of assumptions from which it can be derived. A premise has an empty label; the label of an assumption specifies a single assumption set which contains only the assumption itself. Nodes also have justifications supplied by the problem solver giving the parent nodes from which they were derived.

A special node is used to represent falsity. The assumption sets specified for this node are 'nogood' sets - sets from which inconsistencies have been derived. These sets are used to partition the space into self-consistent environments, and thus to ensure that inconsistencies are not propagated: when computing a node label, the system checks the assumption sets and removes any which contain 'nogood' sets.

5.5.4. REVgraph's Consistency Maintenance (Bowen & Mayhew, '88).

The REVgraph is a 3-D model used in the construction of a geometrically consistent description of a scene from a description of the edge segments in a pair of stereo images. The edges are typically fragmented; the reasoning system uses both bottom-up and top-down processing to complete broken edges, find vertices and identify and describe regions, using rules such as "if 3 lines can be extended to meet at

a point, hypothesise a vertex at that point" and an object-oriented algorithm which focuses on wires (edges) which have been identified as 'interesting'.

The initial data is ambiguous rather than wrong, so the system used to control the uncertain reasoning is called a Consistency Maintenance System (CMS) rather than a TMS. The CMS is context-based: a context is a subset of the database within which there are no contradictions and no paths of justification are incomplete, and is represented by a list of integers each representing a point at which a contradictory context was split. The set of contexts in which a justification is valid is the intersection of the sets of contexts in which its premises are valid; the set of contexts in which a fact is valid is the union of the sets of contexts in which its justifications are valid.

The CMS data structure is a directed graph consisting of 'fact' nodes (representing items in the database) and 'data dependency' (DD) nodes (representing justifications). The information associated with a fact node is:

support-by list	list of all DD-nodes (justifications)
support-for list	list of all DD-nodes for which this fact is a premise
value	pointer to the database
context list	list of contexts of which this fact is a member
contradictions	list of contradictions of which this fact is a part

The information associated with a DD-node is:

premises list	fact nodes required to make this deduction
consequence	pointer to the result of the deduction (a fact node)
rule identifier	pointer to the rulebase

context list	list of contexts in which this justification is valid
disallowed list	list of contexts ruled out by contradiction

There are two types of justification: rigorous justifications (for derivations from logical implications) and heuristic justifications (for derivations from heuristics).

The CMS explores alternative solutions in parallel and allows partial solutions to be examined. Its main disadvantage is that it is very slow if the number of contexts is large. When an n-element nogood set is found, each context containing this set can be split into 2^n new contexts containing subsets of it. Attempts have been made to reduce the number of contexts created after the discovery of an inconsistency by considering only maximal proper subsets of the nogood set and using background knowledge to identify trustworthy elements of it, but keeping the number of contexts within reasonable bounds clearly presents problems.

5.5.5. VICTORS (Provan, '87).

VICTORS is a vision system which aims to identify two-dimensional puppet figures, consisting of a number of parts (head, body, arm etc.) each with rotatory, translational and scaling degrees of freedom with respect to the parts to which they are joined, from input data consisting of sets of four points which represent the vertices of overlapping rectangles.

The data is preprocessed to determine the areas and orientations of rectangles and the overlaps between rectangles, then passed through a set of filters which place restrictions on acceptable assignments of parts to rectangles. Parts which are tightly constrained (for example, the trunk, which must be overlapped by five other parts) are identified as 'seeds' and

used as the starting points for determining sets of locally consistent constraints, then an ATMS is used to establish global consistency, producing a set of dependency graphs - nodes interconnected by constraints - from which puppet figures can be identified.

The use of an ATMS enables the system to provide trace explanations for part assignments, allows holonomic constraints to be defined with variable geometry, permits the exploration of multiple solutions simultaneously, enables the database to be updated with the input of new information, produces robust behaviour with noisy data and occluded or incomplete figures and demonstrates the necessity of using domain-dependent constraints to reduce the search space. There is, however, a price to be paid for these advantages: the ATMS is the most computationally expensive part of the VICTORS system.

5.5.6. LUMP (Hinde et al., '89).

A blackboard system combined with an ATMS and a 'soft focussing' rating system forms the basis of LUMP (Loughborough University Manufacturing Package), a process planning system which serves as an integrating framework for a number of subsystems. Its input, constructive solid geometry strings, comes from a designer system and its output is numerical machine codes or programs which are passed on to factories. There are four major subsystems:

1. A Prolog procedure to translate the input data into manufacturing features
2. A Prolog planner to generate the generic operations used to manufacture a component
3. A proprietary relational DBMS containing information about the

properties and relationships of the machines and tools in a factory
4. A numerical code generation package.

The subsystems interact through a blackboard whose entries have assumption bases attached to them. Modal tags indicate whether entries follow necessarily from their antecedents, when their assumption bases are formed from the union of their antecedents' assumption bases, or represent one of a range of possibilities, when they have a new assumption attached to them to prevent any inconsistency which arises from being propagated back to their antecedents.

The ATMS allows the system to reason with several possible solutions at once, but the rating system should ensure that the 'most obvious' solution is considered first. Ratings can never increase, and will generally decrease, as the formation of a solution progresses; the system will always work on the solution whose current rating is highest, switching to another solution when the rating drops. This should ensure efficiency without sacrificing exhaustiveness.

5.6. OTHER METHODS.

5.6.1. Cohen's Endorsements (Wise, '86).

The essential idea behind the Endorsements method proposed by Paul Cohen is that numeric certainty labels are of limited value without some knowledge of the evidence on which they are based. His method associates with each hypothesis a body of endorsements, which represent reasons for believing or disbelieving the hypothesis. These allow the certainty to be

assigned to the hypothesis to be judged in the light of what the result is to be used for (Spiegelhalter, '86).

The endorsements can be interpreted in two ways: as a method of keeping track of what has been conditioned on, or as a way of coding correlations into networks. (Wise, '86).

Cohen's example application is GRANT, a system for finding agencies to fund a research proposal. Classes of research interests are arranged in a hierarchy; it is assumed that agencies will specify the largest class which contains only their interest, and that sibling classes (distinct sub-classes of a single parent class) will not overlap. Suppose that a class A has sub-classes B and C (Figure 5.1). The link between A and B

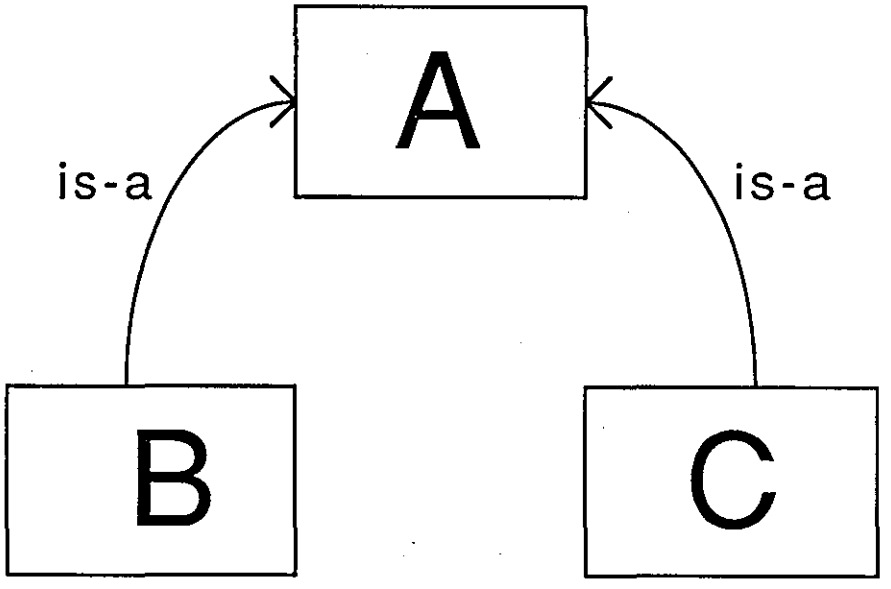


Figure 5.1. A simple hierarchy with 'is-a' links.

represents a positive endorsement as an agency which is interested in A will also be interested in its sub-class; the "is-a, is-a-inverse" link

between B and C represents a negative endorsement as the assumptions imply that an agency which has expressed an interest in B will not be interested in C.

The method does not include any general scheme for comparing two bodies of endorsements; they can just be pairwise ranked (Neapolitan, '90).

5.6.2. Logical formulation of linguistic ideas (Fox, '86).

Fox places the emphasis on describing rather than measuring uncertainty, using qualitative rather than quantitative knowledge. His method uses logical formulations of terms which people use to describe uncertainty, for example:

possible: S is possible if no conditions necessary for S are violated

plausible: S is plausible if S is possible and the arguments for S are stronger than the arguments against S

probable: S is probable if S is possible and there is at least one item of evidence in favour of S

certain: S is certain if a sufficient condition for S is true.

The use of this kind of qualitative representation may reduce the need for precise quantitative assumptions to be made, thus producing a robust system. This method also facilitates the exploitation of analogy, generalisation and causal reasoning in hypothesis formation, allowing patterns in the data to be utilised and enabling the decision to be made to be structured into separate components - features which should compensate for any loss of precision in comparison with numeric methods.

The terms used are applied uniformly to facts, data and rules.

Nominal labels are attached to statements by counting their 'pros' and 'cons' and making comparisons with logically exclusive alternatives. This is a statistically weak method, but is suitable for use in problem domains where a high degree of precision in the measurement of uncertainty is not required. The application described in (Fox, '87), PSYCO (Production SYstem COmpiler), is a small system for medical diagnosis; in this field a strict probabilistic ranking of possible diagnoses is inappropriate as, for example, a doctor would attach more significance to a 10% probability of cancer than to a 75% probability of a common cold.

The method can be combined with numerical methods to allow precise calculations to be carried out where necessary, for example when a close decision between two alternatives has to be made.

5.6.3. Bundy's Incidence Calculus (Corlett & Todd, '86).

As the use of purely probabilistic methods requires a complete set of correlations which are frequently not available, Bundy proposed mixing probabilities with first-order logic in his Incidence Calculus. A probability distribution is posited over a collection of possible incidents, each of which gives a complete specification of the truth values of atomic sentences. Each sentence can then be assigned a probability equal to the measure of the set of possible incidents in which it is true. (Bacchus, '88).

If the set if incidents in which an event E is true is represented by i(E) and the (finite) set of all possible incidents is represented by w, then

if i(E) = w then E is true

if i(E) = {} then E is false

$$p(E) = \frac{|i(E)|}{|w|}$$ (assuming each incident is equally likely)

This approach has several problems associated with it: statistical generalisations cannot be represented (Bacchus, '88), the inference mechanism is unsatisfactory, a computationally expensive inconsistency detection mechanism must be employed when assigning incidences to mutually exclusive events, and the probabilities assigned to the conjunctions of independent events will vary with the way in which incidents are assigned to the events, the distributions of the probabilities being both complex and difficult to analyse (Corlett & Todd, '86).

A variant of the Incidence Calculus which avoids these problems and is amenable to statistical analysis, the Monte Carlo method, is described in (Corlett & Todd, '86). This method has the unusual (and perhaps undesirable) characteristic of sometimes drawing different conclusions from the same data on different occasions.

5.6.4. Lp logic (Bacchus, '88).

The logical formulation Lp proposed by Bacchus extends Bundy's idea of mixing probability with first-order logic to allow the use of random variables, by specifying a probability distribution over the domain of discourse rather than over the sentences of the language. It permits closed formulas such as:

Bark(Fido)

which must have a probability of either 0 (false) or 1 (true), and also

permits open formulas such as:

$[Bark(x)|Dog(x)]_x > 0.5$

where x can be interpreted as being a random variable bounded by the probability term formed by the square brackets, to represent statistical generalisations (in this case, 'more than 50% of all dogs can bark).

Lp can represent empirical probabilities which take the form of statistical statements, but cannot represent a subjective probability assignment to a closed formula; however, Bacchus combines Lp with an inductive mechanism for assigning degrees of belief to sentences based on the empirical generalisations expressed in the logic, using an inductive assumption of randomness. For example, if it is known that 90% of dogs bark and Fido is a random dog, then the degree of belief assigned to bark(Fido) is 0.9. Such degrees of belief cannot be expressed directly in Lp; only the statistical information from which they can be generated can be expressed. The calculated degree of belief will depend on what knowledge is used in the inductive step of randomisation; the maximum possible amount of knowledge is used, following the partial ordering of knowledge

a > b if a -> b is deducible from the knowledge base.

Lp and the belief mechanism together offer plausible inductive inference and sound deductive inference, but the knowledge base must be organised efficiently to ensure that the relevant facts are deduced quickly and a time limit must be applied to deduction as Lp is undecidable.

5.7. CONCLUSIONS.

The methods for handling uncertainty which have been reviewed here cover a very wide range. They aim to handle many different types of uncertainty and ignorance, and each has its own strengths and weaknesses. There is no one method which can be considered best for all AI systems; it may be necessary for a system to use a different method for each of the different types of uncertainty which arise within it.

When selecting methods, it is important to note the assumptions - both explicit and implicit - which are made, and to consider their appropriateness to the problem domain. It is also important to consider the degree of accuracy which is required in the results; there is no point in carrying out lengthy and elaborate calculations to determine exact probabilities if only approximate figures are required.

For a pattern recognition system where the aim is to establish the most probable consistent interpretation of the objects in an image, an appropriate choice would seem to be to use an approximate numeric method to establish a preference ordering for possible object identifications. Truth maintenance could be used to resolve inconsistencies between identifications, if the advantages it offered were sufficient to justify the computational overheads.

CHAPTER 6

RECOGNISING AN OBJECT USING FEATURE DATA

6.1. INTRODUCTION: DESCRIPTION OF THE PROBLEM.

The overall aim of this research was the production of a system which could learn to recognise complex artifacts such as cars in visual images from the output of a feature matcher which finds the best matches for a set of simple feature patterns in an image, giving mean x and y co-ordinates and a rating for each match. It was intended that the final system should be able to identify and locate all the instances of a number of different objects in an image, including partial and occluded views of objects, giving the probability that an instance of an object exists at a given location where insufficient information is available for a positive identification to be made. This system would consist of a number of single object recognisers each comprising a set of probabilistic rules for recognising members of a particular category of objects, together with such other knowledge sources as might be required, working in collaboration through a blackboard to arrive at the most probable identification of all the objects in a test image.

Rather than developing each single object recogniser independently, it was decided that a program should be written which would induce a set of rules for recognising members of a category of objects from a set of training examples.

To ensure accuracy in the descriptive element of the induced rules, the training examples would have to be carefully selected to ensure that they fully define the object category. The drawback with this is that

probabilistic rules developed using a preselected set of images could not be expected to reflect accurately the probabilities which would apply when using the recogniser with images which had not been subjected to the same selection process. It was therefore decided that some provision should be made for updating the probabilistic element of the rules, using feedback information on the results of the recognition process.

The initial requirement was therefore to produce a three-stage system which would:

1. Induce a set of probabilistic rules defining a category of objects from feature data obtained from a set of training examples.
2. Utilise the rules developed in Stage 1 to recognise instances of the object category in test images.
3. Update the probabilistic element of the rules, using feedback data obtained from Stage 2.

This chapter describes the development of the rule induction/recognition/feedback system; the following chapter gives the results of tests conducted on it using both synthetic data and real images showing side views of cars.

6.2. APPROACH.

The approach to be adopted is largely determined by the characteristics of the feature matcher. This takes a set of fairly small, simple feature patterns and finds the features in an image which could match each pattern, giving the pattern number, mean x and y co-ordinates and a rating for each match. The matching is shape, size and orientation

dependent.

If the object to be recognised is large and complex, it will not be possible to use a feature pattern which will match the whole object image; rather, patterns which match small parts of the object will have to be used. A set of suitable object parts and a corresponding set of patterns must therefore be selected. The selection of these parts and patterns may present considerable difficulties, particularly if members of the object category differ very noticeably from one another in appearance.

An insistence on a one-to-one correspondence between object parts and feature patterns would make the selection of appropriate part and pattern sets almost impossible for many categories of object, so the system must allow for a many-to-many correspondence: a single object part may be matched by more than one feature pattern, and a single feature pattern may match more than one object part. This means that the part set may contain several parts of the same size and shape, e.g. the wheels of a car, and that variations in the size and shape of a part can be allowed for by using several different patterns to match it.

The object part set and feature pattern set selected must obviously be large enough to enable the object to be recognised to be distinguished from all other objects which may be expected to occur in the images, but as the running time of the system and the size and complexity of the induced rule set will depend on the number of parts and patterns used, the numbers of both must be kept to a minimum. The system should, ideally, allow the user to experiment with different compositions of sets so that minimal sets can be identified easily.

With objects such as cars where the size and shape of a part can differ considerably from one model to another, the use of a strictly limited pattern set will not allow a highly-rated match to be obtained for every image feature corresponding to an object part, and the match ratings cannot be expected to be a reliable indicator of the probability that an object part has been detected. it therefore seemed appropriate to simply apply a threshold to the ratings: to establish, by examination of the feature data obtained from training examples, the minimum rating which is to be accepted for each pattern, to discard all matchings with ratings below this threshold and to treat all matches with ratings above the threshold as being of equal value. The levels at which the thresholds are set must be low enough for the vast majority of features corresponding to object parts to be preserved. With low thresholds, however, the data can be expected to contain a considerable amount of noise, i.e. features which do not correspond to object parts, and the design of the system must allow for this.

The rules to be induced by the first stage of the system are to be used to perform two distinct functions: to identify sets of features which could correspond to object instances, and to attach probabilities to these sets. It was decided that the task of designing the system should be made easier by using a separate set of rules for each function. This would also simplify the implementation of the feedback stage, as only the probabilistic rules would require updating, the identification rules being fixed by the induction process.

The identification of possible object instances in recognition tests involves finding subsets of the set of features found by the feature matcher whose elements could represent distinct parts of an object, and

checking the relationships between the elements of each such subset to determine whether the relationships which hold between the corresponding object parts are satisfied. The identification rules must, therefore, specify both feature pattern/object part correspondences (to enable the object parts which individual features could represent to be identified) and the relationships which hold between object parts.

If the feature patterns have been specifically chosen to match particular object parts, the part/pattern correspondences could be specified by the user. However, in the final multi-object recognition system a standard set of features may be used to identify parts of all the objects to be recognised, so the correspondences will not necessarily be known to the user; also, the determination of a minimal acceptable pattern set may be easier if the system has the ability to establish the correspondences for itself. They can be established quite easily by comparing the feature data and object data from training examples. Patterns which perform poorly can be eliminated by determining the proportion of features of each pattern whose co-ordinates match those of a particular object part, and recording a correspondence only if the proportion exceeds some preset limit.

The relationships between object parts must be determined from the object-part data which is supplied to the system; to allow comparisons with the feature data to be made, this should include the mean x and y co-ordinates of each part. As the object may be located anywhere within the image frame, what is significant is not the absolute co-ordinates of a part, but its co-ordinates relative to some other part or some fixed point on the object. With a system which is intended to recognise partial and occluded views of objects, any of the parts may be missing from the part

set, so it is not practicable to select one particular part as the origin for an object-based co-ordinate system; the simplest approach is to base the relationship rules on the co-ordinate differences between pairs of parts. It was decided that the rules should specify just the maximum and minimum x and y co-ordinate differences between each ordered pair of parts.

One possible drawback with this approach is that using the horizontal and vertical distances between parts rather than the length and direction of a straight line joining them will make the system sensitive to the orientation of the object, but as the feature matcher is itself orientation-sensitive, this cannot be considered to be a major problem. If an object may occur in several different orientations, recognition can best be accomplished by treating each orientation as a different object-view, to be handled by its own rule set. This has the advantage of providing the user of the (multi-object) system with additional information: the recogniser will specify not only the location of the object, but also its orientation.

Another potential disadvantage is that the rule set will not explicitly state any higher-level part relationships such as the relationships between the distances between different pairs of parts - the fact that the wheel arches on a car must be the same distance apart as the wheels, for example, will not be apparent. However, such relationships may be implicit in the set of part-pair distance limits: the limits placed on the distances between the wheels and the corresponding wheel arches will imply a limit on the difference between the wheel distance and the wheel arch distance.

The identification of maximum and minimum distances, with the assumption that any distance lying between these extremes will be accepted as valid, does not allow for the possibility that the permissible distances between a pair of parts may not be capable of being represented by a single closed interval. Where it appears that a disjunction of several intervals would be a more appropriate representation, the same approach as was suggested for the recognition of different orientations of an object could be adopted: the object category could be split into sub-categories. For example, if the ranges of wheel distances for standard saloon cars and for limousines could be seen to be disjoint, and it seemed desirable to eliminate wheel pairs whose distances fell between these ranges (perhaps because such wheel pairs might belong to vans), then 'saloon' and 'limo' could be recognised independently. However, this should rarely be necessary; even where distances do fall into several disjoint ranges, intermediate distances will not normally be obtained from other objects which could be expected to occur, so amalgamating the ranges will not give rise to recognition errors.

The distances between features can be expected to influence the probability that a feature set corresponds to an object instance, but the relationships between distances and probabilities cannot be specified easily. With naturally occurring objects, the distribution of values of a particular dimension will often follow a predictable pattern - a normal distribution, for example - which can be defined by a simple mathematical formula, but with man-made artifacts the same does not apply; the distribution of car wheel distances, for example, will not appear as a smooth curve, but will have peaks whose heights are determined by the relative popularity of different models. Specifying the distribution function for each relevant dimension, and using these for the

determination of probabilities, would involve the development of a vast and very unwieldy set of probabilistic rules. It therefore seemed necessary to apply a thresholding technique yet again, limiting the use of feature co-ordinates to the identification of candidate feature sets and basing the assignment of probabilities to these sets on other criteria.

The only other information available for use in the derivation of probabilistic rules was the set of object parts to which the elements of a feature set might correspond, and the set of patterns used to identify the features, so it was decided that a table of rules should be drawn up specifying empirical probabilities based on these factors. It was felt that a single feature should not be considered to provide adequate evidence of the existence of an object instance, so the rule table should contain an entry for every set of two or more object parts and corresponding pattern set. If there are n object parts, and part i can be matched by p_i different patterns, the total number of entries in such a table can be shown to be

$$\prod_{i=1}^{n} (p_i + 1) \quad - \quad \sum_{i=1}^{n} p_i \quad - \quad 1$$

For an object set of four parts, two matched by a single pattern and the other two by two different patterns, there will be (2x2x3x3)-(1+1+2+2)-1 = 29 table entries; for a set of five parts, two matched by one pattern, two by two patterns and one by three patterns, there will be (2x2x3x3x4)-(1+1+2+2+3)-1 = 134 entries. Despite the simplifications adopted, then, the task of inducing an adequate rule set for an object with even a very limited number of parts will clearly be quite large.

The probabilities can be derived by establishing, for each part set/pattern set combination, the number of feature sets in the training images which satisfy the identification rules and so could be recognised as object instances, and the number of these which correspond to actual object instances. Dividing the latter number by the former number will yield the required probability.

If the results obtained from the recognition stage are to be used to update the probabilities, it will be necessary to store the two numbers used in the probability calculation so that these can be incremented and the probability recalculated when required.

6.3. PRELIMINARIES: DATA PREPARATION.

Before deriving a rule set for a new object view, a set of training images showing the full range of instances of the object view to be recognised must be obtained. These training examples will be used to determine the maximum and minimum distances between each pair of object parts; the distance limits cannot be adjusted by feedback, so it is important that the examples show all possible extremes of object size and orientation. Then the object parts to be feature-matched must be selected, named and numbered. The parts used must be ones which occur in reasonably consistent positions in all instances of the object view, and have shapes which can be matched by a small number of patterns. There must be sufficient parts to definitely identify the object and to distinguish it from any other objects which could be expected to occur in the images, but no more than are required for effective recognition as the addition of a single element to the part set can be expected to

approximately double the running time of the system. The list of parts must be entered using the format:

part_list(Objectview,[1,PartName1],[2,PartName2],. . . . .]).

The next stage is the selection of the feature patterns to be used. The feature matcher is sensitive to size and orientation as well as shape, so it may be necessary to use more than one pattern to match a single object part - for example, a car wheel may be matched by several circles of differing diameters - but the number of patterns used must be kept down as far as possible as the addition of an extra pattern will have a similar effect to the addition of an extra part on the system's running speed. A single pattern may be used to match more than one object part, where appropriate - the front and back wheels of a vehicle, for example, may be matched by the same pattern(s). The patterns should be numbered, then the images should be fed through the feature-match program (or the matches identified manually) and the features found should be entered in the form:

feature(PicNo,PatternNo,X,Y,Rating).

The object-view instances in each picture must then be identified and numbered (a training picture may include several object instances), and the co-ordinates of each object part must be determined. The object part data should be compared with the feature data to ensure that a sufficient proportion of object parts have been identified by the matching process; if the hit-rate is too low, alterations must be made to the part set and/or pattern set. The object part data should be entered using the format:

object_part(Objectview,PicNo,InstanceNo,PartNo,X,Y).

The comparison of the object part data and feature data will also allow appropriate rating thresholds to be determined for each feature

pattern. The thresholds should be set low enough for the ratings of the vast majority of features representing object parts to exceed them; as the system has quite a high noise tolerance, it is better to include spurious features than to exclude features representing object parts. The thresholds should be entered in the form:

rating_threshold(PatternNo,Threshold).

6.4. DESCRIPTION OF THE SINGLE-OBJECT SYSTEM.

6.4.1. Editing the feature data.

The top-level **edit** predicate is used to prepare feature data for use by the rule induction and recognition programs. It can be used to edit all the available feature data, by entering the goal

edit.

or to edit the data from a single specified picture:

edit(PicNo).

The editing process starts by removing all features whose ratings are below the threshold value, then checks the remaining features for duplication: if an image feature has been matched by more than one pattern (which may occur if, for example, the image feature is a circle the diameter of which falls between the diameters of two pattern circles) then only the highest-rated feature match is preserved. Two matches will be considered to be duplicates of one another if their x and y co-ordinates differ by less than 3. The remaining features are numbered, and recorded using the format:

edfeature(PicNo,FeatureNo,PatternNo,X,Y).

The final function of **edit** is to set the list of identified features in the picture to the empty list, []. This list is used during the recognition process.

6.4.2. The Rule Induction Stage.

The rule induction process uses all the available edited feature data, so it is necessary to ensure that edited data from pictures not included in the training set is not available to it. This can be achieved by delaying loading feature data from test pictures until the induction process is complete, or by specifying the pictures in the training set at the edit stage.

The first part of the process involves marrying together the object-part data and the feature data from the training examples. This is accomplished by the predicate **feature_match**, which checks each feature in turn to see if a corresponding object part can be found (for correspondence, the X and Y co-ordinates of the feature and the object part must both differ by less than 3) and makes the appropriate entry in the matched_feature table:

matched_feature(Object,PicNo,FeatNo,PatNo,Inst,PartNo).

where Inst and PartNo identify the corresponding object part if one has been found, and are set to 0 otherwise.

One of the ways in which the matched_feature table is used is in the determination of pattern/object part correspondences by the **pattern_match** predicate: if a sufficient proportion, say 5%, of the features of a given pattern number have been matched to object parts of a given part number, then an entry is made in the match table:

match(Object,PartNo,PatternNo).

The maximum and minimum X and Y co-ordinate differences between each pair of object parts are then calculated from the object part data by the predicate **set_limits**, and recorded in the distance_limits table in the form:

distance_limits(Object,Part1,Part2,MinX,MaxX,MinY,MaxY).

No allowance is currently made in these limits for possible slight discrepancies between object part co-ordinates and the corresponding feature co-ordinates; introducing some tolerance in the limits would, however, be a simple matter if it were to be considered necessary.

The main part of the rule induction phase is the calculation of the probability that a given set of features will correspond to an instance of the object, given that the feature patterns match the elements of some subset of the object parts and that the feature co-ordinates satisfy the relevant distance limits. This is effected by the predicate **make_sets**, which calls the subsidiary predicate **find_sets** to form every such set of two or more features in the training examples, using the information in the matched_feature, match and distance_limits tables, and counts both the total number of sets (Sets) and the number whose elements all match parts of a single known object instance (Matches) for every distinct part set/pattern set pair, recording the results in the set_probability table in the form:

set_probability(Object,PartSet,PatternSet,Sets,Matches).

The relevant probability is not stored explicitly, but can be calculated readily from this table when required; the percentage probability is (100*Matches)/Sets. The information is stored in this form for ease of updating by the feedback process.

The **find_sets** predicate makes full use of the automatic backtracking facility built in to Prolog. It operates by first invoking the subsidiary predicate **find_pairs** to form every possible ordered set of two different object parts and all the two-element feature sets corresponding to each of them, then invoking the **find_multiples** predicate to 'grow' the feature sets by adding features which could correspond to higher-numbered parts. Its insistence that the part sets should be arranged in part number order ensures that sets will not be duplicated.

The **find_pairs** predicate checks the co-ordinate differences between pairs of features against relevant entries in the distance_limits table, then stores feature pairs together with part pairs whose limits they satisfy in a within_limits table, using entries of the form:

within_limits(Object,PicNo,PartPair,FeaturePair).

Find_multiples uses this table rather than the distance_limits table when growing feature sets, to avoid repeating the calculations required to check limits.

The three sets of induced rules - the match, distance_limits and set_probability tables - are listed for inspection by the user.

6.4.3. The Recognition Stage.

For recognition, the image has first to be fed through the feature matcher and the resulting feature data edited as for the training images. The test picture number and the object view to be sought are specified using the goal:

search(Object,PicNo).

The threshold probability to be accepted for recognition can also be specified if required:

search(Object,PicNo,Threshold).

The default threshold is 100%, but a much lower threshold will be required if the feedback facility is to be invoked after the recognition stage, to ensure that no object instances in the picture escape recognition.

Feature sets corresponding to possible object instances are identified by the **search_sets** predicate, which calls the **find_set** predicate used in the rule induction stage, then calls **add_to_list** to assign each feature set a number, calculate the probability that it represents an object instance using the information in the set_probability table, and insert the set number and probability in a list arranged in decreasing order of probability.

When this list is complete, the **report** predicate is invoked to notify the user of the results of the search. The first element of the list is checked to see if its probability exceeds the threshold value. If so, the feature set is checked to see if it contains any previously identified features and, if it does not, presented to the user for acceptance or rejection. When a set is accepted, the features in it are added to the list of identified features for the picture (which was initialised as part of the editing process), a new object instance is created and the relevant object parts are recorded for use by the feedback process. The first element of the list is then discarded, and the process is repeated until the probability threshold is passed or the list is empty.

6.4.4. The Feedback Stage.

The feedback stage uses the object part data recorded at the conclusion of the recognition stage and the relevant predicates from the rule induction stage to update the set_probability table.

Note that the function of the feedback stage is just to update probabilities; the distance limits cannot be altered, or the set of probabilities calculated using the original limits would be invalidated. If an object instance which occurs in a test picture is not recognised because it falls outside the range determined by the training examples, then if the recognition program is required to recognise such instances the rule induction stage must be repeated, with the test picture included in the set of training examples.

If the recognition process has identified only some of the features corresponding to parts of an object instance, because some of the feature distances fall within the limits and others fall outside them, the feedback process should not be invoked as it would produce variable results: some probabilities would be made more accurate, others less so. Only complete, accurate recognition results should be used for feedback.

The modified set_probability table is listed at the conclusion of the feedback stage, but the match and distance_limits tables are not listed as they are not altered by feedback.

6.5. RUNNING THE SYSTEM.

'RECOGNISE1' was written in Cprolog on the VAX 11/750. It is run by typing:

cprolog

to enter the Prolog interpreter, then when the prompt appears entering the names of the master file specifying the program files to be used:

[master].

and the data files for the training examples, e.g.:

[quadtest].

The feature data on the training examples is edited by entering the goal:

edit.

When the system responds with

yes

the rule induction stage can be invoked by entering

learn(Object).

The rule sets are printed out at the conclusion of the learning stage, which should only take a minute or two if the number of training examples is not too great. When the prompt reappears a test picture file can be loaded, then edited using:

edit(PicNo).

The object which has been learnt can then be sought by entering the goal:

search(Object,PicNo).

or **search(Object,PicNo,ProbThreshold).**

When a feature set/part set is presented for acceptance or rejection,

the user must enter

a.

to accept it, or

r.

to reject it. At the conclusion of the recognition process another test picture file can be loaded, or the feedback facility can be invoked by entering the goal:

feedback(Object,PicNo).

Appendix B contains edited listings of the test runs (which are described in Chapter 7).

CHAPTER 7

TESTING THE SINGLE-OBJECT SYSTEM

7.1. INTRODUCTION.

The original intention to test the system on data produced by R.S.R.E.'s feature matcher had to be abandoned as problems with the feature matcher meant that appropriate data could not be supplied on time. The initial testing was, therefore, carried out using synthetic data; for the main tests, R.S.R.E. supplied processed images from which feature data was extracted manually, using a process as close to that used by the feature matcher as possible.

The use of synthetic data had the advantage of allowing some of the system's capabilities - particularly, its ability to cope with noise - to be demonstrated more systematically than would have been possible if only 'real' data had been used.

The test on processed images of cars was expected to provide a good illustration of the problems which could be expected to arise in selecting appropriate sets of object parts and patterns for matching, and in using a training set which may not fully define the object to be recognised.

7.2. TEST USING SYNTHETIC DATA.

The synthetic data was specially devised for this test, and was intended to test the system's capability to deal with noise, its ability to distinguish different object parts represented by identical shapes and its

capacity to handle pictures containing more than one object instance. The object used was a quadrilateral, quad, with two circles, a square and a triangle at its vertices.

7.2.1. Data Preparation.

The selection of object parts and feature patterns presented no problems; the object parts to be matched were the vertex shapes:

1 left_circle

2 square

3 right_circle

4 triangle

and the feature patterns used were:

1 circle

2 square

3 triangle

The distance limits between quad vertices were predetermined and the training examples, pictures 1, 2, 3 and 4, were carefully devised to fully define these limits. Pictures 1, 2 and 3 each contained one instance of a quad; picture 4 contained two quads. Spurious features were added to bring the total number of features in each picture to twelve: four circles, four squares and four triangles. These 'noise' features were not placed randomly; their positions were selected to ensure that several partial feature sets could be found among them, but the only complete sets of four features which satisfied the distance limits were the actual quad instances.

The test, picture 5, also contained twelve features, of which just

one set of four features represented a complete quad, but several pairs of features could represent part of a quad.

Figures 7.1 to 7.5 show all the pictures used, with the features displayed on a 120x120 grid. The quad vertices are shaded for ease of identification.

7.2.2. Test Results.

A listing of the system run is given in Appendix B. The rule induction process correctly identified all the object part/feature pattern matches and the vertex distance limits. As all the object instances in the training examples were complete, i.e. they all contained all four object parts, the Matches figures in the set-probability table entries could be expected to equal the total number of training instances; the row of fives in the table thus provides confirmation that the table is correct. The Sets figures in the table could be expected to decrease as the size of the pattern/part sets increased, reaching the number of Matches for a complete part set; this, again, can be seen to apply.

A search of picture 5 with a probability threshold of 10%, low enough to pick up all the feature sets with a positive probability of representing an object instance, produced four feature sets: one complete set of four features, with a 100% probability, and three smaller sets with lower probabilities. A visual inspection will confirm that these represent all the possible quads in the picture. (The sets are shaded differently in Figure 7.5; note that the circle at (60,110) is contained in two different sets, and so is shaded with both horizontal and vertical lines.)

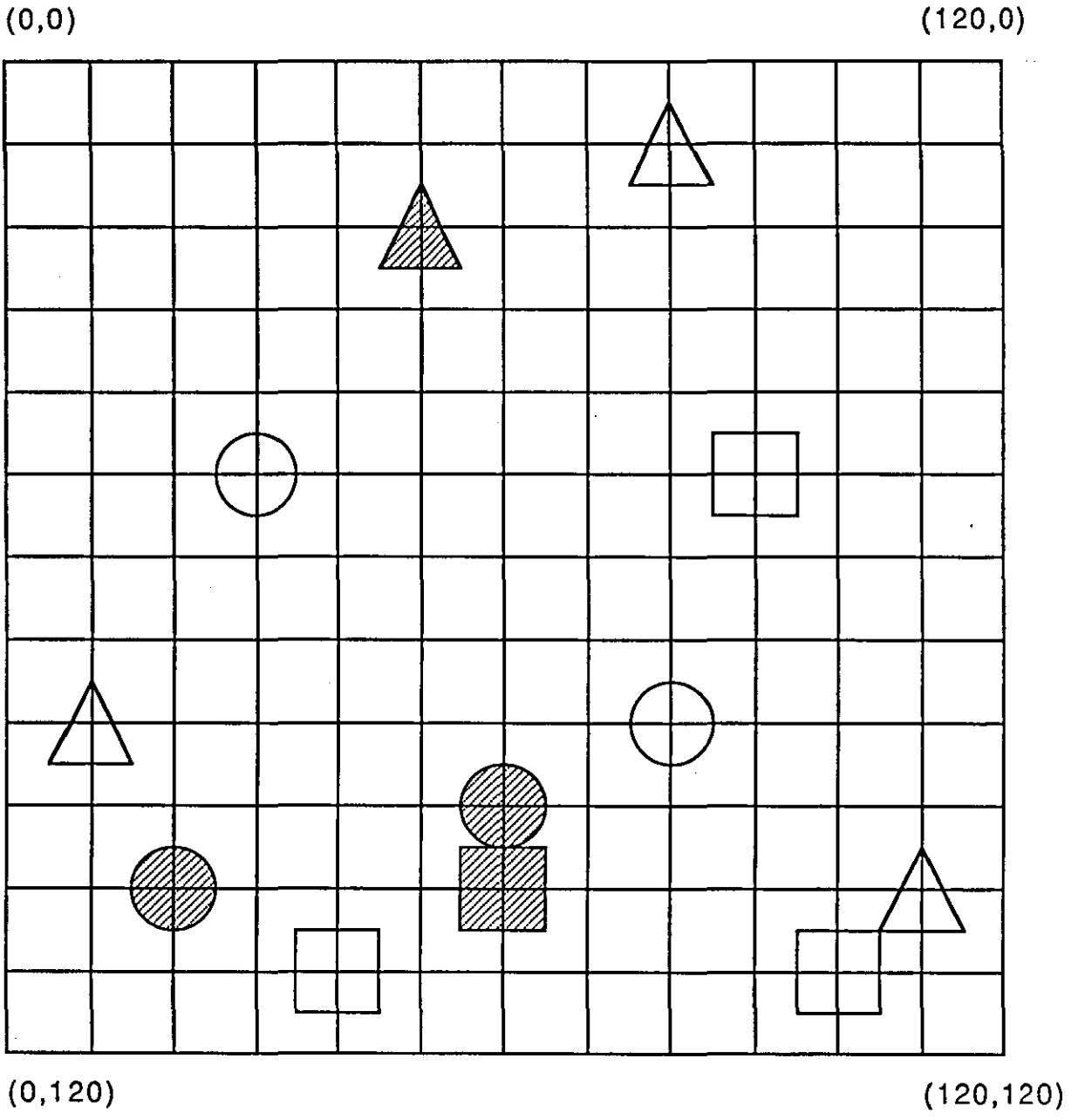


Figure 7.1. Quad Picture 1.

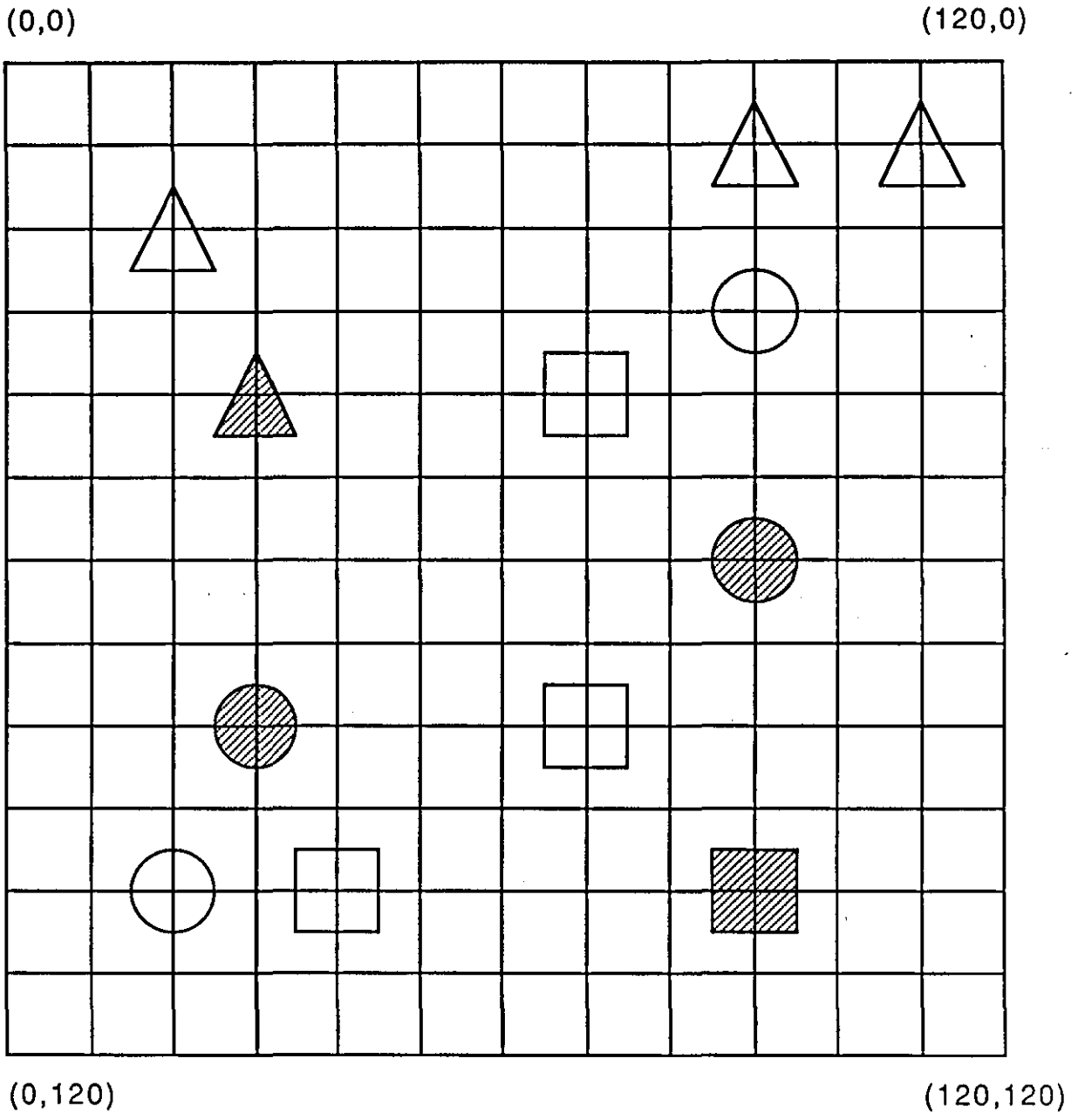


Figure 7.2. Quad Picture 2.

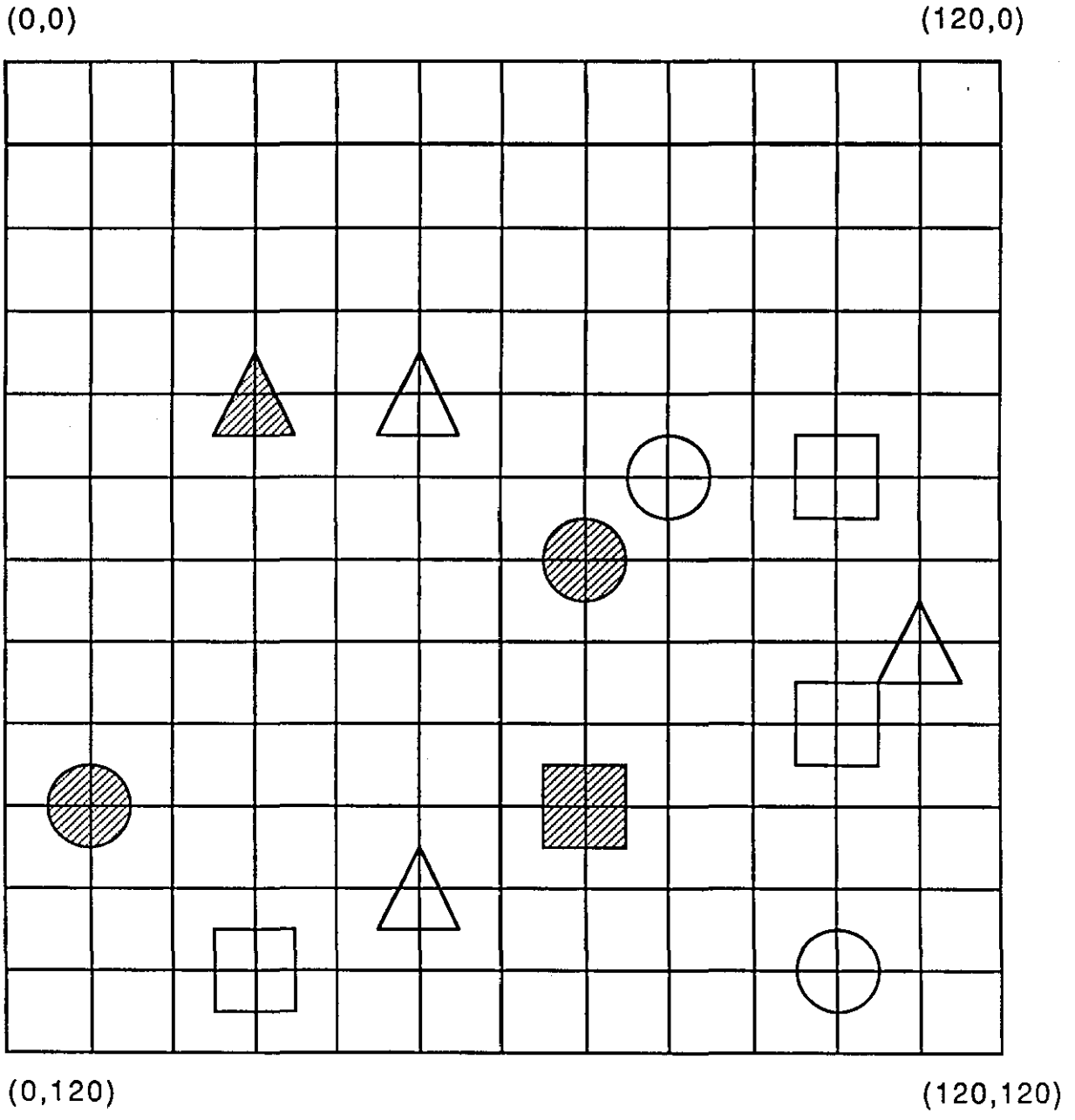


Figure 7.3. Quad Picture 3.

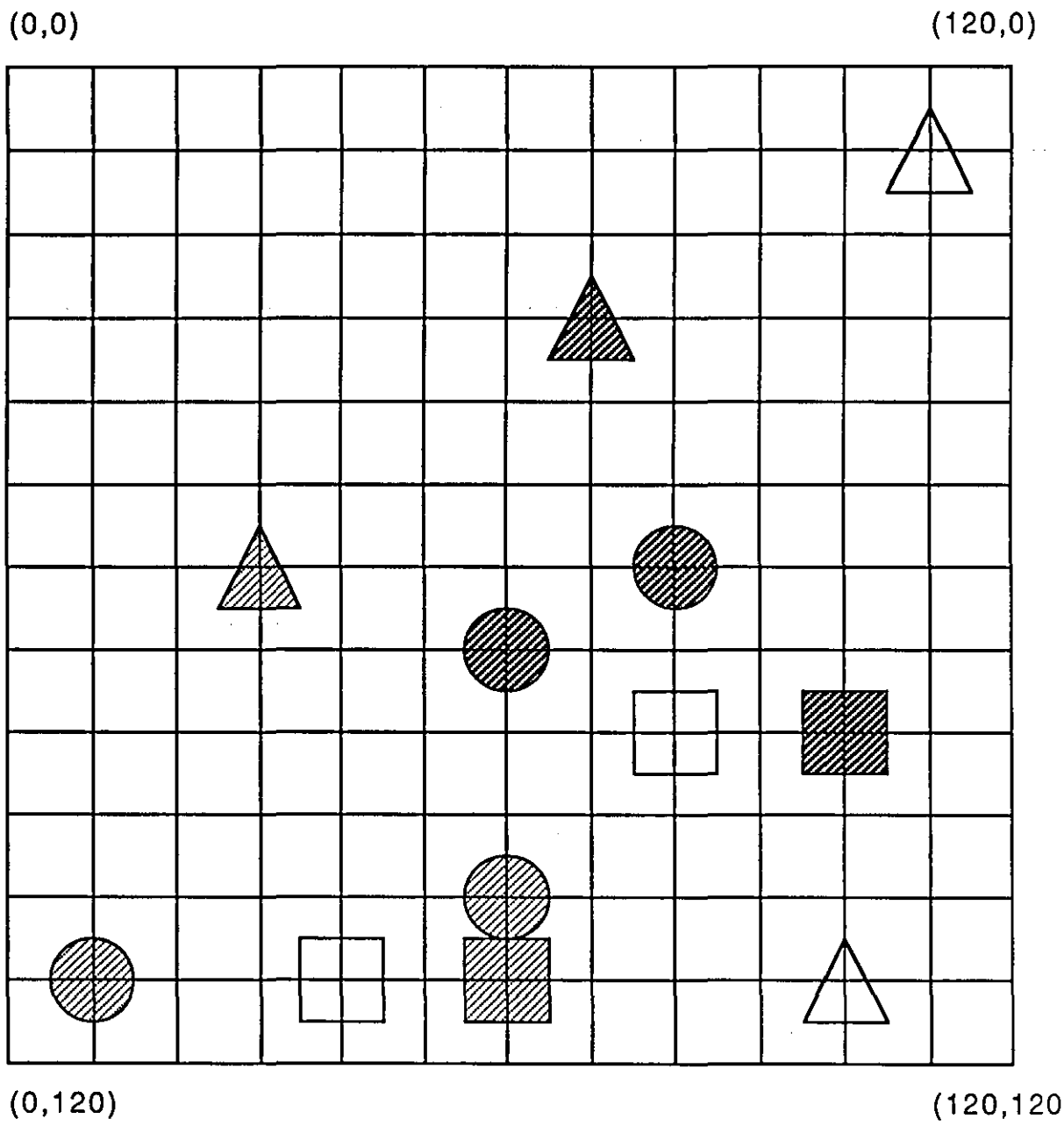


Figure 7.4. Quad Picture 4.

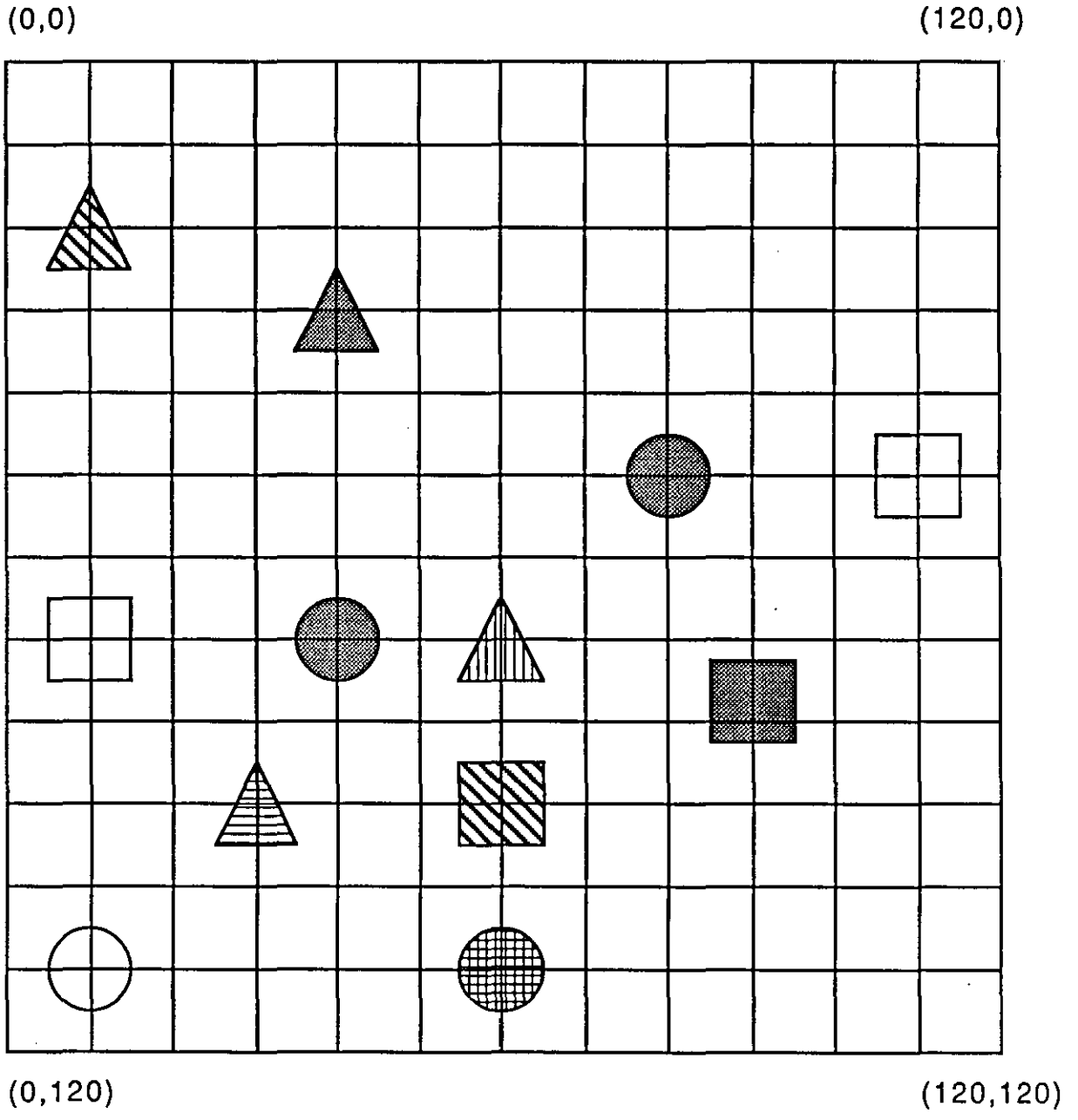


Figure 7.5. Quad Picture 5.

Accepting the complete feature set, rejecting the three partial sets and invoking the feedback procedure yielded an updated set-probability table with most of the Sets figures showing an increase compared with those in the original table, and the Matches figures all increased to 6. Alternative tables could be obtained by accepting and rejecting different combinations of sets.

7.2.3. Conclusions.

This test, though limited in its scope, can be considered to have been a complete success; the results showed that the system can perform as required when presented with data of this type, and can produce a set of rules which constitute an effective component of a recognition program even when presented with only a very limited amount of training data.

It was shown that the system can cope well with noisy data; the presence of a large proportion of spurious features did not hinder the recognition process.

Despite the fact that the quad instances in the training examples were all complete, the recognition program was able to pick up the feature sets in the test example which could represent partial or occluded images of quads. Partial objects could be recognised because in the determination of set probabilities all feature sets are used, not just maximal feature sets. If the program were to be altered to calculate probabilities from just maximal sets, the probabilities assigned to partial sets would reflect the number of occurrences of partial objects in the training pictures; the partial quads would not then be recognised. It was felt that the ability to recognise partial objects even when these

were not included in the training set was a useful one, so no such alteration was made.

7.3. TESTS USING IMAGES OF CARS.

The main tests were carried out using images of cars supplied by R.S.R.E., Malvern. Twenty-one photographs showing side views of various models of car, all facing to the right, were selected and processed: the boundaries between areas of different intensity were extracted, then the resulting line images were filtered to remove line segments whose changes in direction exceeded a given threshold. Figure 7.6 shows four typical photographs; Figure 7.7 shows the processed image produced from one of them.

7.3.1. Data Preparation.

A preliminary visual examination of the images suggested that the object parts which offered most potential for matching were the wheels, wheel arches, windows and the roof/windscreen/bonnet section. A careful examination of these parts was carried out to enable appropriate feature patterns to be selected.

The positions of the car wheels were indicated by circles of varying sizes, corresponding to either a small wheel hub, a large hub or the tyre. The wheel arches appeared in most of the images as semi-circles. The wheel circles and wheel arch semi-circles were measured, tabulated (see Table 7.1) and plotted on a histogram (Figure 7.8). It can be seen that there was a considerable amount of overlap between wheel diameters and

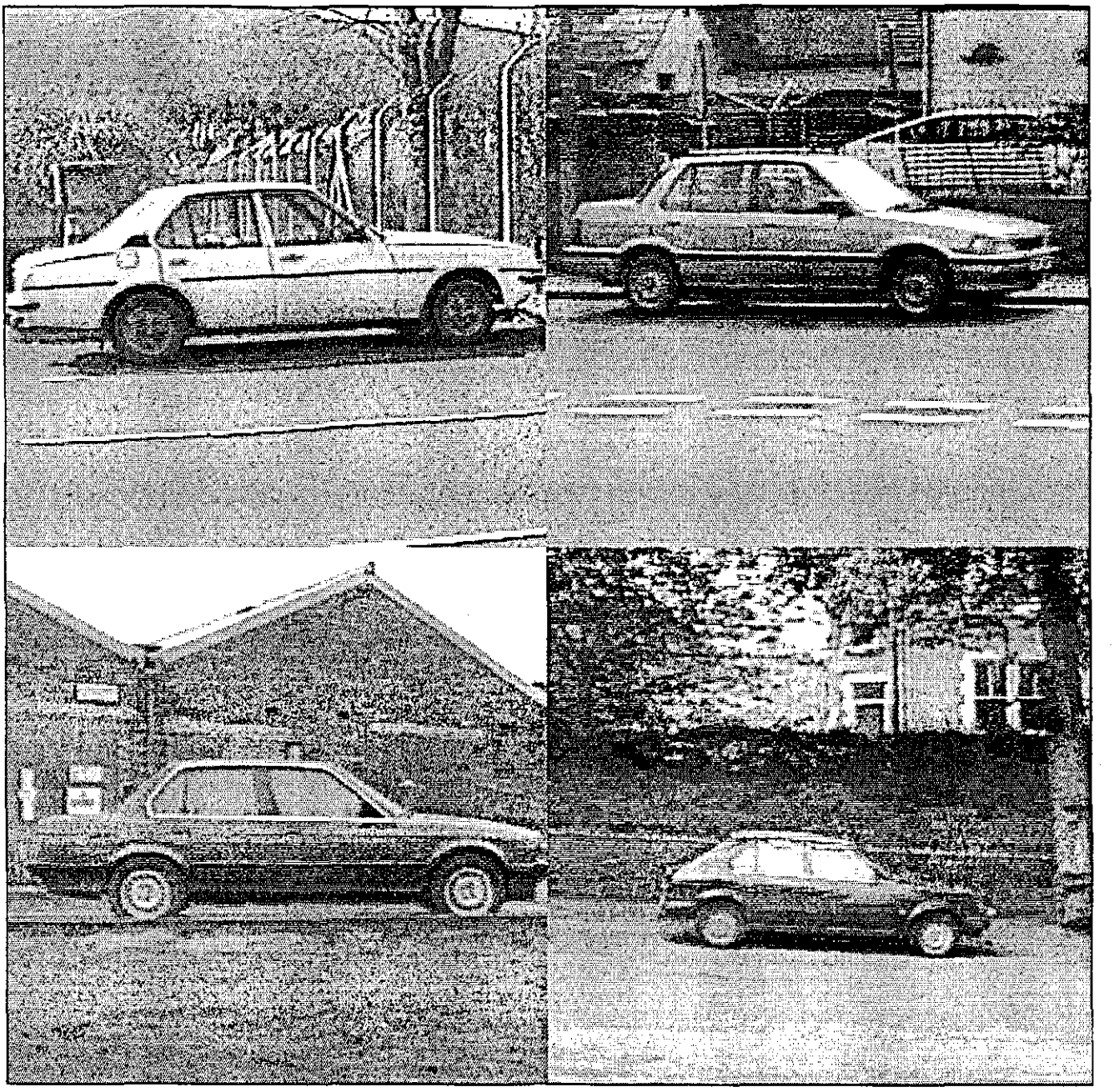


Figure 7.6. Four typical photographs of cars.

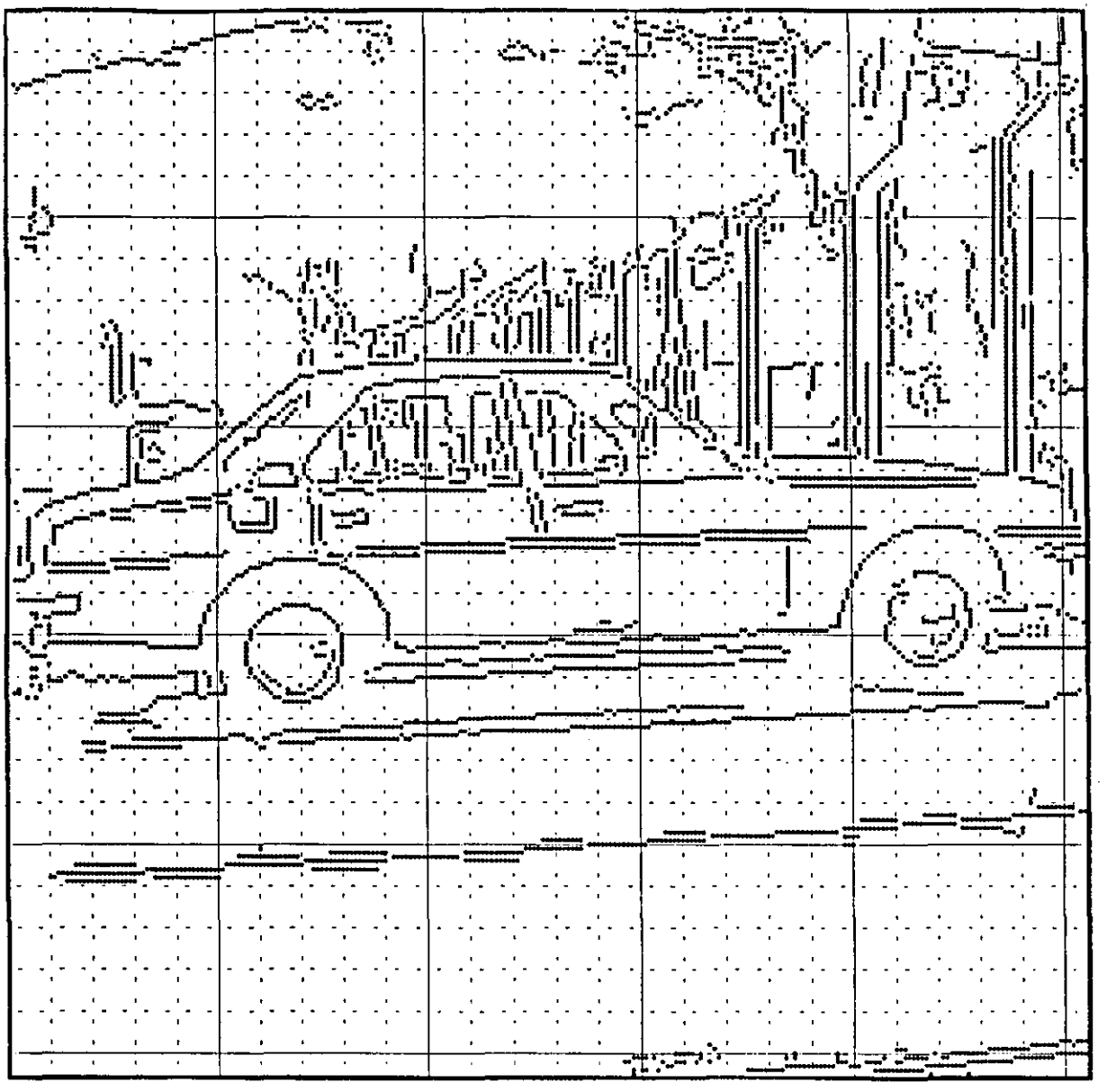


Figure 7.7. The processed image of a car.

Image No.	Wheel dia. (mm.) Rear	Front	Wheel arch dia. (mm.) Rear	Front
1	14	14	23	24
2	13	13	24	24
3	7	8	20	20
4	14	15	21	21
	19	-	-	-
5	12	12	23	23
6	14	14	-	25
	21	20	-	-
7	13	-	29	29
	19	19	-	-
8	16	-	24	27
	19	19	-	-
9	8	8	25	25
	14	14	-	-
10	10	-	20	20
	13	-	-	-
11	8	-	20	22
	12	12	-	-
12	25	21	25	27
13	-	15	26	26
	-	21	-	-
14	14	14	25	26
	-	25	-	-
15	-	14	-	23
	-	20	-	-
16	11	-	20	17
17	-	-	12	16
18	13	12	24	21
	20	-	-	-
19	12	12	22	17
	18	-	-	-
20	9	-	20	-
	13	13	-	-
	19	19	-	-
21	10	10	15	19

Table 7.1. Car wheel and wheel arch measurements.

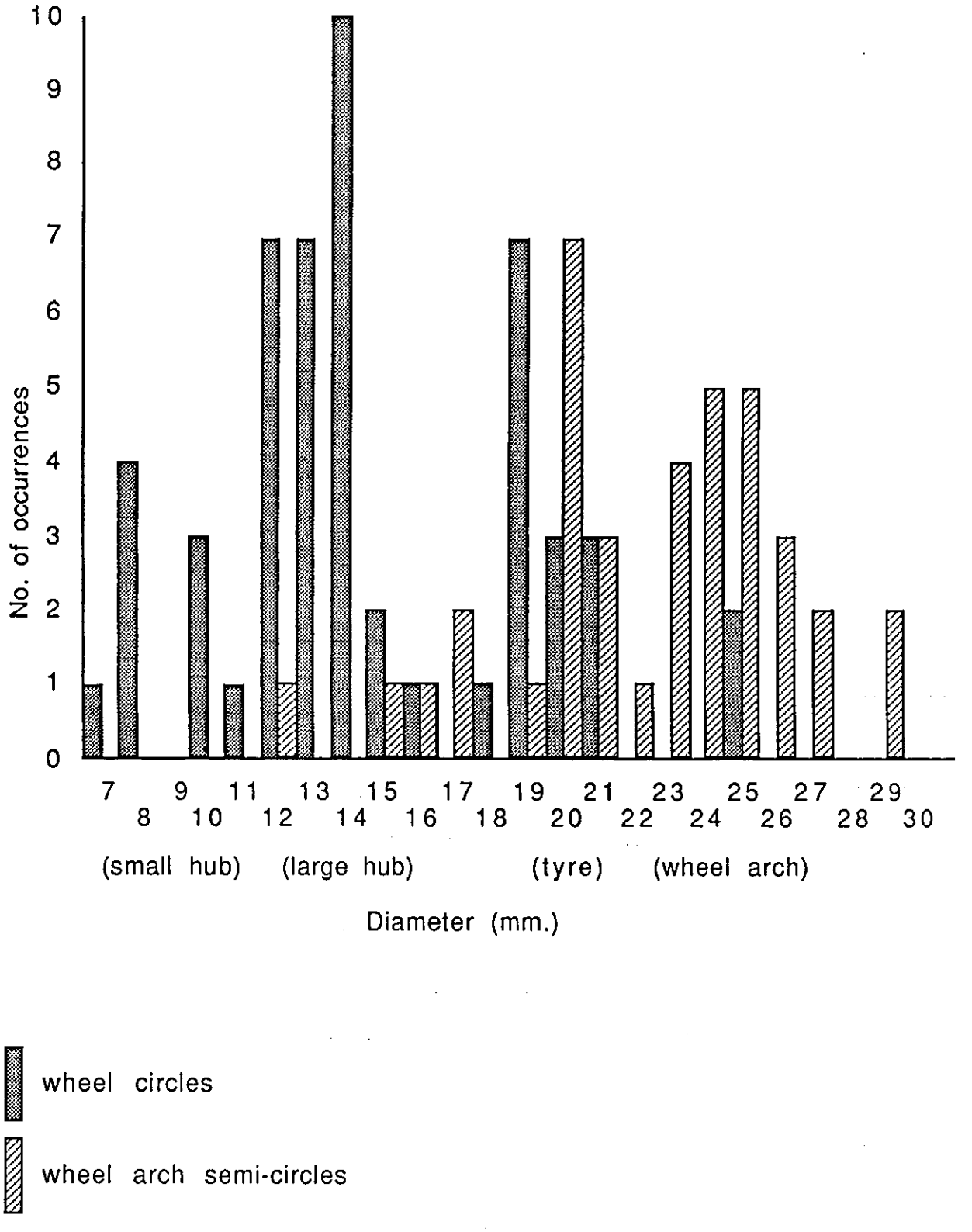


Figure 7.8. Histogram of wheel and wheel arch diameters.

wheel arch diameters. It was decided that the majority of both could be detected if circles 9, 13 and 20mm. in diameter and a semi-circle 24mm. in diameter were used as patterns.

The windows presented rather more problems. In many of the images, the car windows were indistinct - the window area frequently contained so many lines that it was impossible to decide which represented the windows. Where a clear outline could be seen, the top length, height and base length were measured; the results are given in Table 7.2. No patterns could be selected which would identify more than a small proportion of windows, so it was decided to drop windows from the part set unless they proved necessary for recognition.

For the car top - the roof/windscreen/bonnet section - the length of the roof line, the distance for which the bonnet remained approximately horizontal and the vertical distance between the roof and the bonnet were measured. The results are given in Table 7.3. Two car top patterns were selected, with vertical roof/bonnet distances of 14mm. and 17mm. The roof and bonnet lengths for both patterns were set at the minima for the images used, 30mm. and 20mm. respectively.

The initial feature set thus consisted of the six patterns shown in Figure 7.9, and the object part set consisted of five parts - two wheels, two wheel arches and the car top.

Unfortunately R.S.R.E.'s pattern matcher could not be used on the images, so the feature matching was carried out manually, trying to ensure that the data obtained was as close as possible to that which would have been produced by the feature matcher. It was assumed that

Image No	Rear window			Front window		
	Top (mm.)	Height (mm.)	Base (mm.)	Top (mm.)	Height (mm.)	Base (mm.)
1	19	17	32	15	17	30
2	-	-	-	-	-	-
3	20	11	30	14	11	24
4	20	14	32	20	14	33
5	-	-	-	-	-	-
6	18	12	34	19	12	32
7	18	14	32	-	-	-
8	16	13	28	-	-	-
9	20	14	32	17	14	30
10	14	9	21	13	9	23
11	-	-	-	-	-	-
12	20	13	32	19	14	40
13	14	15	25	10	14	22
14	20	14	32	15	14	30
15	-	-	-	-	-	-
16	-	-	-	-	-	-
17	-	-	-	-	-	-
18	20	13	24	13	13	27
19	-	-	-	-	-	-
20	-	-	-	-	-	-
21	-	-	-	-	-	-

Table 7.2. Car window measurements.

Image No.	Roof length (mm.)	Roof/bonnet distance (mm.)	Bonnet length (mm.)
1	45	17	35
2	38	16	30
3	40	13	30
4	50	17	20
5	30	14	30
6	40	17	30
7	50	17	30
8	50	17	30
9	40	15	30
10	30	12	20
11	35	15	30
12	50	18	30
13	30	18	20
14	40	17	20
15	50	17	20
16	60	17	20
17	55	17	20
18	50	17	25
19	60	15	20
20	40	14	20
21	30	13	20

Table 7.3. Car top measurements.

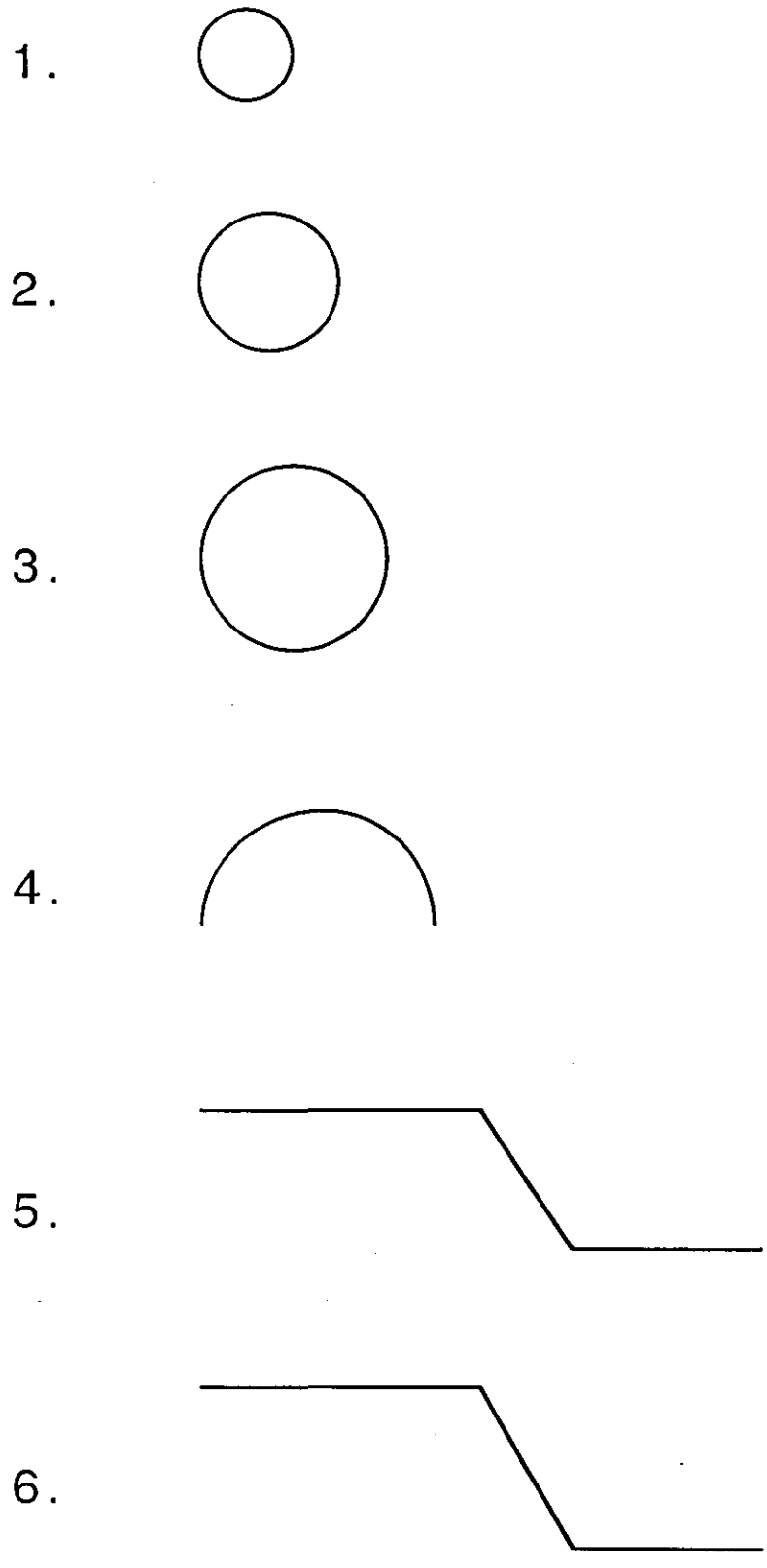


Figure 7.9. Feature patterns for car recognition.

the circle patterns would identify circles in the images with diameters within 2mm. of the pattern diameter; comparable allowances were made when matching the other patterns. The matches were rated out of 100, and only features with ratings of at least 40 were noted. This manual matching identified features corresponding to at least three of the five car parts in each image, and very few spurious features as none of the patterns occurred frequently in the image backgrounds. The system could be expected to perform adequately with this data, so no changes were made to the part or pattern sets.

The object part data consisted of the co-ordinates of those parts which could be clearly identified in the images; where the relevant area of an image was indistinct, no attempt was made to estimate the part co-ordinates. The data obtained from many of the pictures was therefore incomplete - this would allow the system's ability to learn from partial data to be investigated. The missing data related to parts for which no corresponding features had been identified by the matching process, so only the distance limits calculated by the system would be affected by the omissions; the match and set_probability tables should be the same as would have been obtained if complete part data had been available.

7.3.2. Tests Conducted and Results Obtained.

Both the training set and the recognition tests had to be drawn from the twenty-one car images available. An initial set of system runs were executed using data from just the first six images, with each in turn as the recognition test and the other five as the training set; these runs were intended to test the adequacy of the object part and pattern sets, to give some indication of how the choice of training and test images would

affect the results obtained, and to reveal what problems the use of an inadequate training set would create.

With a set of just five training examples, the induction process could not be expected to yield full, accurate rule sets. In some of the runs the match tables were incomplete, as some of the part/pattern correspondences could only be deduced from one of the six images. All of the distance limits tables were complete, but the figures they contained varied considerably - most of the twenty ranges specified in each table were too narrow. None of the set_probability tables were complete; a full table would contain 173 entries, whereas these had between 61 and 85 entries, the shorter set_probability tables being produced where the match table was incomplete.

To allow for the omissions in the set_probability table, the threshold probability for recognition was set at 0%. Despite the apparent inadequacy of the rule sets, the recognition process identified some of the features corresponding to car parts and thus correctly located the car in five out of the six runs, though in only one case was the identified set complete. In one run, two separate sets of two car parts were identified, but these could not be recognised as belonging to the same car because of a deficiency in the distance_limits table. In another run, a wheel and the corresponding wheel arch were picked up but the system incorrectly identified them as the front wheel and arch before correctly identifying them as the rear wheel and arch. The correctly identified (part number, pattern number) sets for each image were:

Image 1: {(3,4),(1,2)}

Image 2: {(5,6),(4,4),(3,4),(2,2),(1,2)}

Image 3: {(4,4),(3,4)}

Image 4: -

Image 5: {(5,5),(4,4)},{(3,4),(1,2)}

Image 6: {(4,4),(1,2)}

The fact that five out of six cars were recognised despite the smallness of the training sets suggests that the part set and pattern set used were adequate. To check for redundancy in the sets, the results given above can be examined to determine where recognition would have failed if any of the parts or patterns had been omitted. It can be seen that if part 2 (the front wheel), part 5 (the car top) and patterns 1, 3 and 6 had not been used, the five cars would still have been found. However, it was felt that the parts and patterns would all be required for recognition of the cars in some of the later images, so no alterations were made to the sets.

The feedback process was invoked at the end of each recognition run. If the results used for feedback are accurate and complete, the same set_probability table should be obtained as would result from including the test image in the training set, so to produce a table for comparison the rule induction process was run with all six images in the training set. As expected, the required table was only produced by the run where the 'accepted' set was complete; feeding back incomplete information produced variable results, improving the accuracy of some entries but reducing the accuracy of others.

Although the results obtained from the initial runs were not all equally good, the range of variation in performance suggested that selection of an appropriate training set is not as critical to success as had been expected. It was therefore decided that for the tests using all

twenty-one images, the training sets should be selected randomly.

The second set of runs were conducted using every fifth image (starting, in successive runs, with images 1, 2, 3, 4 and 5) for recognition, the remaining images forming the training set. As the feedback process had been tested adequately by the initial runs, it was not invoked here; the same rules were used for all the recognition tests in each run.

With the size of the training set approximately tripled, these runs could have been expected to produce markedly better results. The induced rule sets did, indeed, appear to show a definite improvement over those obtained in the initial runs, the ranges specified in the distance_limits tables being wider and the set_probability tables having more entries, but the recognition results were much the same - again, the system generally identified some of the object parts, but not all of them. This is perhaps partly because the higher-numbered images varied more than the low-numbered ones used for the initial tests - with a wider variety of images, a larger training set will presumably be required to produce comparable results.

The car which was completely identified in the first set of runs was initially only partially identified in the second set of runs, despite the increase in training set size. This was because the relevant probabilities given by the set_probability tables decreased, rather than increasing, as the set size increased. As feature sets are offered to the user for acceptance or rejection in decreasing order of probability rather than decreasing order of magnitude, this led to subsets of the correct feature set being presented before the full set. If a subset is accepted,

its features will be labelled 'identified' and subsequent sets containing any of these features will be rejected by the system without reference to the user - but the user cannot be expected to reject a correct subset without knowing whether the full set will be presented later.

This problem could be seen to have arisen because in some of the images, spurious features occurred in close proximity to features corresponding to object parts. For example, the image set included several cars pictured at a slight angle, with both the front and the rear edges of the windscreen visible. The feature sets for these images contained two car top features which were too far apart for either of them to have been rejected as a duplicate, but close enough for both to fall within the specified distances from the wheels and wheel arches. (See Figure 7.10). The system thus produced two complete feature sets, only one of which could be considered correct, from one (correct) set of wheels and wheel arches, leading to a lower probability being assigned to a complete set than to a wheels/wheel arches set.

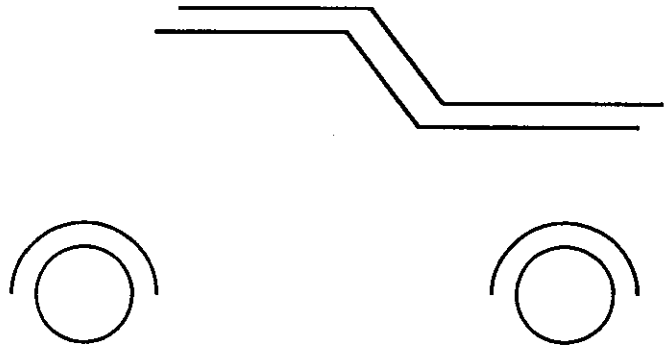


Figure 7.10. Typical features detected in an image of a car at an angle.

For the final two runs, all twenty-one images were used as the training set, then the five images from which the least successful recognition results had been obtained in the previous runs were used for testing. The first of these runs used the original definition of duplicates: features whose x and y co-ordinates differed from one another by less than 3. For the second run, the limit was raised from 3 to 5 to exclude a larger number of features. In both runs, features were only accepted if their co-ordinates exactly matched those of the corresponding object parts.

The system should, of course, be able to identify completely any car image which is exactly the same as one in the training set, so the recognition results from these two runs should have been perfect. In the first run, if all partial feature sets were rejected the system did eventually offer a complete, correct feature set for acceptance in each case, but the number of sets which had been assigned higher probabilities was sometimes very considerable: for one image, forty-nine sets had to be rejected before the required set was given. Relaxing the definition of duplicates drastically reduced this problem; in the second run, for four out of the five tests the first set offered was correct. However, recognition failed on the fifth test because the 'correct' feature had been rejected in favour of a higher-rated duplicate. If features 'within duplicate range' of the object parts had been accepted, the results would have been perfect.

7.3.3. Conclusions.

The recognition of cars from real images presented far more problems than the recognition of specially devised objects from pictures

which had been carefully compiled for the purpose. The main problem proved to be not selecting appropriate object parts and feature patterns for matching, but obtaining a set of training images which would fully define the dimensions of the object and the required part/pattern correspondences. The small set of images available clearly did not constitute an adequate training set; a much larger number of images, carefully selected to show all possible variations in car sizes and part shapes, would be required for the induction of a full and accurate set of rules for recognition.

The training and test images used contained only positive examples of cars - there were no negative examples or 'near misses'. The fact that positive probabilities of recognition were obtained even when the test car image lay partly outside the range specified by the training examples suggests that the system will have problems detecting 'near misses'; this approach seems most appropriate for use with fuzzy object categories, where there is no definite boundary beyond which the recognition process should definitely fail.

The test using synthetic data showed that the system can cope well with background noise, but these tests revealed that object-based noise - spurious 'shadow' features detected in the near vicinity of object parts - is much more of a problem. Background noise can be eliminated effectively by the distance-limits rules, but these will not eliminate all object-based noise unless the limits are much more tightly defined than is likely to be the case with a broad object category such as cars. The edit module does remove duplicates, i.e. features which lie very close to higher-rated features, but as 'shadows' can be more distinct than object parts in the processed images, there is a clear risk that the 'real' feature

will be discarded and the 'shadow' retained, so relaxing the definition of duplicates to remove a larger number of features does not appear to be a good option for solving the 'shadow' problem unless the user is prepared to accept object parts being located to 'within duplicate range'. The best approach seems to be to take care to select for feature matching object parts which are rarely 'shadowed'.

If a set of just two features is to be considered sufficient to give a positive probability of object recognition, there are obvious dangers in selecting a part set which contains pairs of parts such as the front wheel and wheel arch and the back wheel and wheel arch of a car which can easily be mistaken for one another. The risk of confusion arising here could perhaps have been avoided by treating a wheel and the corresponding arch as a single object part, to be matched by composite circle/semi-circle feature patterns. The rating thresholds for the patterns could be set low enough to ensure detection of the part if either the wheel or the arch were to be clearly visible in the image.

Despite the fact that the training sets and the object part set used had significant deficiencies, the system performed surprisingly well, locating the cars successfully in most of the tests conducted. It is clear that as well as offering the potential for inducing reliable, accurate recognition programs from comprehensive training sets, this system can be used to produce a program which, though far from perfect, can yield useful results even when only minimal training data is available.

CHAPTER 8

DEVELOPMENT OF THE FINAL SYSTEM

8.1. SYSTEM STRUCTURE.

The aim of this project was to produce a system which would learn to recognise a number of different objects/object views from feature data and data giving the locations of object parts in a set of training images, then use the induced recognition rules to provide the most probable identifications of the objects in test pictures, using feedback information about the accuracy of the results to update the probabilistic element of the rules. Such a system would contain a considerable number of components, and it was intended that a blackboard system should be used to enable these components to communicate with one another and to access the image data.

A standard blackboard system comprises a blackboard data structure and a set of independent knowledge sources, together with a control component/scheduler (see Section 4.2.). Several systems have been developed for image understanding applications; these include the UMass Schema System (Draper et al., '88), described in Section 4.5.3., and Nagao et al.'s system for analysis of complex aerial photographs (Nagao et al., '88). Both these systems have separate independent knowledge sources for each type of object to be recognised, together with a number of object-independent knowledge sources. Neither system employs rule induction or feedback - their object-specific knowledge sources were developed separately, then incorporated into the blackboard system.

If a learning element is to be included in the system, clearly it will

not be possible to have a separate, independent, immutable knowledge source for each type of object. It was suggested in Section 4.7. that a two-tier blackboard system could be used, with the problem data on the bottom tier, the object recognition knowledge sources on the top tier and the rule induction and feedback modules, which would act as meta-knowledge sources using the object recognition knowledge sources as data, above both, as indicated in Figure 8.1.

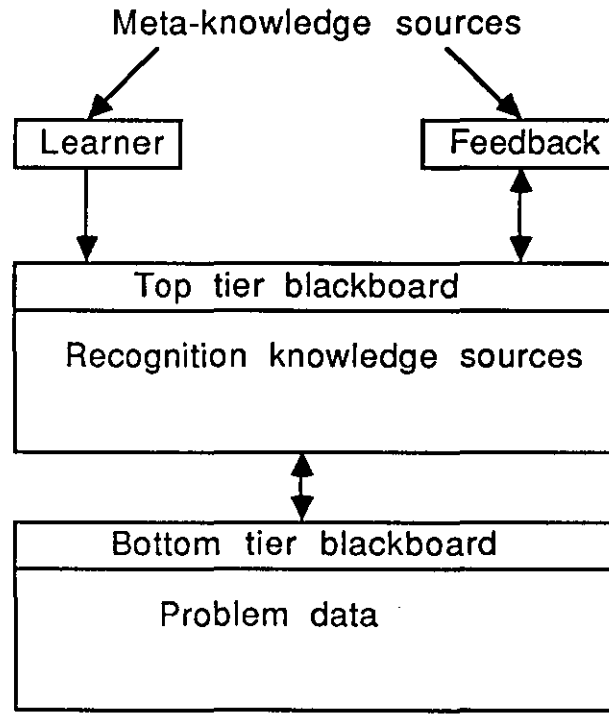


Figure 8.1. Plan of a two-tier blackboard system.

The system is to be based on the single-object learner/recogniser system described in Chapter 6. Each object recogniser will consist of the recogniser program described in Section 6.4.3., together with the object-specific rules developed by the learner described in Section 6.4.2. Rather than duplicating the recogniser program in each of a set of independent object recogniser knowledge sources, it seems sensible to

treat the recogniser program as one knowledge source which acts in co-operation with each set of object-specific rules. It is these rules, rather than the recogniser program, which are subject to alteration, so the recogniser program can be removed from the top-tier blackboard, leaving it holding just the induced rules.

The system is to be written in Prolog, which does not oblige the programmer to distinguish between program and data, so the induced rules, which are treated as data by the learner and feedback modules, can also be treated as data by the recogniser program; the two-tier blackboard can thus be replaced by two parallel blackboards, one containing problem data and the other containing recognition data.

Figure 8.1. shows the learner and feedback modules accessing just the top-tier blackboard, but in fact both need to have access to the bottom tier as well. There are also other knowledge sources, such as the edit module described in Section 6.4.1., which operate on the problem data. None of the knowledge sources use all the types of data which are held on the two blackboards; the required structure of the system can be clarified by partitioning both blackboards, and allowing the knowledge sources access to only those sections which are relevant to their operation.

The final structure is depicted in Figure 8.2. It can be seen that the novel two-tier system has been translated into something very similar to a standard blackboard system; it would be possible to replace the two parallel blackboards with a single blackboard partitioned into two sections, one holding the problem data and the other holding the recognition data, but it was felt that retaining separate blackboards for

the two types of data would make the way in which the system is constructed clearer.

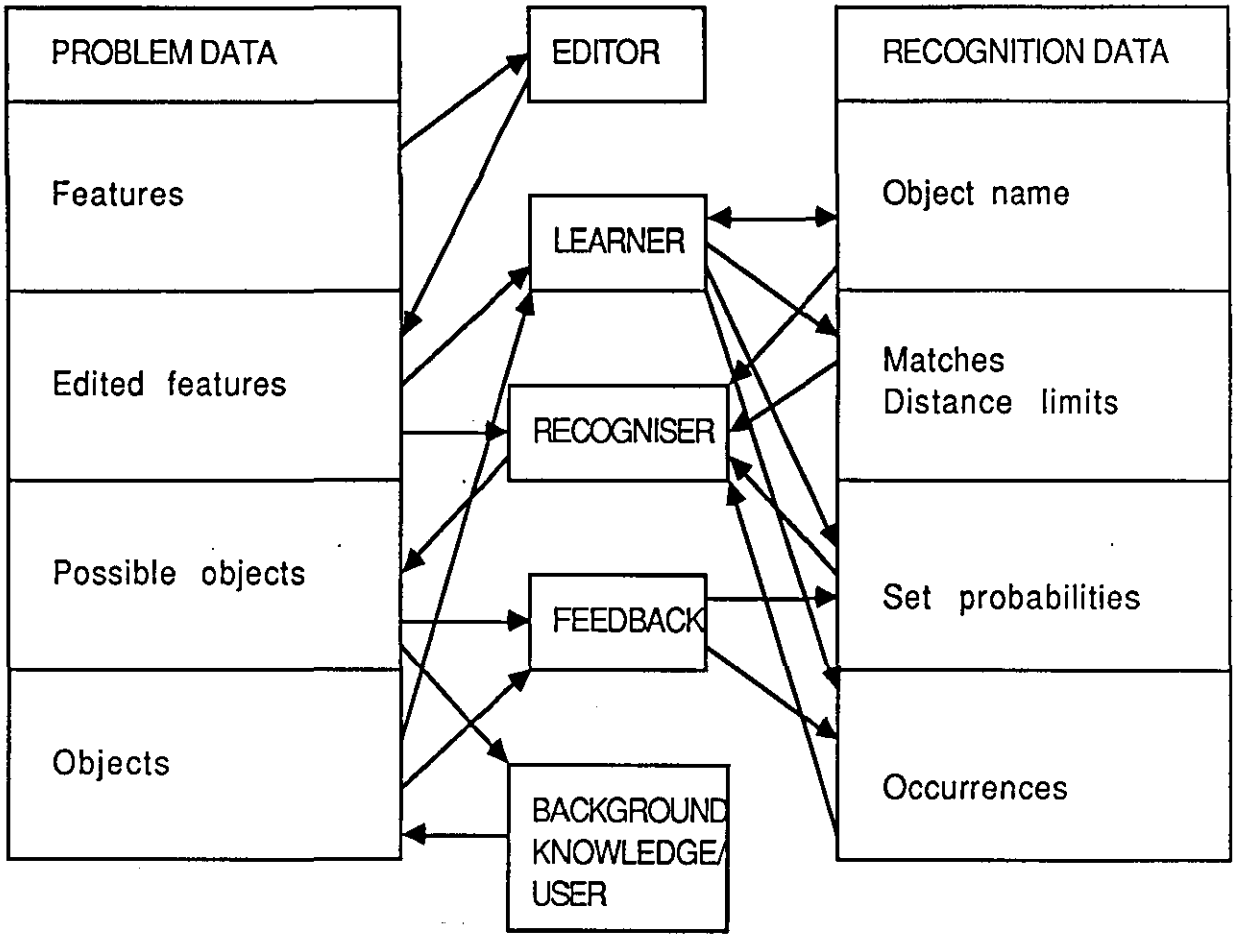


Figure 8.2. The system structure.

It is the decision to treat the induced rules as data which has made this simplification possible. The use of Prolog as a programming language makes the implementation of this decision a trivial matter - in Prolog, the translation of a statement from the status of 'rule' to that of 'data' does not require any alteration of structure or format - but the same approach could be adopted , though with more difficulty, were a procedural, rather than declarative, language to be used. A procedural

program could be provided with a set of standard rule patterns containing variables rather than constants, and each rule could be represented by a list of data items comprising a pattern identifier and a list of values to which the variables in the pattern are to be instantiated. For example, the rule:

If shape is circle and size is small then object is ball

could be represented by:

Pattern1: If shape is X and size is Y then object is Z

DATA: 1,circle,small,ball

The original intention was to base the blackboard system on either the Edinburgh Prolog blackboard shell (Jones et al., '88) described in Section 4.6.2. or the LUMP truth-maintained blackboard (Hinde et al., '89) described in Section 5.5.6., but both of these seemed unnecessarily complex - they offered facilities which would not be required here - so to ensure that the basic system structure would not be 'buried' in unnecessary complications, a simple blackboard was written from scratch. The design of the system as a whole is described in the next section; the knowledge sources are described in more detail in Section 8.3, and the testing of the complete system is described in Chapter 9.

8.2. DESIGN OF THE BLACKBOARD SYSTEM.

8.2.1. Introduction.

The first stage in designing a software system normally involves establishing exactly what task the system is required to perform: what sort of input will be provided, what options should be available to the

system user and what output is to be produced, then breaking the task down into separate stages and drawing up a flowchart which depicts these stages and the way in which they follow on from one another, illustrating graphically the way in which the system will operate.

If the system is intended to exhibit intelligent behaviour, such a specification of the operations to be performed and the order in which they are to take place cannot be considered appropriate - an A.I. system is not intended to follow a preset path, but to decide for itself at each stage of its operation which of the actions available to it should be performed next in the light of the circumstances under which it is running, basing its decisions on heuristic rules, estimates and measures of the expected utility of various operations etc. However, a detailed analysis of the behaviour required of the system is still a prerequisite of the design task. The target behaviour is described in Section 8.2.2.

For a blackboard system, there are three distinct major elements to be designed: the blackboard itself (or blackboards, if more than one is to be used), the knowledge sources and the controller/scheduler. The designs of these are inter-related, and are also related to the nature of the problem(s) to be solved and the structure of the problem space. As this blackboard system was based on an existing system (the single-object recogniser) from which the main knowledge sources were derived, the knowledge sources are considered first here, in Section 6.2.3, followed by the blackboard structure in Section 6.2.4. and the control element in Section 6.2.5. (With a system designed from scratch, or based on a general-purpose blackboard shell, a different design order might be more appropriate). Other aspects of the system such as the initialisation process and the user interface are described in Section

6.2.6. There is inevitably a certain amount of overlap between the sections; the design of the blackboard structure, for example, necessitated changes to the knowledge sources.

8.2.2. Required system behaviour.

The behaviour required of the blackboard system is an extension of the behaviour of the single-object system; the data format should not require any alteration, and when presented with data on a single object this system should produce the same results as the single-object system, though the required operations (editing feature data from training examples, learning recognition rules, editing data from a test image, searching for possible object instances, confirming/rejecting possible objects, feeding back results to update the rules) should be programmed automatically by the blackboard controller instead of being specified individually by the user.

Where a number of different objects are to be identified, the user should be able to supply either a separate set of training data for each object, or a composite training set containing data on all the objects/object views to be learnt; in the latter case, it should be possible for a single training image to contain instances of more than one object. It should not be necessary for the user to specify the object(s) to be learnt; when the system is presented with part data for an unknown object, the object should be learnt automatically.

When all previously unknown objects have been learnt, the feature data from training examples could be deleted, or it could be retained for use as negative examples of future objects to be learnt. It was decided

that the latter approach should be adopted, as the inclusion of additional negative examples in a training set should improve the accuracy of the induced probabilities. (It is assumed that training data will include part data for all the objects in each image which the system has not already learnt and may be required to identify, so earlier sets of training examples will not include unrecognised instances of a new object to be learnt). Similarly, all new images, including training images for new objects to be learnt, should be regarded as test images for all previously learnt objects and recognition results from them should be used for feedback purposes - this will ensure that where a separate set of training images is provided for each object to be learnt, the order of presentation of the training sets will not affect the results obtained.

Feature data from test images could be similarly preserved for inclusion in future training sets, but as test images will not be subject to the same vetting procedure as training images it was felt that they could not be guaranteed not to contain unidentified instances of a new object, and if all data were to be preserved the system would eventually become choked with data and the running speed would deteriorate, so it was decided that feature data from test images should be deleted at the end of the feedback process.

When searching test pictures, the system should search for the most likely object first, estimating relative likelihoods by counting the number of occurrences of each object. Feature sets which have a high probability of representing instances of an object should be accepted or rejected immediately, and the component features should be marked as 'identified' so that the system need not search for less likely objects if there are no appropriate features remaining unidentified. Decisions on

feature sets with lower probabilities should be deferred until the search procedure has been completed. The user should be able to adjust the probability thresholds for immediate decisions and deferral during the operation of the system, to allow for the fact that probabilities calculated by the system may be very inaccurate initially, but will improve in accuracy as the number of images examined increases. Separate thresholds could be set for each object, but it was decided that for simplicity the same thresholds would be applied to all objects.

The single-object system referred all acceptance/rejection decisions to the user, but it was thought desirable to automate the decision-making process as far as possible in the final system. It should be possible to specify object-specific background checks to be conducted on candidate feature sets. Any sets which fail the checks should be rejected immediately; of the sets which pass, those with probabilities above the immediate decision threshold should be accepted automatically and only those with lower probabilities on which a deferred decision is required should be referred to the user.

The results of each search conducted should be used for feedback purposes, but where the objects in an image have been satisfactorily identified without searching for all known objects, it was decided that further searches should not be conducted merely to obtain further data for feedback.

The single-object system provided an automatic printout of all the recognition rules at the conclusion of the learning process, and of the set-probability rules after feedback. This is not required here; instead, the user should be able to request a printout of the rules for a specified

object.

The system could run through all the operations available to it before consulting the user for further instructions (automatic bid execution), or consult the user at the conclusion of each separate operation (manual bid execution). The user should be able to decide which of these options is required, and it should be possible to switch from one to the other during the operation of the system.

This specification will mean that there are a considerable number of options available to the user. Some choices will have to be made as part of the system initialisation procedure; it was decided that to detail the options available during a system run, the final facility to be incorporated should be a 'help' module.

8.2.3. Knowledge sources.

The editor, learner, recogniser and feedback modules described in Chapter 6, with such modifications as proved necessary, formed the main knowledge sources for the system. It was decided that the background knowledge/user box in Figure 8.2. should be represented by two knowledge sources, Acceptor and Selector: the system user would select an acceptance threshold and a referral threshold, which could be adjusted when required during the operation of the system; feature sets with probabilities equal to or above the acceptance threshold would be handled by the Acceptor, which would deal with them automatically, without referral to the user, and feature sets with probabilities between the referral threshold and the acceptance threshold would be handled by the Selector which would refer its decisions to the user for

confirmation/rejection, sets with probabilities below the referral threshold being discarded.

This arrangement allows the user to gradually relinquish control of the image identification process as feedback improves the accuracy of the probabilities assigned to feature sets; the intention is that initially the acceptance threshold should be set at above 100% and the referral threshold should be set at 0%, then the acceptance threshold should be lowered and the referral threshold raised until eventually the two coincide, when the identification process would be entirely automatic.

An additional knowledge source, Remover, is required to remove data on test images from the blackboard after the identification process has been completed. There are thus seven knowledge sources altogether: Editor, Learner, Recogniser, Feedback, Acceptor, Selector and Remover. The internal operation of these is described in Section 8.3.

8.2.4. Blackboard organisation.

In order to establish which information should be held on each of the blackboards and how it could best be organised, the inputs, outputs and data to be read were determined for each of the knowledge sources to be used. The distinction between inputs and data to be read was made so that blackboard entries which would be altered or deleted by the knowledge source (the inputs) could be distinguished from those which would be required but would remain unchanged (the read data). Inputs which are altered also appear in the outputs list; inputs which are deleted do not.

The information for each knowledge source was as follows:

Knowledge Source		Problem data	Recognition data
Editor	**Input:**	features	-
	Read:	-	rating thresholds
	Output:	edited features	-
		identified_features	
Learner	**Input:**	object parts	list of object names
			occurrences
	Read:	edited features	object part list
	Output:	-	match table
			distance_limits table
			set_probability table
			list of object names
			occurrences
Recogniser	**Input:**	-	-
	Read:	edited features	match table
			distance_limits table
			set_probability table
	Output:	feature sets	-
		probability list	
Acceptor	**Input:**	probability list	occurrences
		identified_features	
	Read:	feature sets	-
	Output:	object parts	occurrences
		identified_features	
Selector	**Input:**	probability list	occurrences
		identified_features	
	Read:	feature sets	-
	Output:	object parts	occurrences
		identified_features	
Feedback	**Input:**	object parts	set_probability table
		feature sets	
		edited features	
	Read:	-	-
	Output:	-	set_probability table

Remover	**Input:**	edited features identified_features feature sets	-
	Read:	-	-
	Output:	-	-

After analysis of this information, grouping together items which commonly occur together, and taking into account the structure of the problem space, it was decided that it would be most convenient to arrange the problem blackboard in four sections, one for unedited feature data, one for edited feature data, one for possible objects and one for actual objects, and that these sections should contain:

problembb1: unedited features

problembb2: edited features, identified features

problembb3: feature sets, probability list

problembb4: object parts

The recognition blackboard was also arranged in four sections, containing:

recognisebb1: object names, part lists, rating thresholds

recognisebb2: match table, distance_limits table

recognisebb3: set_probability table

recognisebb4: occurrences

The input/read/output requirements for each knowledge source are summarised in Table 8.1.

For clarity, it was decided that the same format should be used for all entries on both blackboards:

<Blackboardname>bb<SectNo>(<Datatype>,(<DataItems>)).

For example, edited features would be entered on the problem blackboard

Knowledge Source	Problembb Section			Recognisebb Section		
	Input	Read	Output	Input	Read	Output
Editor	1	-	2	-	1	-
Learner	4	2	-	1,4	1	1,2,3,4
Recogniser	-	2	3	-	2,3	-
Acceptor	3	3	2,4	4	-	4
Selector	3	3	2,4	4	-	4
Feedback	2,3,4	-	-	3	-	3
Remover	2,3	-	-	-	-	-

Table 8.1. Knowledge Source Requirements for Blackboard Access.

in the form:

problembb2(feature,(PicNo,FeatureNo,PatternNo,X,Y)).

and matches would be entered on the recognition blackboard as:

recognisebb2(match,(Object,PartNo,PatternNo)).

The knowledge sources were modified as necessary to use these data formats. (No changes were made to the format of data internal to a knowledge source, e.g. the matched_feature table used by Learner, so that such internal data could be distinguished easily from blackboard entries).

The input data format is the same as for the single-object system: the object data supplied consists of a part list for each object to be recognised and rating thresholds for each feature pattern, the training data consists of a list of features and object_parts for each image, and the test data consists of just features, with the formats:

part_list(Object,[(1,PartName1),(2,PartName2),...]).

rating_threshold(PatternNo,Threshold).

feature(PicNo,PatternNo,X,Y,Rating).

object_part(Object,PicNo,InstNo,PartNo,X,Y).

When data is read in a special predicate, **bb_enter**, is called to enter it on the blackboards: part lists and rating thresholds on recognisebb1, feature data on problembb1 and object_part data on problembb4.

Information about new blackboard entries is passed to the controller/scheduler (see Section 8.2.5), which checks to see which knowledge sources require these entries, notifies their bidders so they can bid for the operations they could perform, and schedules the bids. As features and object_parts are entered in batches, notifying the controller of each individual entry would be very inefficient; instead, **bb_enter** flags each type of entry, using

bb_entered(object_part,Object,PicNo).

and **bb_entered(feature,PicNo).**

When all possible entries have been made, the flags are checked and the appropriate notifications are carried out.

The final task performed by **bb_enter** is to identify and label training images, to distinguish them from test images. This is necessary because the feature data from test images is to be removed after identification, while the feature data from training images is to be retained. The predicate **training** is used to identify training images; the format is simply

training(PicNo).

The **training** clauses are not held on either of the blackboards as they contain 'system' information required only during the process of making and scheduling bids, not problem or recognition data.

8.2.5. The controller and scheduler.

The controller/scheduler of a blackboard system is responsible for selecting operations to be performed, and instructing their execution; it must be able to identify the possible knowledge source operations which could be carried out at any point, and to assign an order of priorities to these operations so that those with the highest priority can be selected for execution first. (See Section 4.2.). It must also allow the user to intervene where necessary during the operation of the system, to enter new data or to examine the state of the blackboard.

The design process can be simplified by separating the control and scheduling functions: the scheduler is then responsible for maintaining a list of ranked knowledge source bids, which specify potential operations and their priorities, and the controller is responsible for selecting the highest-ranked bid from this list and instructing its execution, and for liaising with the system user. As the design of the controller will depend on whether the maintenance of an up-to-date bid list can be carried out independently by the scheduler, or requires supervision by the controller, the design of the scheduler is considered first.

The starting point for the development of a method for making and ranking bids was again an analysis of the knowledge sources: the pre-conditions which must be satisfied before each knowledge source can make a bid to perform its operation and the restrictions on the order in

which the bids can be executed were analysed as follows:

Editor **Pre-conditions:** New entries of feature data on problembb1
Restrictions: Must be the first bids executed

Learner **Pre-conditions:** New entries of object part data on problembb4, object name not in list of learnt objects
Restrictions: Must follow all editor bids to ensure edited feature data is available for all images in training set; must precede Recogniser bids, so as many different objects as possible can be sought in test images

Recogniser **Pre-conditions:** New entries of edited feature data on problembb2, object in list of learnt objects, no object part data for this image, object on problembb4
Restrictions: Recogniser bids should be executed in decreasing order of no. of occurrences of object

Acceptor **Pre-conditions:** New probability list on problembb3, probability of first element in list is equal to or above acceptance threshold
Restrictions: Must precede Recogniser bids, so if all possible features in image have been identified recognition of further objects can be aborted

Selector **Pre-conditions:** New probability list on problembb3
Restrictions: must follow all Recogniser bids

Feedback	**Pre-conditions:**	Probability list removed from problembb3
	Restrictions:	Must precede Recogniser bids, so that probabilities used by Recogniser are as up-to-date as possible.
Remover	**Pre-conditions:**	Probability list removed from problembb3, image not a training example
	Restrictions:	Must follow Feedback bids

All the knowledge sources have as one of their pre-conditions the addition of an entry to, or the removal of an entry from, the problem blackboard so it was decided that these events should be used to trigger the bid creation process.

The triggering condition for a knowledge source is specified by a **wants** or **wants_removed** clause which comes at the start of its bidder program file and has the format:

wants(KS,PBBSection,Datatype).

or **wants_removed(KS,PBBSection,Datatype).**

The way in which the triggers are used is as follows: when one of the knowledge sources or the data entry process described in the previous section enters some new data on the problem blackboard, a clause of the form:

new_entries([[PBBSection1,Datatype1,(Data1)],

. . . . [PBBSection*m*,Datatype*m*,(Data*m*)]]).

is used to invoke the **new_entries** predicate, which takes each new entry in turn and searches for all the **wants** clauses which match it.

Similarly, when data is removed from the blackboard, a clause of the form:

removed_entries([[PBBSection1,Datatype1,(Data2)],
. . . . [PBBSection*n*,Datatype*n*,(Data*n*)]]).

is used to invoke **removed_entries,** which searches for matches between data which has been removed and **wants_removed** clauses. Whenever a match is found, the relevant information is passed on to the knowledge source bidder by:

make_bid(KnowledgeSource,[PBBSection,Datatype,Data]).

The **make_bid** predicate for each knowledge source, the definition of which forms the main part of the bidder program file, is responsible for checking any remaining pre-conditions, making bids and passing them on to the scheduler. Ratings could be attached to the bids by either **make_bid** or the scheduler; it was decided that as the rating given to a bid would depend on the knowledge source, this task should also be assigned to **make_bid.**

The bid ratings are required to embody the restrictions given in the above analysis of knowledge sources. These give the following partial ordering of the knowledge source operations (where < is to be interpreted as "must precede"):

Editor < Learner < Recogniser

Acceptor < Recogniser < Selector

Feedback < Recogniser < Remover

To obtain a complete ordering, it is necessary to establish orders of priority for the operations which must precede and succeed Recogniser bids. The pre-conditions for bid execution are such that Learner, Acceptor and Feedback bids cannot exist for the same image at the same

time, so the relative priorities attached to these operations are irrelevant. Similarly, Selector and Remover bids cannot exist for the same image at the same time, so these operations too can be ordered randomly. It was decided that the following total ordering should be used:

Editor < Learner < Feedback < Acceptor < Recogniser < Selector < Remover.

The operations with the highest precedence must be awarded the highest ratings; the priority to be given to Recogniser bids is to depend on the number of occurrences of the object being sought, so a range of ratings is required for Recogniser. The ratings selected are shown in Table 8.2.

Knowledge Source	Rating
Editor	100
Learner	90
Feedback	80
Acceptor	70
Recogniser	40 - 60
Selector	30
Remover	20

Table 8.2. Knowledge Source Bid Ratings.

The bid details are passed to the scheduler by:

schedule(KnowledgeSource,(BidData),Rating).

The operation performed by **schedule** consists merely of inserting the bid details into a bid list arranged in decreasing order of rating. This list is set up during the system initialisation process (described in Section 6.2.6).

When the bid creation and scheduling process has been completed, execution of the system will continue from the point at which it was interrupted, i.e. from the clause following the **new_entries** or **removed_entries** clause. As the operations which will be interrupted for bid creation do not use the bid list, which is 'system' information, used only by the controller and scheduler (which do not have access to the problem blackboard and so cannot instigate the bid creation process themselves), the creation of new bids will not have any effect on the way in which the interrupted operation is carried out. The **new_entries** and **removed_entries** clauses can therefore be placed at any point from where they will be called exactly once during the execution of the process during which the blackboard changes they refer to are made.

The decision to call the bid creation and scheduling process directly from the knowledge sources and the data entry process makes it independent of the blackboard controller, thus simplifying the design of the control mechanism. The basic control loop is very simple:

(1). Remove the top-ranked bid from the bid list.

(2). Execute this bid.

(3). Repeat from (1) until the bid list is empty.

This loop needs to be modified slightly to allow the system user to intervene when necessary. The specification given in Section 8.2.2. requires the user to be given the option of automatic bid execution (the

controller consults the user for further instructions only when the bid list is empty) or manual bid execution (the user is consulted after each operation), which can be achieved by replacing stage (3) above by:

(3a). If manual bid execution selected

Then consult user

Else Repeat from (1) until bid list is empty.

This modified loop is embodied in the controller predicate **run**. The user's choice of manual or automatic bid execution, which is established as part of the initialisation process and can be altered during the user consultation process (see Section 6.2.6), is recorded by:

autorun(m). (manual)

or **autorun(a).** (automatic)

8.2.6. Miscellaneous processes: initialisation, user consultation.

The initialisation process, **init**, is responsible for establishing a number of different parameters and variables. The process sets the initial values of the bid list (system information, not held on a blackboard) and the list of learnt object names (on recognisebb1), both empty lists, and the total number of object occurrences provided as training examples or recognised in test images (on recognisebb4; required in the calculation of Recogniser bid ratings), initially zero. The user-selectable parameters are read in; these are the duplicates limit (i.e. the distance within which features are to be regarded as duplicates, which was fixed in the single-object system, but is selectable here), the acceptance and referral thresholds (discussed in Section 8.2.3) and the choice of manual or automatic bid execution.

The read routines used provide basic error-trapping: for numbers the

minimum and maximum acceptable values are specified and checked, for characters a list of alternatives is specified and checked. For the duplicates limit, the range of acceptable values is 0 to 10; features are regarded as duplicates only if the distance between them is strictly less than the limit, so a limit of 0 will mean that no features will be rejected as duplicates. The acceptance and referral thresholds are specified as percentage probabilities; candidate feature sets will be automatically accepted as object instances if their probability is equal to or above the acceptance threshold, so while the system is being trained, when no automatic acceptance is required, this threshold must be set at above 100; the maximum and minimum values here are therefore 0 and 110, with a suggested start level of 110. The minimum value for the referral threshold is also 0, and its maximum value is equal to the acceptance threshold (which is set first).

When the initialisation process is complete control is passed to the user consultation process, **consult_user**, which is also called whenever the bid list is empty and - in the case of manual bid execution - after the execution of each bid. This process provides the user with the facilities outlined in the system specification in Section 8.2.2. There is a **help** facility, advertised every time **consult_user** is called, which lists the options available. These depend to some extent on whether manual or automatic bid execution has been selected; **help** has been designed to give only those options which currently apply. The full range of possibilities is:

- read in a data file
- execute the next bid (manual bid execution only)
- switch to automatic (manual bid execution only)
- switch to manual (automatic bid execution only)

- view the recognition rules for an object
- alter the acceptance/referral thresholds
- quit the system.

Commands not on the list of options will not be recognised; if the user wishes to perform an operation which has not been specified, this can best be accomplished by quitting the system (**quit.**), then returning when the operation has been completed by entering **consult_user.** or **run.**

The command to read in a data file is:

[Filename].

This will read only a single file - a list of file names is not acceptable. When a file has been read, the system will call **bb_enter** to enter the data it contains on the problem blackboard; this in turn will initiate the bid creation process, and Learner bids for any new objects on which training data has been provided will be scheduled. If automatic bid execution has been selected, the bids on the bid list will be executed before the user is consulted again. This means that if data from a set of training examples is contained in a number of different files, they must be read in together by listing them in a single master file; if the user attempts to read them sequentially, the object(s) will be learnt from just the data in the first file. With manual execution the files can be read in sequentially.

The commands to switch from manual to automatic bid execution or vice-versa, **auto.** and **man.**, are straightforward: the existing **autorun** clause is retracted and a new one is substituted.

The recognition rules for an object can be viewed by entering:

show_rules(Object).

The print format used for the recognition rules in the single-object system has been altered considerably to improve readability. The new format can be seen from the listings of sample system runs in Appendix B.

To alter the thresholds for acceptance and referral, the command is **alter**. The system prints out the current values, then prompts the user to enter the new ones. Both values must be entered even if only one is to be changed.

8.3. THE KNOWLEDGE SOURCES.

8.3.1. Introduction.

Each of the knowledge sources used in this system has the same overall format. There is a master file which specifies all the program files which constitute the knowledge source; the first program file is the bidder, which is followed by between one and eight files which make up the body of the knowledge source.

The bidders also share a common format: they start with a **wants** or **wants_removed** clause, followed by **make_bid**, as described in Section 8.2.5, then end with an **execute** clause which gives instructions for the execution of a bid created by **make_bid**. There is no reason why the first two components could not be duplicated to correspond to alternative sets of pre-conditions for bid creation, but this was not necessary for this application.

The sections of the single-object system, RECOGNISE1, described in Chapter 6 on which the Learner, Recogniser and Feedback knowledge sources were based were not independent of one another; the recogniser and feedback modules both used some predicates defined for the learner. To ensure independence of the knowledge sources, all shared predicates have been grouped together as 'knowledge source utilities'; the only predicates defined in a particular knowledge source are those which are specific to that knowledge source. The individual knowledge sources are described in Sections 8.3.2. to 8.3.8, and the utilities are described in Section 8.3.9.

8.3.2. Editor.

Editor's function is to edit the feature data for an image. It takes the image features from problembb1, removes duplicates and those with ratings below the appropriate rating threshold, numbers those remaining and enters them on problembb2. It also initialises the list of identified features on problembb2.

The **edit** program file is virtually the same as the edit file in RECOGNISE1, described in Section 6.4.1. The only changes made are the addition of a **new_entries** clause and those changes necessitated by the fact that the data used is now held on the blackboards.

8.3.3. Learner.

Learner is responsible for learning the recognition rules for an object. It adds the object name to the list of learnt object names on recognisebb1, develops match and distance_limits rules and places them

on recognisebb2, and develops set_probability rules which it places on recognisebb3. It then counts the number of occurrences of the object in the training set, placing this on recognisebb4 and also adding it to the total number of occurrences on recognisebb4. Learner uses, but does not alter, all the available edited feature data on problembb2, adds features which match object parts to the list of identified features on problembb2, and uses then removes the object part data on problembb4 .

The rule induction process, which is the central part of Learner's function, is the same as the rule induction stage of the single-object system, described in Section 6.4.2. The program files used - **learn**, **pattern_match** and **limits**, which are contained in Learner itself, and **feature_match**, **make_sets**, **find**, **check_match** and **check_limits**, which are contained in Utilities as they are shared with other knowledge sources - differ only slightly from those described in Section 6.4.2. However, rather than printing out the rules as was done in the single-object system, Learner follows the rule induction process by counting and removing object instances, using the new Utilities program file **remove_instances** (described in Section 8.3.9).

8.3.4. Recogniser.

The formation of feature sets which could correspond to instances of an object, placing these on problembb3, and the insertion of the object, set number and probability of each set into a probability list arranged in descending order of probability, also on problembb3, is the task carried out by the Recogniser knowledge source.

Recogniser has a slightly more complex bidder than most of the

other knowledge sources, as it may be required to make not just one bid at a time, but several - one for each of the objects which could be sought in an image. The bids are assigned ratings which depend on the number of occurrences of the object, so that the objects which have occurred most frequently in the past, and so can be considered the most likely to be found in a new image, are sought first.

The program files used, **search** and **add** in Recogniser, and **make_sets**, **find**, **check_match** and **check_limits** in Utilities, have been adapted from those used in the recognition stage of the single-object system, described in Section 6.4.3, the functions of which have been divided between Recogniser, Acceptor and Selector. The main alteration to **search** is the inclusion of a check to ensure that image contains at least two unidentified features which could correspond to parts of the object being sought, the search being aborted if it does not. More extensive changes have been made to **add**: a single probability list is used for all the searches conducted on an image, so entries in the list now have to specify the object sought as well as the feature set number and probability; feature sets are not entered in the list unless their probability is equal to or above the referral threshold; and when a new set is added to the list, the new predicate **remove_subsets** is called to find and remove any subsets of it already in the list. This last change, which will ensure that only maximal feature sets are considered as potential object instances, should prevent some of the problems which arose during the testing of the single-object system (see Section 7.3.2).

8.3.5. Acceptor.

Feature sets in the probability list on problembb3 whose

probabilities exceed the acceptance threshold set by the user are handled by Acceptor. Acceptance of such sets as object instances is not automatic; Acceptor checks to ensure that they do not contain any features which have already been identified, then carries out object-specific background checks before deciding whether they should be accepted or discarded. When a set is accepted, the user is notified, the list of identified features on problembb2 is updated and the relevant object parts are entered on problembb4.

Acceptor contains just one program file, **accept**, but also uses the **identify** and **write_set** files in Utilities, which it shares with Selector. All these files are derived from the RECOGNISE1 file **report**, described in Section 6.4.3. The operation of **accept** is straightforward; it operates on each element of the probability list whose probability exceeds the acceptance threshold in turn, carrying out the checks described above then, if these succeed, calling **write_set** to write out the feature set, writing "Accepted" beneath it, and calling **identify** to record the newly identified object instance.

The background checks, which are called by:

background_check(Object,PicNo,Parts,Features)

must be programmed by the user, and entered into the system at the same time as the object part lists and rating thresholds. They can be used to check more complex relationships between potential object parts than the simple distance limits which are checked when feature sets are constructed. They are not optional; if no checks are required for a particular object, then an empty check must be specified:

background_check(Object,_,_,_) :- !.

8.3.6. Selector.

Selector's function corresponds to that of the **report** predicate in the recognition stage of the single-object system (Section 6.4.3). It removes the probability list for an image from problembb3, checks each element in turn, refers those which pass the checks to the user for acceptance or rejection, then records the parts of accepted object instances on problembb4.

The program file **select** is very similar to **accept**, having been derived from the same source; it only differs in that it does not compare probabilities with the acceptance threshold, and refers sets which satisfy the checks to the user for acceptance or rejection rather than accepting them automatically.

8.3.7. Feedback.

Feedback is responsible for updating the recognition blackboard in accordance with the results of the image identification process. It removes object part data from problembb4 and uses this, in conjunction with feature set data on problembb3 (which it alters, but does not remove), to update the set-probability rules on recognisebb3 and the occurrences on recognisebb4.

Like Recogniser, Feedback has a bidder which can make several bids at a time, one for each object which has been sought in the image; these bids, however, are all given the same rating.

The program file **feedback** is the same as the file used in RECOGNISE1 (Section 6.4.4), but for the fact that the old instruction to list the updated set-probability rules has been replaced by a call to **remove_instances** (in Utilities). The **feature_match** and **check_match** files in Utilities are also used.

8.3.8. Remover.

This very simple knowledge source is a garbage collector: it clears out the features, list of identified features (problembb2) and feature sets (problembb3) for a test image when they are no longer required. The single program file, **remove**, contains just the appropriate **retract/retractall** commands.

8.3.9. Knowledge Source Utilities.

The files contained in Utilities are as follows (those described as being 'from RECOGNISE1' are taken from the single-object system, with slight modifications to allow for changes in data format):

general_utilities: updated version of RECOGNISE1 **utilities**, used by various knowledge sources and system processes.

make_sets, find, check_limits: from RECOGNISE1, used by Learner and Recogniser.

feature_match, check_match: from RECOGNISE1, used by Learner and Feedback.

identify: adapted from part of RECOGNISE1 **report**, used by Acceptor and Selector to record a feature set which has been accepted as an object instance i.e. to add the features to the list of identified features on problembb2, number the instance and record the object parts on problembb4.

write_set: adapted from part of RECOGNISE1 **report,** used by Acceptor and Selector to write out feature sets/part sets.

remove_instances: new file, used by Learner and Feedback to remove object part data from problembb4 and update object occurrences/total occurrences on recognisebb4.

8.4. RUNNING THE SYSTEM.

This system, RECOGNISE2, was written in Cprolog like its predecessor, RECOGNISE1. It is very simple to use, as most operations are carried out automatically by the blackboard controller. It can be run by first entering the Prolog interpreter, then when the prompt appears, entering:

[recognise2/master].

This master file contains instructions to load the system program files, knowledge sources and utilities. When these have been loaded, the system has to be initialised by entering:

init.

The initialisation process prompts the user to enter the duplicates limit and acceptance and referral thresholds, and to select manual or automatic bid execution. (See Section 8.2.6). When the process is

complete, the system responds:

What now?

(Enter "help." to view options)

The options available are described in Section 8.2.6. There is no need for the user to supply specific instructions for the system to learn recognition rules for an object, or to search a test image; all that is necessary is to specify the data files to be read in, then the appropriate operations will be carried out automatically.

The system informs the user of the operations which are being carried out and the results obtained, with the exception of recognition rules, which are displayed only on request. If automatic bid execution has been selected, the user is only consulted when (s)he is required to accept or reject a feature set with a probability between the referral and acceptance thresholds, and when the bid list is empty, i.e. when no further operations can be performed until more data has been supplied. With manual bid execution the user is also consulted after each operation, but has only to enter

run.

to instruct the next operation to be performed.

A full listing of RECOGNISE2 is provided in Appendix A; Appendix B contains sample runs.

CHAPTER 9

TESTING THE IMAGE IDENTIFIER

9.1. INTRODUCTION.

The image identifier, RECOGNISE2, could not be tested using data from the feature matcher because suitable data could not be obtained from R.S.R.E., so the tests, like the RECOGNISE1 tests described in Chapter 7, had to be conducted using synthetic data and 'semi-real' data.

It was not felt necessary to provide a further demonstration of the capabilities which RECOGNISE2 has in common with RECOGNISE1, so the synthetic data was specially devised to examine how well the system succeeded in providing the additional facilities outlined in the specification in Section 7.2.2. The tests were particularly intended to illustrate its ability to distinguish between a considerable number of different objects, including objects which differed from one another only in size or only in orientation, and to show how it could cope with the introduction of a scond training set containing data on new objects after the objects in the initial training set had been learnt.

The 'semi-real' data consisted of feature data extracted manually from a set of photographs of traffic - cars, vans and lorries - on a moderately busy road in Loughborough. The tests conducted on this data were intended to give some idea of the kind of results which could have been obtained if the data which had been expected from R.S.R.E. had materialised.

9.2. TESTS USING SYNTHETIC DATA.

The objects selected for identification in these tests were simple geometric figures: squares, triangles and rectangles. The system was to be presented with test images containing a number of different figures, including some which overlapped or partially occluded one another, and was to identify the shape, size and orientation of each of them.

Eight different objects were chosen: squares of two size ranges (big and little), right-angled isosceles triangles with four different orientations, and two rectangles, one taller than it was broad, the other broader than it was tall. Two sets of training examples were to be provided, one for the squares and triangles and one for the rectangles, and there were to be two sets of four test images, the first set containing just the squares and triangles and the second set containing all eight objects.

9.2.1. Data Preparation.

The first six objects to be learnt are shown in Figure 9.1. The object parts selected for feature-matching were the vertices of the shapes, so these have been numbered in the figure; they are named top-left, top-right, bottom-right, bottom-left and right-angle, as appropriate. The vertices can all be matched by twelve feature patterns, four right angles and eight 45^{o} angles, which are shown in Figure 9.2. (The right angles will, of course, also match the vertices of the rectangles, so no further feature patterns will be required when the set of objects is expanded).

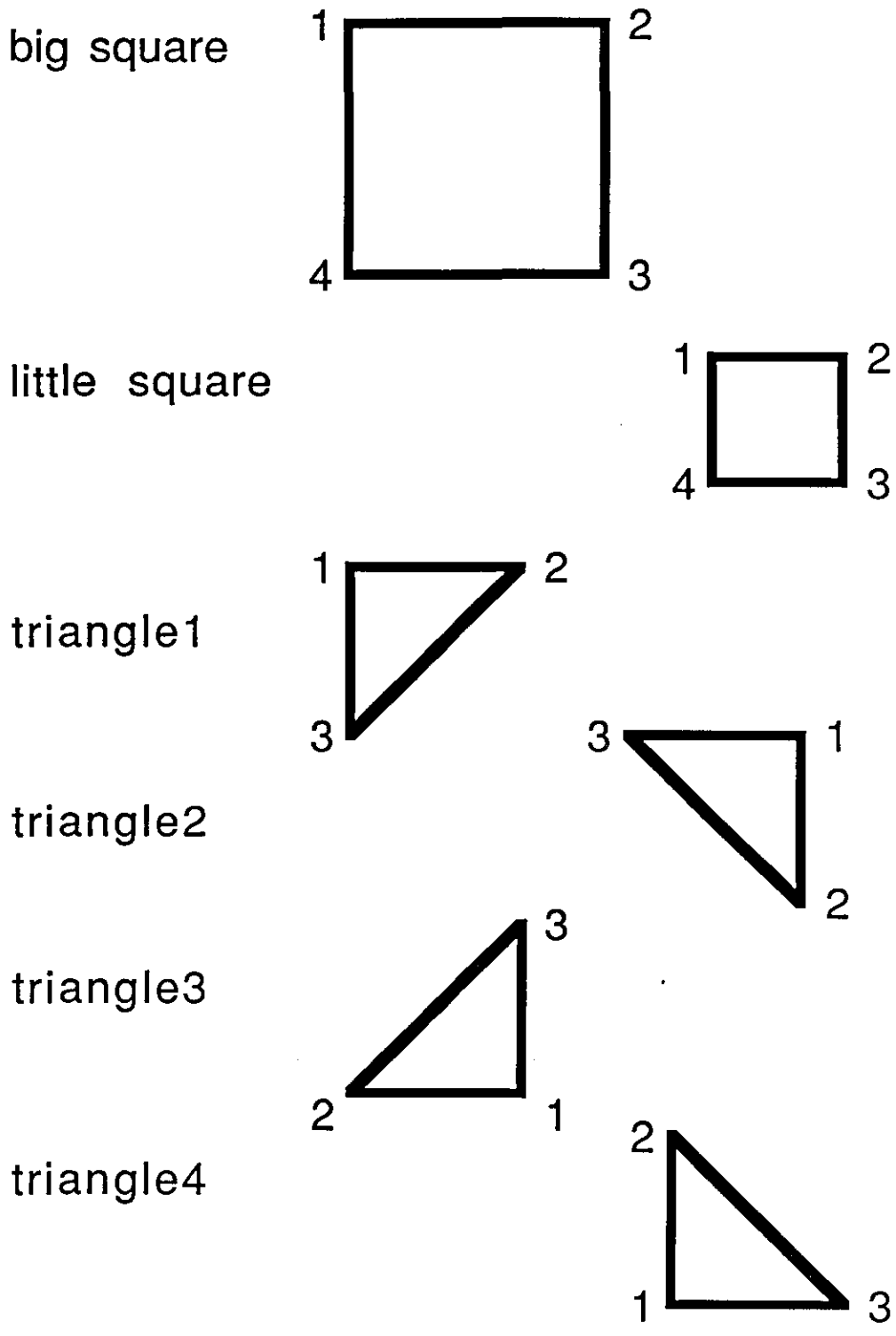


Figure 9.1. The first six shapes, with vertices numbered.

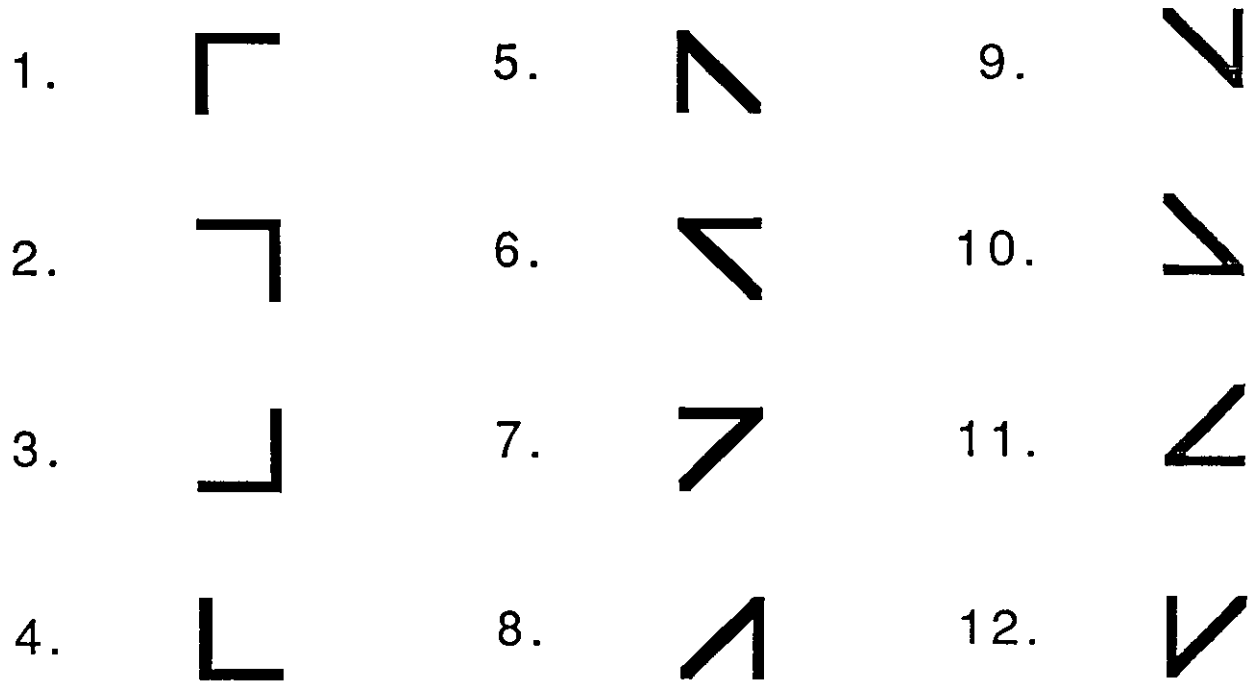


Figure 9.2. Feature patterns for shape recognition.

The feature matcher specifies the mean x and y co-ordinates of each of the matches it finds, but it was decided that to simplify the manual matching used as a substitute for the feature matcher, the x and y co-ordinates of the vertex points should be substituted. However, this means that where more than two lines meet at a point, all the vertices formed will have the same co-ordinates - for example, if two lines cross one another at right angles, four right angle vertices all with the co-ordinates of the crossing point will be obtained. As these vertices are not merely duplicates of one another and none of them may be discarded, the system's duplicates limit must be set at 0.

If several different vertices occurred at the same co-ordinates as an object part in the training images, the system could, as a result, come

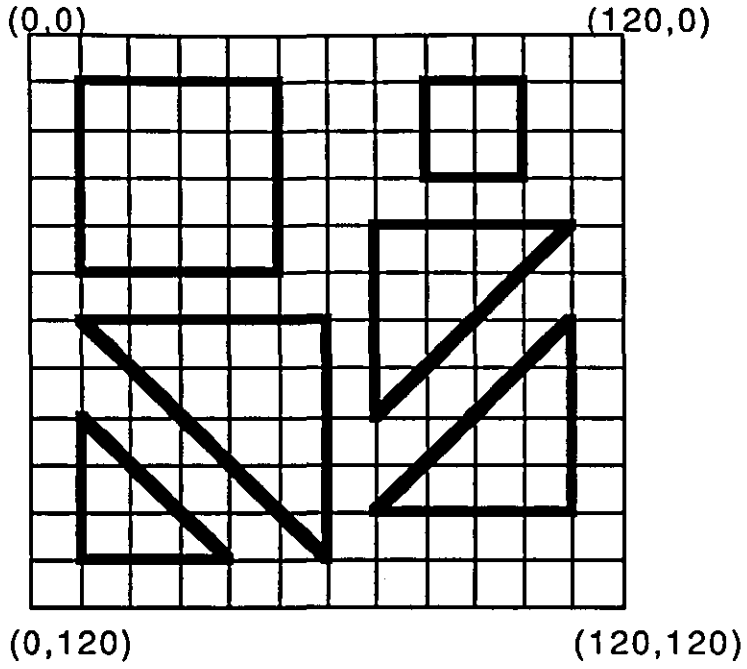
up with a large number of spurious object part/feature pattern matches. To ensure that only the correct matches would be found, it was therefore decided that the training examples used should show disjoint object instances. The maximum and minimum dimensions of each object could be specified by just two instances, so only two training examples, each containing one instance of each of the six objects, were required (see Figure 9.3).

The first set of test images is shown in Figure 9.4. The images do not increase significantly in complexity because it was hoped that after the feature sets found in the first two tests (Shapes3 and Shapes4) had been accepted or rejected manually, with the acceptance limit at 110, the acceptance limit could be lowered to demonstrate the use of Acceptor to automatically accept high-probability sets in Shapes5 and Shapes6.

The rectangles to be learnt from the second training set are shown in Figure 9.5, with their vertices numbered in the same way as the earlier shapes. These, too, can be learnt from just two training instances each; the single image which comprises the second training set is shown in Figure 9.6. Figure 9.7. shows the four images which make up the second set of tests - again, it is intended that acceptance or rejection of feature sets should be manual for the first two (Shapes8 and Shapes9), and automatic for the second two (Shapes10 and Shapes11).

9.2.2. Background Checks.

The recognition rules induced by the Learner will ensure that the distances between pairs of features in candidate feature sets fall within the appropriate limits, but this is not sufficient to guarantee that



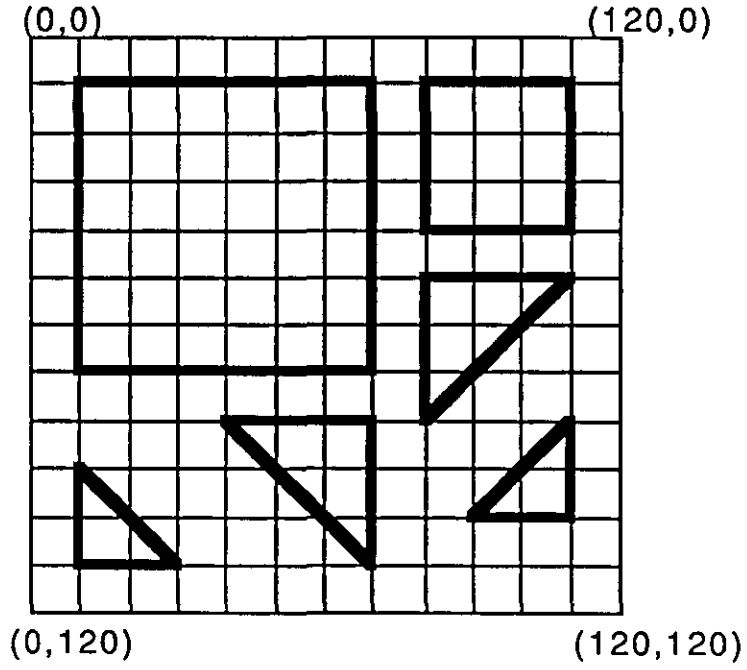
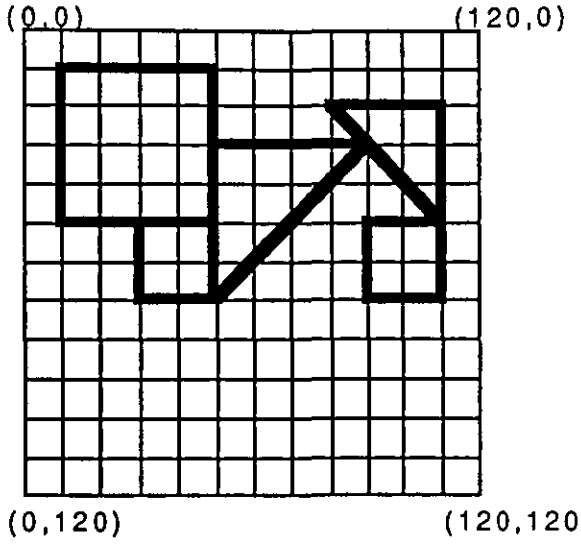
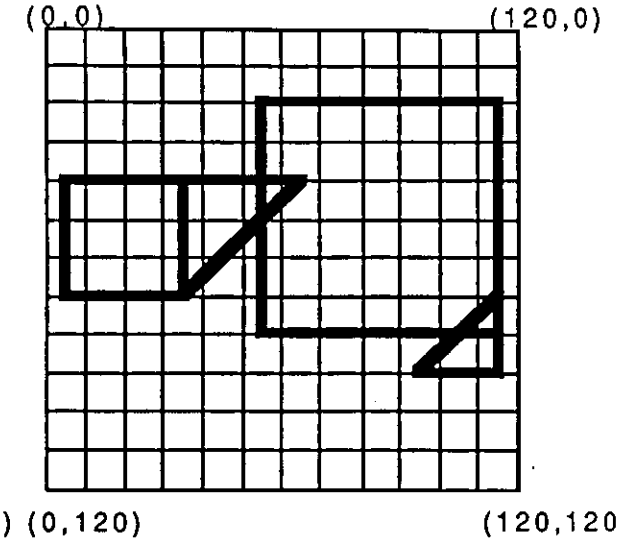


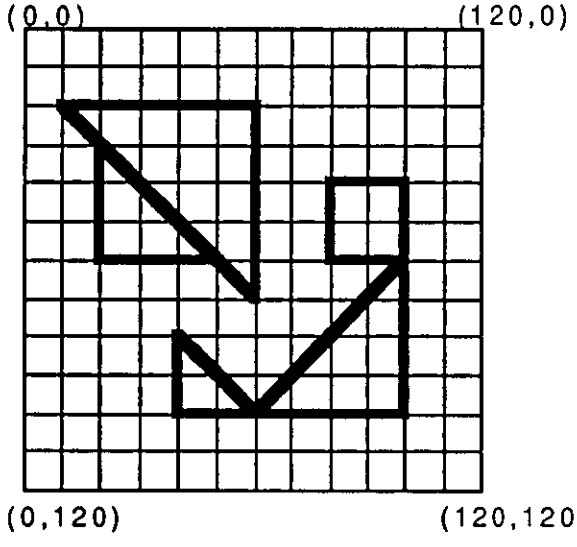
Figure 9.3. The first set of training images for shapes.



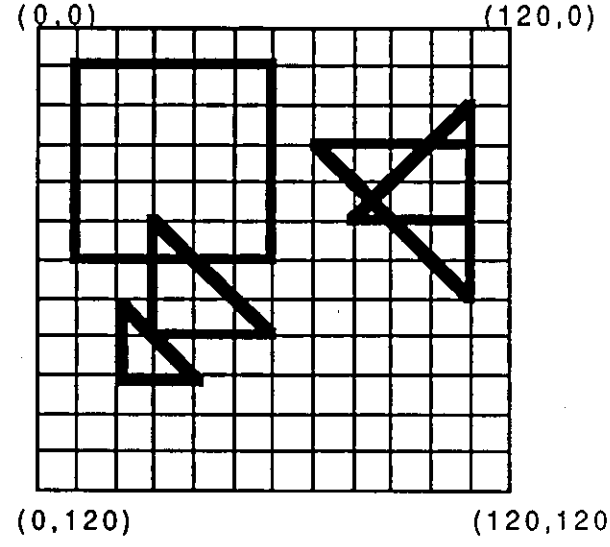
Shapes3



Shapes4



Shapes5



Shapes6

Figure 9.4. The first set of test images for shapes.

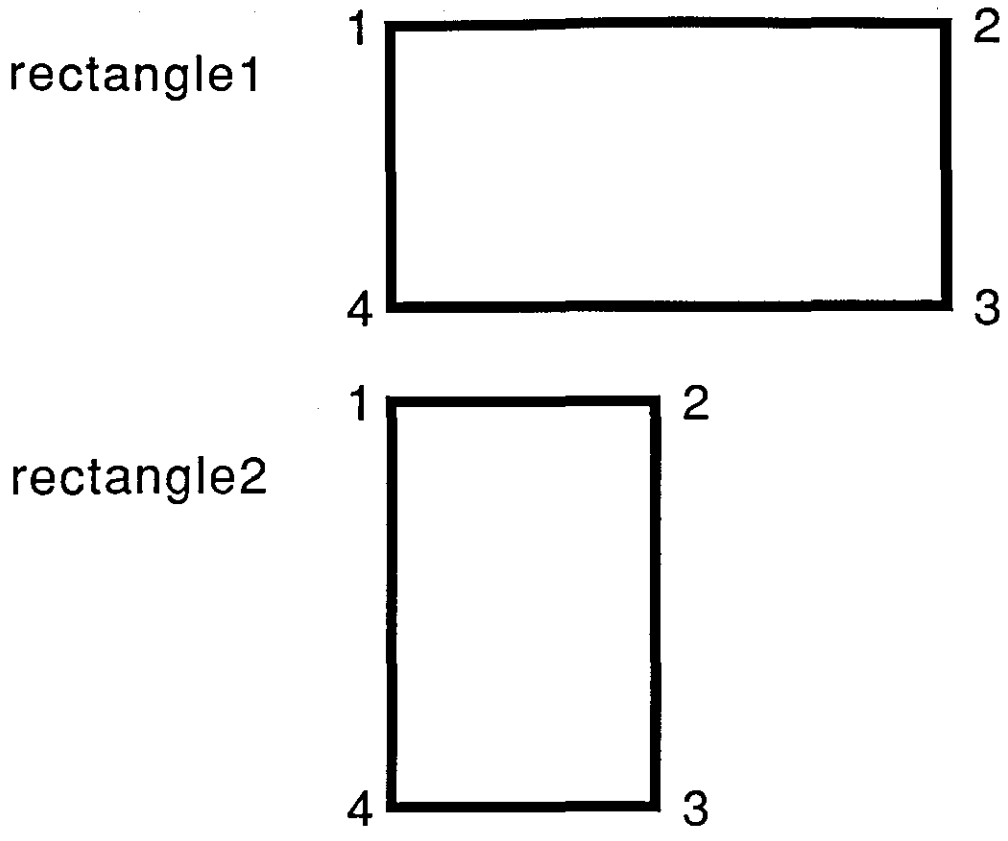


Figure 9.5. The rectangles, with vertices numbered.

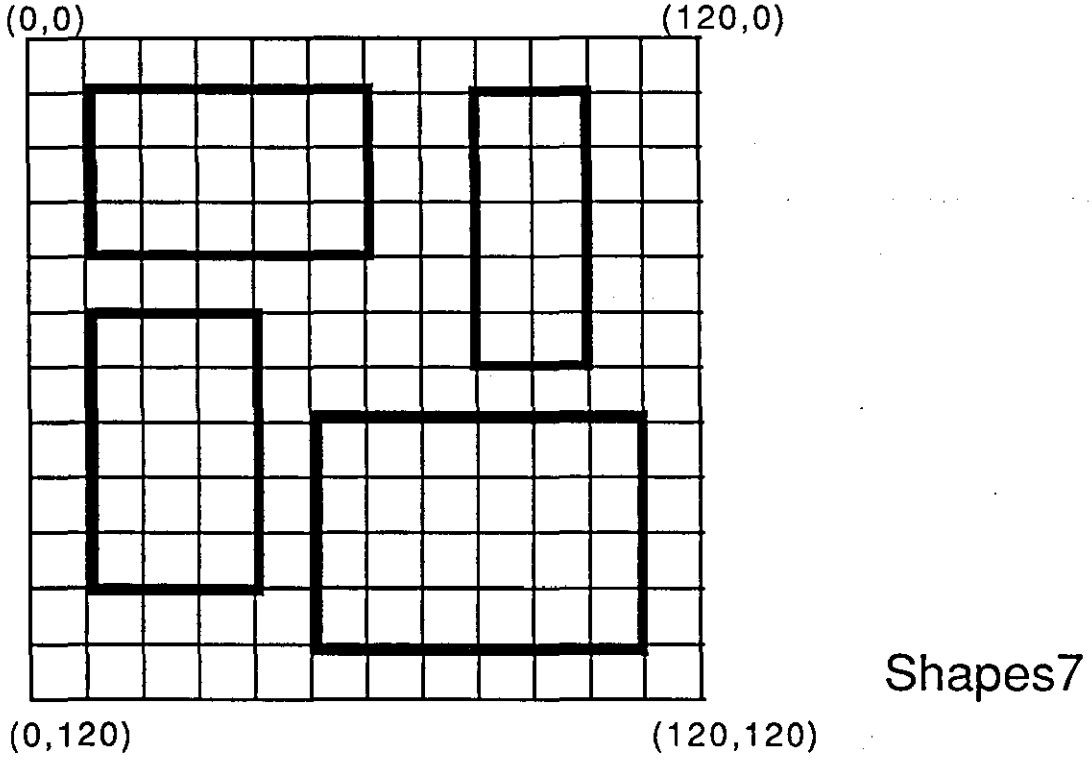
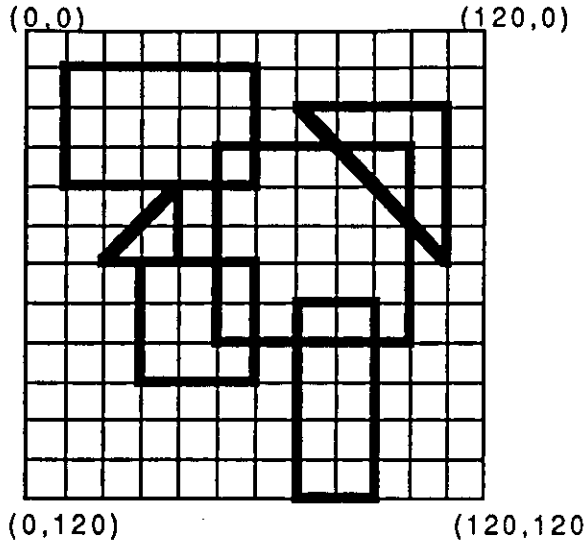
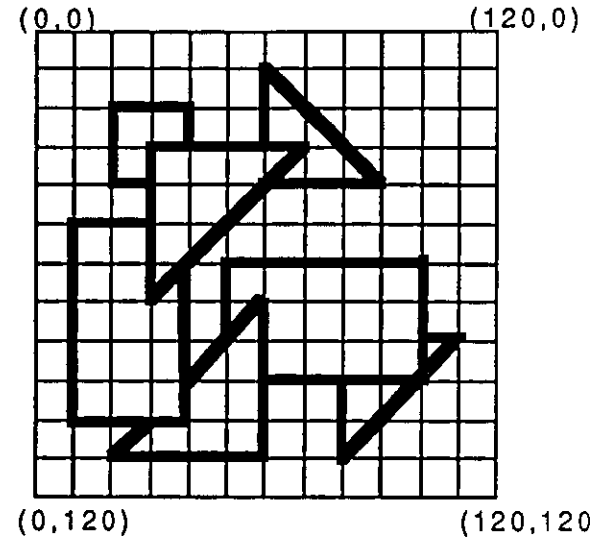


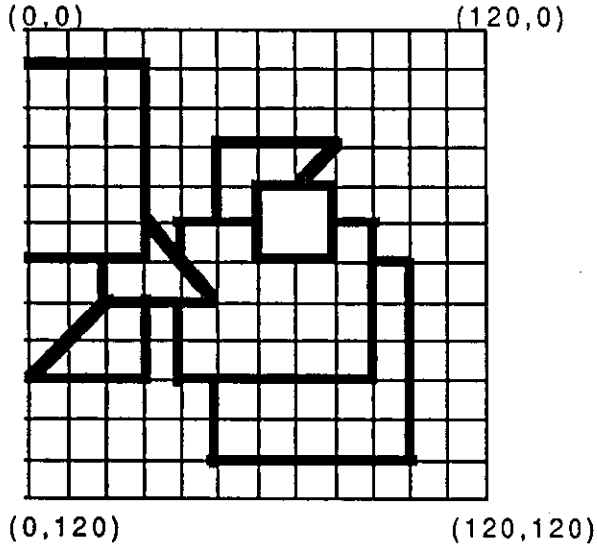
Figure 9.6. The training image for rectangles.



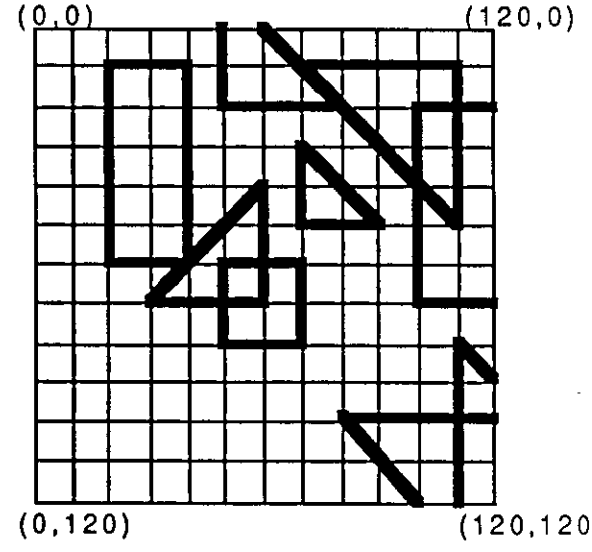
Shapes8



Shapes9



Shapes10



Shapes11

Figure 9.7. The second set of test images for shapes.

candidate squares will be square, and candidate triangles will be isosceles. For example, if the sides of a big square could be between 40 and 60 units long, a rectangle with width 60 units and height 40 units would satisfy the induced rules. However, the background checks facility can be used to eliminate candidates which are not of the right shape.

The background checks are specified in the **shapescheck** program file, which is to be loaded into the system together with the object part lists and rating thresholds (contained in **shapesgen, rectanglesgen**). No checks are necessary on rectangles as any rectangle which satisfies the distance limit rules will be acceptable, so empty checks must be specified for these:

```
background_check(rectangle1,_,_,_) :- !.
background_check(rectangle2,_,_,_) :- !.
```

For squares and triangles, the checks must allow for the fact that candidate feature sets may not be complete. If a feature set contains just two adjacent vertices of a square, or the right angle and one other vertex of a triangle, no checks will be necessary; however, checks must be made on two-element sets containing the opposite vertices of a possible square or the two 45^{o} vertices of a possible triangle, and on all sets of three or more features. The two-element checks specify the object name and part set, then pass the picture number and feature numbers to **opposite_corners**, which succeeds if the absolute values of the differences between the x co-ordinates and the y co-ordinates of the features are equal:

```
background_check(Object,PicNo,PartSet,[F1,F2]) :-
        !,
        opposite_corners(PicNo,[F1,F2]).
```

The checks on larger feature sets pass the picture number and the first three feature numbers to **equal_sides**, which succeeds if any two of the three features are the same distance apart:

```
background_check(_,PicNo,[F1,F2:Rest]) :-
        member(F3,Rest),
        !,
        equal_sides(PicNo,[F1,F2,F3]).
```

The calls to **opposite_corners** and **equal_sides** are preceded by cuts (!) to ensure that the call to **background_check** will fail if they fail; this allows the **shapescheck** file to be concluded with an empty check to cover all object/part set combinations which have not already been specified.

9.2.3. Tests conducted and results obtained.

Two system runs were carried out using the shapes data. The first run was a straightforward test of the operation of the system; the second run was intended to show how the recognition rules induced for rectangles would be affected by the stage at which the rectangles training data was introduced (before or after the first set of tests). Edited listings of both runs are provided in Appendix B.

For the first run, the system was initialised with a duplicates limit of 0, acceptance threshold of 110 (no automatic acceptance), referral threshold of 0 (referral to user of all candidate feature sets) and automatic bid execution. The training data on the first six shapes, **shapes1** and **shapes2**, was loaded together with the **shapesgen** and **shapescheck** files; the system automatically proceeded to edit the

training data and to learn rules for the shapes.

When first test file, **shapes3**, had been loaded the system searched for all the shapes except triangle3 (which could not be present as the data did not include any features to match its 45^{o} angles) in the order in which the shapes had been learnt - with exactly two occurrences of each shape in the training images, the numbers of occurrences could not yet have any effect on the search order. The feature sets found in the image were all assigned probabilities of 100%, as would be expected; the five shapes were all correctly identified and were accepted by the user, and two (correct) partial feature sets which were also found were rejected. When the recognition process was complete, the rules for four of the shapes sought were updated by feedback (no feature sets had been found for the other shape, triangle4), and the test data was removed from the blackboard.

For the second test, **shapes4**, the search order was affected by the numbers of occurrences of each shape, the shape which had appeared twice in **shapes3**, little_square, being sought first. The procedure the system followed and the results it obtained were again satisfactory.

After the second test, **show_rules** was used to request a printout of the recognition rules for each object. After so few examples, the probability rules necessarily showed a low level of accuracy, probabilities of 100% being given for many partial sets; far more precise probabilities would be required for the automatic acceptance procedure to work perfectly, but to check its operation the acceptance threshold was nonetheless lowered to 100 before the next test.

The object instances in **shapes5** were such that, despite the inaccuracy of the rules, the results obtained by the automatic acceptance procedure were perfect. The small square in the image was found first, and accepted. The system then searched for big squares, but none of the partial big squares it found had high enough probabilities for immediate acceptance, so it deferred judgement on these and went on to search for triangles. When the four triangles had all been found and accepted, the partial big squares could all be rejected as they contained features which had already been identified, so there was no need to refer any partial sets to the user for decisions.

The results obtained from **shapes6**, in contrast, clearly reveal the dangers inherent in using automatic acceptance prematurely. The system again began by searching for little squares, but the probabilities of two of the partial little squares found had not yet been established accurately and were still set at 100%, so these were erroneously accepted; the false identification of their features meant that the big square in the image was automatically rejected. (The triangles in **shapes6** were, however, identified correctly).

The acceptance threshold was altered back to 110 before **rectanglesgen** and the rectangles training image, **shapes7**, were loaded. The recognition rules for rectangles were printed out as soon as they had been learnt, for later comparison with the rules learnt in the second system run; the fact that the partial set probabilities were not all 100% showed that the first set of training images had been correctly included in the rectangles training set.

The results obtained from **shapes8** and **shapes9**, which had both

been designed to contain large numbers of partial feature sets, to try to ensure that the probability tables would be as accurate as possible (given the limited number of images used) for the final two tests, were as expected. The incorporation of the rectangles into the set of learnt objects had clearly been successful. The feedback of faulty information from **shapes6** meant that the probability assigned to the complete big square in **shapes8** was too low, but with referral to user rather than automatic acceptance for these tests, this did not matter much.

The acceptance threshold was again reduced to 100 before loading **shapes10**, a fairly difficult test image containing partially occluded object instances. In this test, four objects were accepted automatically, three of them correctly and one (a partial triangle for which inadequate rules had been developed) incorrectly; a further eight feature sets, including sets corresponding to the remaining three objects in the image, were referred to the user. The final test, **shapes11**, contained a mixture of overlapping objects and partial objects, six of which were correctly accepted automatically, the remainder being correctly referred.

The probability tables were printed out again at the end of the run. Most of the tables had improved in accuracy since the earlier printout, but the table for big squares showed a marked deterioration. This was because the feature sets corresponding to some of the instances of rectangles had satisfied the recognition rules for the big square. These sets had not been accepted (or offered for acceptance) as big squares because they failed to satisfy the background checks, but they had still been used for feedback purposes, to update the big square probability table.

The second system run was initialised with the same thresholds as the first, but with manual bid execution to allow all the training data to be loaded at the start of the run. When the data had been loaded the help facility was invoked, and the command to switch to automatic execution was entered to allow the rules to be learnt automatically. The rules for both types of rectangle were printed out, and could be seen to be identical to those obtained in the first run. The first set of tests were then repeated, this time with the acceptance threshold held at 110 for all four of them; the results were basically the same as those obtained in the first run, except that some possible partial rectangles were found (and rejected), and the inclusion of **shapes7** in the training set for the little square had improved the accuracy of its probability table sufficiently for only one partial little square to have to be rejected before the system correctly identified the big square in **shapes6**. The second set of tests were not repeated, as it was felt that nothing further could be learnt from these.

9.2.4. Conclusions.

These tests were, on the whole, very satisfactory; the system appeared to meet the specification given in Chapter 8 in every respect. The image identification results obtained were not perfect, but this was more because insufficient data was used to allow accurate probabilities to be established than because the system did not work as it was intended to.

The recognition task selected for these tests was one which required the use of background checks, as well as the induced recognition rules, to distinguish between some of the objects occurring in the

images. The incorporation of background checks into the system significantly increases its flexibility and the range of problems to which it could be applied, but the facts that these checks were not taken into account when determining probabilities, and that without these checks some of the rectangles could be mistaken for squares, meant that low probabilities were established for big squares in the first test run. It would be possible to alter the system to enable checks to be carried out when feature sets are first created, which would improve the accuracy of the probabilities induced in this situation, but this modification does not seem desirable. It would increase the running time, as it would obviously take longer to check every feature set than to check just those maximal sets which represent possible object instances; more importantly, it would affect the overall structure of the system. The learning and recognition stages would be less self-contained if pre-programmed background knowledge were to be introduced earlier in the identification process; the acceptance/rejection stage appears to be a more appropriate place for this type of knowledge to be employed. In view of the fact that all the probabilities for an object are affected when such problems arise, a better approach might be to set separate acceptance/referral thresholds for each object, using lower thresholds for objects which suffer from interference from other similar objects.

9.3. TESTS USING PHOTOGRAPHS OF TRAFFIC.

For these tests, photographs were taken of traffic travelling along the main road outside Loughborough University. The photographs were all taken from approximately the same location, at a sufficient distance from the side of the road for the largest vehicles to just fit within the

frame. The background varied slightly, depending on the angle at which the camera was held; it consisted of some small trees, a hedge and a row of houses, some of which had satellite dishes on their walls which could be mistaken by a feature matcher for car wheels.

Thirty-two photographs were selected for the tests. They showed four different types of vehicle: cars, vans, small lorries and big lorries, all facing to the right. (Other vehicles, e.g. motorbikes, buses, occurred too infrequently to be included). The sixteen photographs which appeared, from a brief visual inspection, to contain the most extreme examples of each vehicle type were chosen to form the training set. They each showed just one vehicle; there were five instances of cars, five vans, four small lorries and two big lorries. The remaining sixteen photographs, which were to be used as recognition tests, showed nine cars, three vans, four small lorries and three big lorries - a total of nineteen vehicles as three of these photographs showed two vehicles each.

9.3.1. Data Preparation.

Unfortunately R.S.R.E.'s pre-processor, which was applied to the car photographs used for the tests described in Section 7.3., was unavailable for use with these traffic photographs; the feature patterns had to be selected, and the matches found, directly from the photographs themselves rather than from edged and filtered images. Figures 9.8 and 9.9 show typical training pictures; it can be seen from these that the task of measuring vehicle parts was not an easy one, and the results obtained can be expected to correspond only roughly to those which would have been obtained if the pre-processor and feature matcher had been

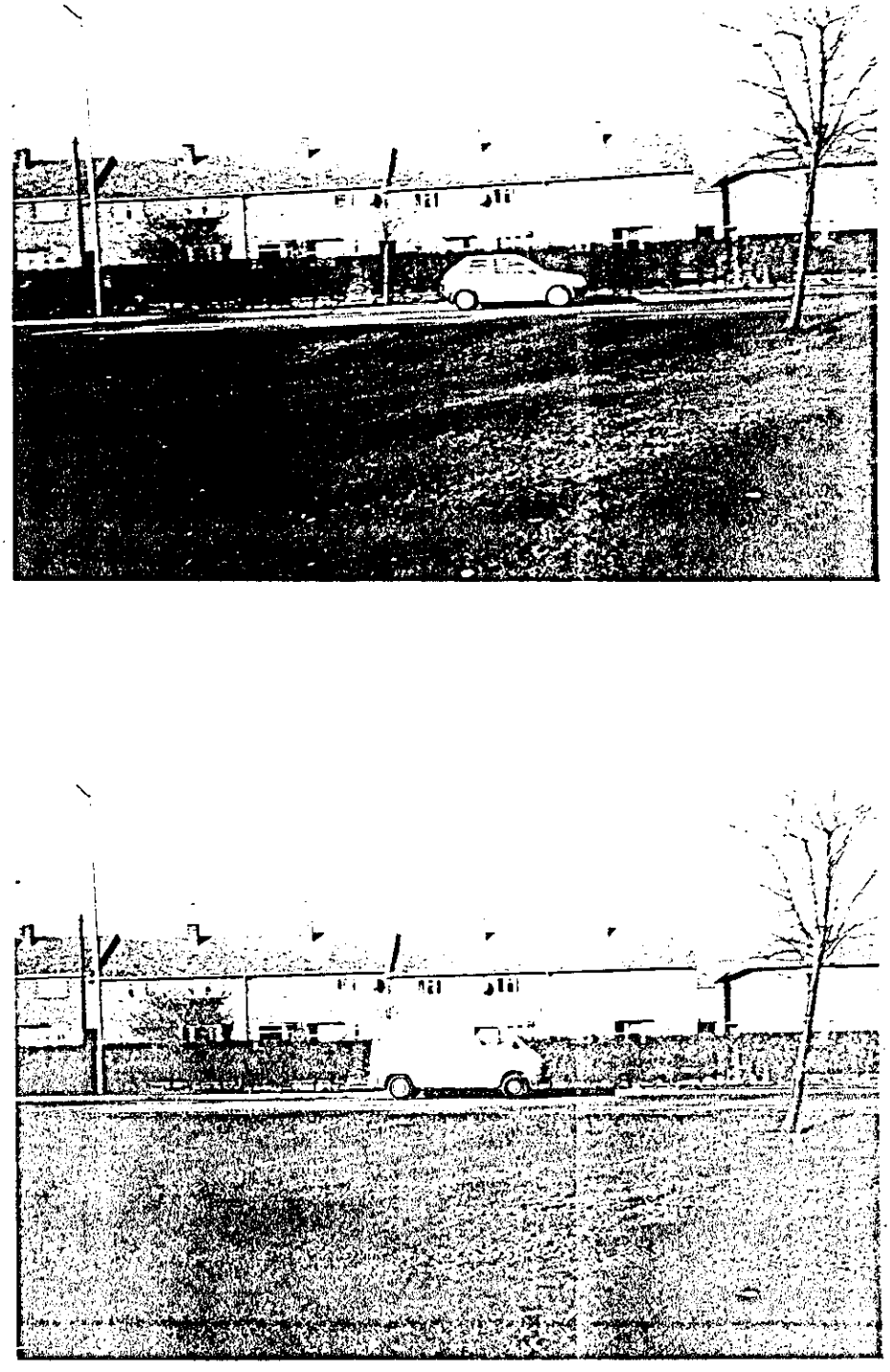


Figure 9.8. Typical traffic photographs - car and van.

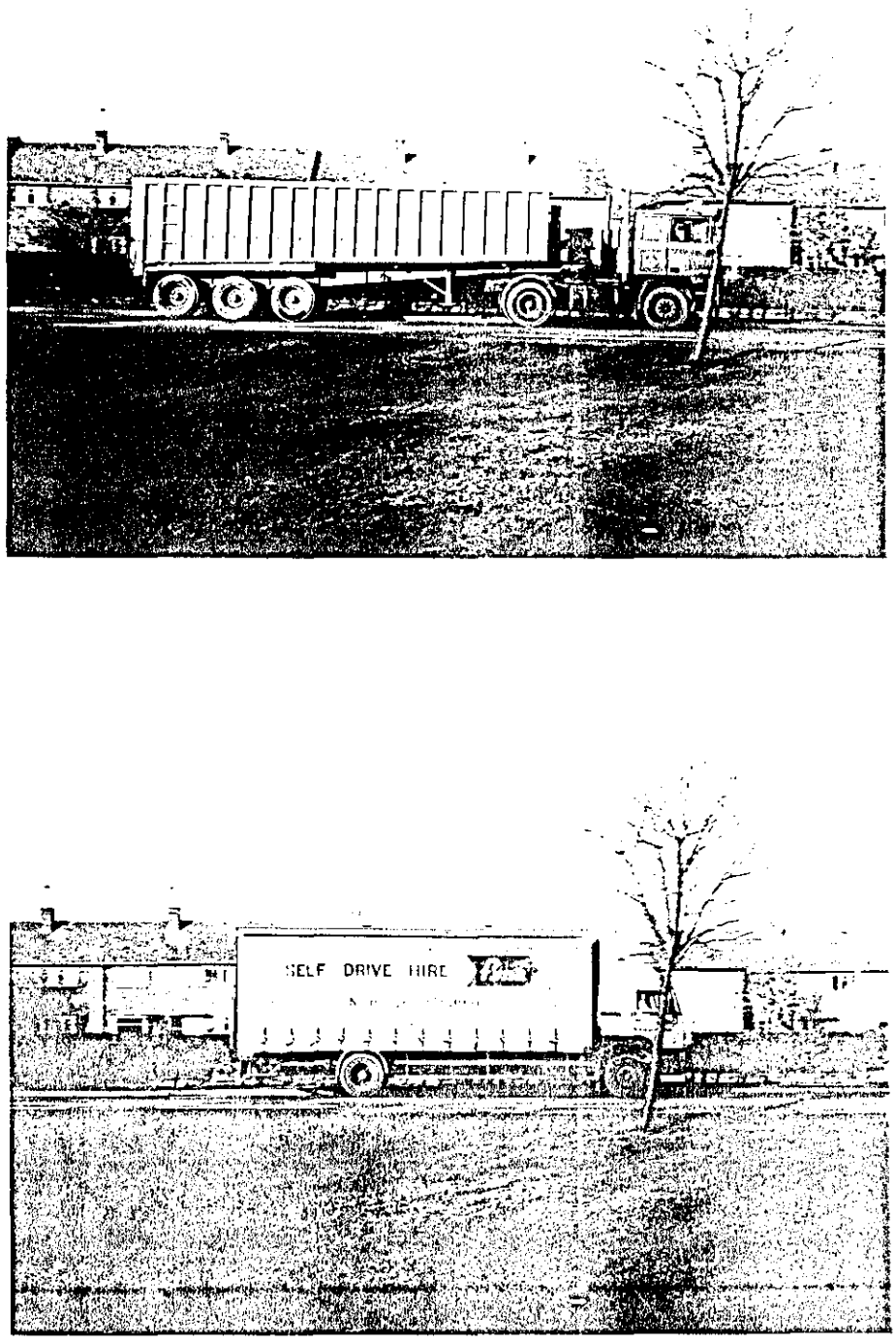


Figure 9.9. More traffic photographs - big and small lorries.

available.

The procedure followed was approximately the same as that described in Section 7.3.1., and so is not described in detail here. As the photographs were taken at a sufficient range for big lorries to fit into the frame, the cars appeared at a much smaller scale than those in the earlier tests; the variations in window shape became insignificant at this scale, and the co-ordinate differences between the wheels and the wheel arches could no longer be determined reliably, so a different set of car parts had to be selected for matching. The car parts used this time were:

1 front_wheel

2 rear_wheel

3 top

4 front_window.

The tops of the vans were not of a distinctive enough shape to be matched reliably, so only three van parts were used:

1 front_wheel

2 rear_wheel

3 front_window.

The part set for small lorries, the back parts of which varied considerably, was:

1 front_wheel

2 rear_wheel

3 cab_top

4 cab_window

and for the large lorries with five pairs of wheels, the part set was:

1 wheel1 (front wheel)

2 wheel2

3 wheel3

4 wheel4

5 wheel5 (rear wheel)

6 cab_top.

The feature patterns selected are shown in Figure 9.10. (All measurements in millimetres).

It was assumed, when identifying features in the photographs, that satellite dishes would be recognised as small wheels (pattern 1), complete house windows would be recognised as van/lorry windows (pattern 6), and house windows which were partially occluded by the hedge would be recognised as car windows (pattern 5), so a large number of features could be expected to be found in each photograph. The features were given ratings of up to 100; the thresholds were all set at 50, and features whose ratings would be below the thresholds were not included in the data, as they would in any case be eliminated by the editing procedure, which left between ten and twenty features per photograph. As the object parts selected for matching, though difficult to measure accurately, were easy to locate in the photographs, the object part data provided for the training pictures was complete.

9.3.2. Background checks.

No background checks were carried out; as this was the main application for which the system was designed, it was considered that it should be possible to obtain adequate results without them. A program file specifying 'empty' checks was therefore drawn up.

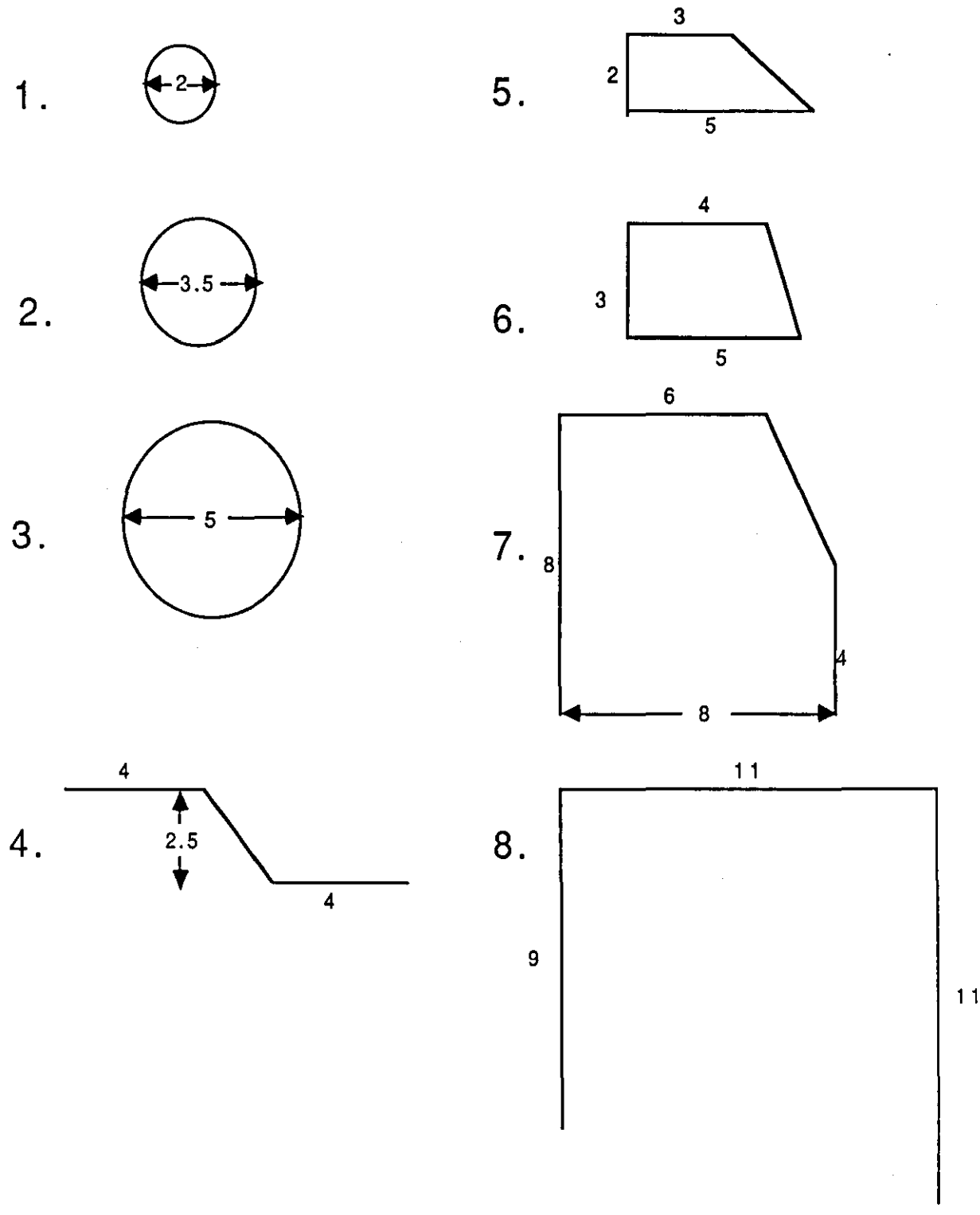


Figure 9.10. Feature patterns for traffic identification.

9.3.3. Tests conducted and results obtained.

For the tests conducted using the traffic data the duplicates limit was set at 3 and the acceptance and referral thresholds at 110 and 0 respectively. The training data was all loaded at the start of the system run, then when the objects had been learnt the test files were loaded one at a time. The test photographs were shuffled before being numbered, so the order of the tests was random.

In the first run, the system failed to find all the appropriate part/feature pattern matches - in particular, some of the window/window pattern matches were missed. This was because a given proportion of features of a particular pattern must match instances of a part for a match between the part and the pattern to be recognised; there were so many windows in the photographs that the proportions of matches were too low. The **pattern_match** file was edited to reduce the proportion required from one in ten to one in twenty, and the system was re-run.

When the correct match tables had been established the recognition results obtained were very impressive, seventeen of the nineteen vehicles being identified correctly. The two which were missed were a large lorry, with just its cab and the front two wheels visible, and a van, the wheel hubs of which were unusually faint.

As the training sets were small some of the distance limits learnt by the system were inevitably too narrow, so not all the parts of each vehicles were found - five of the vehicles were identified by just a pair of parts, the average being just under three parts per vehicle. However,

so long as sufficient parts were found for the system to make a positive identification the failure to find the remaining parts cannot be regarded as a problem.

There were very few incorrect identifications. Just twelve candidate sets had to be rejected, including ten pairs of satellite dishes which were incorrectly identified as pairs of wheels; as a result of their rejection the probability attached to a pair of car wheels declined steadily from 50% after training to under 35% at the end of the run, and the probability attached to a pair of van wheels similarly declined, showing that such recurring errors could eventually be eliminated by raising the referral threshold. One other rejected set was the top and rear wheel of a car pictured at an angle, a problem which was discussed in Section 7.3.2, and the final one was the front and rear wheels of a small lorry in the first test picture, which were identified first as wheels 2 and 3 of a large lorry. The system also found the cab top of the small lorry; had this picture occurred later in the run, the correct three part set would have been assigned a higher probability than the incorrect two part set, so the error would not have occurred.

A listing of the test is included in Appendix B.

9.3.4. Conclusions.

The system produced excellent results on this test after the **pattern_match** file had been altered to allow the correct part/pattern matches to be found, the only errors arising being ones which could be eliminated by using a more adequate training set and allowing the system to run for long enough for accurate probabilities to be established and

reliable acceptance and referral thresholds to be set.

The **pattern_match** table could, perhaps, be established more reliably by comparing the number of part/feature matches with the number of parts rather than the number of features; the system would then be less sensitive to the occurrence of large numbers of features of a particular pattern in the background of the images.

It must be borne in mind when assessing the results that the feature data used was obtained by manual matching, not by the feature matcher program. Where any doubt existed as to whether a background feature matched a given feature pattern, the match was included, so this data should have contained at least as much noise as 'real' data would have done, but the test still cannot be expected to give a totally accurate impression of how well the feature matcher/image identifier combination would work.

CHAPTER 10
DISCUSSION

10.1. SUMMARY OF WHAT HAS BEEN ACHIEVED.

This project was essentially practical in its objectives; the aim was to produce a working image identifier. Its practicality was somewhat marred by the fact that the development of the feature matcher program which was to provide the input data (Varga et al., '89; Series et al., '89) was discontinued before appropriate data for the system tests had been produced, so the testing had to be carried out using synthetic data and feature data extracted manually from processed images and photographs. The results obtained from the synthetic data were very satisfactory; this does not, of course, guarantee that the feature matcher/image identifier combination would have worked well, but the indications are that it would have done.

The keynote throughout the design and development of the image identifier was simplicity - in each design area, the simplest techniques which were consistent with satisfactory performance were adopted, and approximations were made whenever these would not compromise the quality of the output. The overall design of the system was simplified by splitting the design process into two stages: developing a recogniser for a single object, then incorporating this recogniser into a blackboard system to produce the image identifier.

When establishing recognition rules for objects, only the simplest relationships between object parts - the distance limits between pairs of parts - were considered; the rules based on these relationships proved

adequate, so there was no need to consider more complex relationships.

A major consideration in most applications of Artificial Intelligence is how to control searches and how best to prune search trees. When searching for feature sets which could represent object instances, it was decided that the simplest method would be to prune before and after the search, not during it. The features were pruned to remove duplicates and features with low ratings, then an exhaustive search was conducted for sets of the remaining features which satisfied the distance limits, and at the end these sets were pruned, where necessary, by carrying out background checks and eliminating sets with low probabilities. The decision not to carry out any pruning during the search meant that probabilities would not be required at this stage, so the process of identifying candidate feature sets could be separated from the process of attaching probabilities to them. This in turn meant that the introduction of feedback to update the probabilities was straightforward.

Chapter 5 contains a description of a large number of methods of handling uncertainty, most of which are conceptually or computationally quite complex, but here again the simplest method was adopted: probabilities were established empirically by counting feature sets. The scale of the problem was such that probabilities could be established from first principles for every combination of feature sets and object part sets of interest, so there was no need to consider how the probabilities were related to one another, to try to establish dependencies between pieces of evidence or to adopt rules for combining probabilities.

The fact that probabilities were established before conducting background checks on candidate feature sets meant that the probabilities

assigned to checked sets might not be accurate. However, the main function of the probability measures was to rank candidate sets; as the inaccuracies introduced by the background checks would not normally significantly affect the rankings, they could be disregarded safely. Problems would only arise where the background checks were necessary to distinguish two different objects from one another, when separate probability thresholds might have to be applied to each object to ensure that the results obtained would be correct. (See Section 9.2.4).

The construction of the blackboard system was again simplified as far as possible. The controlling and scheduling functions of the blackboard were reduced to a minimum by having each knowledge source assign ratings to its own bids rather than having the scheduler calculate bid ratings. As the knowledge sources all performed distinctly different functions, there was no need to consider whether any of the bids in the bid list had become redundant and should be removed when a knowledge source operation had been carried out; this, too, simplified the control element of the system.

The designers of most of the general-purpose blackboard systems reviewed in Section 4.6 adopted the policy of attempting to build in all the facilities which might be required for any application. This blackboard illustrates an alternative approach: producing a minimal system to which only those facilities which are definitely required can be added ensures that the final system will not be burdened with unnecessary overheads.

The incorporation of rule induction and feedback into the blackboard system caused the greatest problems, but a neat solution was found to the difficulties this raised: the rules were treated as data and held on a

secondary blackboard. The recogniser knowledge source was effectively a general-purpose recogniser containing empty rule slots which could be customised to recognise a particular object by inserting the appropriate rule data into these slots. This technique should be fairly generally applicable as rule induction systems are almost invariably constrained to induce rules of a limited number of different patterns.

The use of standard rule patterns meant that all possible objects must be recognisable by the same types of rules; this could limit the applicability of the system, so it was decided that a facility to incorporate different types of recognition rules into the system, in the form of background checks, should be provided. The background checks, of course, are not induced - they are programmed by the system user.

The net effect of all these simplifications is that a system has been produced which is conceptually quite simple and easy to understand, and runs quite quickly because of the very small number of calculations required, but nevertheless identifies the objects in test images effectively even when presented with very limited amounts of training data.

10.2. SUGGESTIONS FOR FURTHER WORK.

The image identifier which has been developed can be regarded as a basic experimental system to which a number of extra facilities could be added. Some of these facilities would require close co-operation between the image identifier and the feature matcher, which could not be achieved while the feature matcher was unavailable; others would depend on the

particular application for which the system was to be used.

The system currently operates in a bottom-up manner - the identification of images proceeds through several clearly defined stages (identification of image features, formation of feature sets, selection of candidate sets which represent object instances) which are always carried out in the same order. It would perhaps be useful to introduce a top-down element: an initial set of feature patterns could be used to identify areas of the image which contained object instances, then the feature matcher could be re-invoked to search these areas for further features which would confirm the identity of the objects. Restricting the search for the secondary set of features to specific areas of the image should speed up the matching process significantly.

The running time of the identifier is heavily dependent on the number of parts in each object; the inclusion of one additional object part will approximately double the number of different feature sets to be considered. The identification of objects with large numbers of parts could perhaps be tackled best by adopting a layered approach. The objects could be split into sub-assemblies; when these sub-assemblies have been identified from the primary features which match their parts they could themselves be entered on the problem blackboard as secondary features, from which the objects could then be identified.

The system currently notifies the user of the objects it has identified, but makes no attempt to put this information to any further use. An Interpreter knowledge source, which would be invoked after Selector and before Feedback, could be included to produce an interpretation of the results; this could be a verbal description of the image (for example,

"there is a small lorry facing right in the centre of the image, with a car facing right to the left of it"), or a graphical representation of the objects which have been identified. Alternatively, any specific information which might be required for a particular application of the system could be extracted from the output.

10.3. CONCLUSIONS.

A new approach to the identification of objects in visual images has been developed. The approach uses the data produced by a feature matcher program developed at R.S.R.E. Malvern, which finds the best matches for a set of feature patterns in a preprocessed image. It involves first identifying individual image features which could correspond to specific parts of objects, then forming sets of these features which could correspond to complete objects, the rules governing the formation of feature sets being learnt from training examples in which the locations of object parts are specified. The technique is simple, very tolerant of noise, requires minimal computation and appears to produce very good results.

The system has been implemented using a blackboard system in which the learning element has been incorporated by the simple expedient of treating the induced rules as data. This allows a range of different objects to be recognised by a single recogniser knowledge source; new objects can be introduced without any alteration to the program, producing a degree of flexibility which is unusual in image identification systems.

There is considerable scope for continued development; the inclusion of a top-down element, the introduction of a layered technique for the

identification of more complex objects and/or the addition of an interpreter to further process the output could increase the system's speed and versatility.

Tests have been conducted using synthetic data, with very promising results; the system was able to identify geometric shapes such as squares and triangles even where the shapes overlapped or were partially occluded. Further tests using feature data extracted manually from photographs of vehicles travelling along a main road showed that cars, vans and lorries could be distinguished from one another with a high degree of reliability.

REFERENCES AND BIBLIOGRAPHY

Arbib M.A. 1972: The Metaphorical Brain. An Introduction to Cybernetics as Artificial Intelligence and Brain Theory. John Wiley & Sons.

Ashby W.R. 1960: Design for a Brain. John Wiley & Sons.

Bacchus, F. 1988: Representing and Reasoning with Probabilistic Knowledge. University of Alberta Ph.D. Thesis.

Baldwin J.F. 1983: F.R.I.L. - An Inference Language Based on Fuzzy Logic. Expert Systems '83, Churchill College, Cambridge 14 - 16 Dec. 1983.

Baldwin J.F., Martin T.P. & Pilsworth B.W. 1988: Support Logic Programming. In: FRIL Tutorial Manual, FRIL Systems Ltd.

Barrow H.G. 1989: A.I., Neural Networks and Early Vision. A.I.S.B. Quarterly No. 69, pp. 6 - 25.

Bobrow D.G. & Collins A. 1975: Representation and Understanding. Academic Press.

Boden M. 1977: Artificial Intelligence and Natural Man. The Harvester Press.

Bodington R. & Elleby P. 1988: Justification and Assumption Based Truth Maintenance Systems: When and How to Use Them. Workshop on Reason Maintenance Systems & their Applications, University of Leeds, 14-15 April 1988.

Bodington R.M., Sullivan G.D. & Baker K.D. 1990: Experiments on the use of the ATMS to label features for object recognition. Proc. of First European Conference on Computer Vision (ECCV 90), Antibes, France, April 1990. pp. 542 - 551.

Bowen J.B. & Mayhew J.E.W. 1988:
Consistency Maintenance in the REVgraph Environment. Workshop on Reason Maintenance Systems & their Applications, University of Leeds, 14-15 April 1988.

Brady M. 1986: Machine Vision - The Advent of Intelligent Robots. Addison-Wesley AI Masters. pp. 7 - 63.

Brady M., Agre P.E., Braunegg D.J. & Connell J. 1984:
The Mechanic's Mate. Proc. of 6th European Conference on A.I., ECAI-84, Pisa, Italy 5-7 Sept. 1984.

Bratko I. 1986: Prolog Programming for Artificial Intelligence. Addison-Wesley.

Bruner J.S., Goodnow J.L. & Austin G.A. 1956:
A Study of Thinking. John Wiley & Sons.

Buchanan B.G., Feigenbaum E.A. & Lederberg J. 1971:
A Heuristic Programming Study of Theory Formation in Science. IJCAI-2, London, 1971. pp. 40 - 48.

Buchanan B.G. & Shortliffe E.H. 1984:
Rule-Based Expert Systems. The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley.

Bundy A., Silver B. & Plummer D. 1983:
An Analytical Comparison of Some Rule Learning Programs. Expert Systems '83, Churchill College, Cambridge 14-16 Dec. 1983.

Bundy A., Silver B. & Plummer D. 1985:
An Analytical Comparison of some Rule Learning Programs. Artificial Intelligence Vol. 27 No.2. pp. 137 - 181.

Carbonell J.G., Michalski R.S. & Mitchell T.M. 1983:
An Overview of Machine Learning. In: Machine Learning An A.I. Approach. Ed. Michalski, Carbonell & Mitchell. Tioga Press.

Clocksin W.F. & Mellish C.S. 1984:
Programming in Prolog 2nd Edition. Springer-Verlag.

Corlett R.A. & Todd S.J. 1986:
A Monte Carlo Approach to Uncertain Inference. In: Artificial Intelligence and its Applications. Ed. Cohn & Thomas. John Wiley & Sons. pp. 127 - 137.

de Kleer J. 1984: Choices without Backtracking. Proc. of Conference of the American Association for Artificial Intelligence, Austin, Texas, August 1986. pp. 79 - 85.

de Kleer J. 1986(a): An Assumption-based T.M.S. Artificial Intelligence Vol. 28 pp. 127 - 162.

de Kleer J. 1986(b): Extending the ATMS. Artificial Intelligence Vol. 28 pp. 163 - 196.

de Kleer J. 1986(c): Problem Solving with the ATMS. Artificial Intelligence Vol. 28 pp. 197 - 224.

de Kleer J. & Williams B.C. 1986:
Back to Backtracking: Controlling the ATMS. Proc. of 5th National Conference on A.I., AAAI-86, August 1986.

Dietterich T.G. & Michalski R.S. 1979:
Learning and Generalization of Characteristic Descriptions: Evaluation Criteria and Comparative Review of Selected Methods. IJCAI-'79 Vol. 1 pp. 223 - 231.

Dietterich T.G. & Michalski R.S. 1983:
A Comparative Review of Selected Methods for Learning from Examples. In: Machine Learning An A.I. Approach, Ed. Michalski, Carbonell & Mitchell. Tioga Press. pp. 41 - 81.

DiManzo M., Adorni G., Giunchiglia F. & Ricci F. 1985:
Building Functional Descriptions. Proc. of 5th International Conference on Robot Vision and Sensory Controls, Amsterdam, 29-31 Oct. 1985. pp. 403 - 412.

Dodd D.H. & White R.M. 1980:
Cognition: Mental Structures and Processes. Allyn & Bacon.

Doyle J. 1979(a): A Glimpse of Truth Maintenance. IJCAI-'79 Vol. 1 pp. 232 - 237.

Doyle J. 1979(b): A Truth Maintenance System. Artificial Intelligence Vol. 12. pp. 231 - 272.

Draper B.A., Collins R.T., Brolio J., Hanson A.R. & Riseman E.M. 1988:
Issues in the Development of a Blackboard-based Schema System for Image Understanding. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison Wesley.

Duda R.O. & Hart P.E. 1973:
Pattern Classification and Scene Analysis. John Wiley & Sons.

Edmonds E.A. 1981: Domains of Interest in Fuzzy Sets. Int. Journal of Man-Machine Studies No. 15 pp. 461 - 468.

Engelmore R.S. & Morgan A.J. (Eds.) 1988:
Blackboard Systems. Addison Wesley.

Engelmore R.S., Morgan A.J. & Nii H.P. 1988:
Blackboard Systems - Introduction. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison Wesley.

Erman L.D., Hayes-Roth F., Lesser V.R. & Reddy D.R. 1988:
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison-Wesley.

Erman L.D., London P.E. & Fickas S.F. 1988:
The Design and an Example Use of Hearsay-III. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison-Wesley.

Farreny H. 1989: A.I. and Expertise: Heuristic Search, Inference Engines, Automatic Proving. Ellis Horwood.

Feigenbaum E.A. 1983: Knowledge Engineering: The Applied Side. In: Intelligent Systems - the Unprecedented Opportunity. Ed. Hayes & Michie. Ellis Horwood.

Forsyth R. & Rada R. 1986: Machine Learning: Applications in Expert Systems and Information Retrieval. Ellis Horwood.

Fox J. 1986: Knowledge, Decision Making, and Uncertainty. In: Artificial Intelligence and Statistics. Ed. Gale. Addison-Wesley. pp. 57 - 76.

Fox J. 1987: Dealing With Uncertainty. In: Intelligent Knowledge-Based Systems - An Introduction. Ed. O'Shea, Self & Thomas. Harper & Row. pp. 52 - 67.

Franksen O.I. 1978: On Fuzzy Sets, Subjective Measurements, and Utility. Workshop on Fuzzy Reasoning: Theory and Applications. Queen Mary College, Univ. of London. 15 Sept. 1978.

Freeman P. & Newell A. 1971: A Model for Functional Reasoning in Design. IJCAI-2, London, 1971. pp. 621 - 640.

Fretwell P., Goillau P. & Hearn D.B. 1987: Using the Essence of an Object in Computer Recognition. Research Note SP4, R.S.R.E. Pattern Processing and Machine Intelligence Division. March 1987.

Fretwell P. & Goillau P.J. 1987: Linguistic Definition of Generic Models in Computer Vision. In: Lecture Notes in Computer Science Vol. 301. Ed. Kittler. Springer-Verlag. pp. 306 - 314.

Frost R.A, 1986: Introduction to Knowledge Based Systems. Collins.

Gabrielides G. 1988: A System that Learns to Recognise 3-D Objects. Loughborough University of Technology Ph.D. Thesis.

Gaines B.R. 1976: Foundations of Fuzzy Reasoning. Int. Journal of Man-Machine Studies. No. 8. pp. 623 - 668.

Gale W.A. 1986: Artificial Intelligence and Statistics. Addison-Wesley.

Garnham A. 1987: Artificial Intelligence - An Introduction. Routledge & Kegan Paul.

Gaschnig J. 1982: Prospector: An Expert System for Mineral Exploration. In: Introductory Readings in Expert Systems. Ed. Michie. Gordon & Breach. pp. 47 - 64.

Goldberg A. & Robson D. 1989:
Smalltalk-80. the Language. Addison-Wesley.

Gordon J. & Shortliffe E.H. 1985:
A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space. Artificial Intelligence Vol. 26 No. 3 pp. 323 - 357.

Hart A. 1986: Knowledge Acquisition for Expert Systems. Kegan Paul.

Heckerman D. 1986: Probabilistic Interpretations for MYCIN's Certainty Factors. In: Uncertainty in Artificial Intelligence. Ed. Kanal & Lemmer. Elsevier. pp. 167 - 196.

Hinde C.J. 1985: Artificial Intelligence and Expert Systems. In: Further Developments in Operational Research. Ed. Rand & Eglese. Pergamon Press.

Hinde C.J. 1986: Fuzzy Prolog. In: Int. Journal of Man-Machine Studies Vol. 24 pp. 569 - 595.

Hinde C.J., Bray A.D., Herbert P.J., Launders V.A. & Round D. 1989:
A Truth Maintenance Approach to Process Planning. In: Artificial Intelligence in Manufacturing. Ed. Rzevski. Springer-Verlag. pp. 171 - 188.

Holmes J.N. 1988: Speech Synthesis and Recognition. Van Nostrand Reinhold. pp. 102 - 127.

Johnson R.W. 1986: Independence and Bayesian Updating Methods. Artificial Intelligence Vol. 29 No. 2. pp. 217 - 222.

Jones J. & Millington M. 1986:
An Edinburgh Prolog Blackboard Shell. University of Edinburgh Dept. of Artificial Intelligence Research Paper No. 281.

Kodratoff Y., Manago M. & Blythe J. 1988:
Generalization and Noise. In: Knowledge Acquisition for Knowledge-Based Systems. Ed. Gaines & Boose. Academic Press. pp. 301 - 324.

Kodratoff Y. 1988: Introduction to Machine Learning. Pitman.

Kotz S. & Stroup D.F. 1983: Educated Guessing. Marcel Dekker.

Laird P.D. 1988: Learning from Good and Bad Data. Kluwer Academic Press.

Langley P., Bradshaw G.L & Simon H.A. 1983:
Rediscovering Chemistry with the BACON System. In: Machine Learning: An Artificial Intelligence Approach. Ed. Michalski, Carbonell & Mitchell. Tioga Press. pp. 307 - 329.

Lindsay P.H. & Norman D.A. 1977:
Human Information Processing. Academic Press.

Loisel R. & Kodratoff Y. 1981:
Learning (Complex) Structural Descriptions from Examples. IJCAI-'81 Vol. 1. pp. 141 - 143.

Marr D. 1979: Representing and Computing Visual Information. In: Artificial Intelligence: An MIT Perspective. Ed. Winston & Brown. MIT Press. pp. 17 - 80.

McAllester D.A. 1978: A Three-Valued Truth Maintenance System. M.I.T. AI Memo 473.

McGregor J.N. 1988: The Effects of Order on Learning Classifications by Example: Heuristics for Finding the Optimal Order. Artificial Intelligence Vol. 34 No. 3. pp. 361 - 370.

MacLennan B.J. 1983: Principles of Programming Languages: Design, Evaluation and Implementation. Holt-Saunders.

Mamdani E.H. & Gaines B.R. 1981:
Fuzzy Reasoning and its Applications. Academic Press.

Michalski R.S. 1983: A Theory and Methodology of Inductive Learning. In: Machine Learning An A.I. Approach. Ed. Michalski, Carbonell & Mitchell. Tioga Press.

Michalski R.S., Carbonell J.G. & Mitchell T.M. (Eds.) 1983:
Machine Learning An A.I. Approach. Tioga Press.

Michie D. 1986: On Machine Intelligence. Ellis Horwood.

Michie D. & Bratko I. 1986: Expert Systems - Automating Knowledge Acquisition. Addison-Wesley AI Masters. pp. 7 - 20.

Minsky M. 1975: A Framework for Representing Knowledge. In: The Psychology of Computer Vision. Ed. Winston. McGraw-Hill. pp. 211 - 277.

Mitchell T.M. 1979: An Analysis of Generalization as a Search Problem. IJCAI-'79 Vol. 1 pp. 577 - 582.

Nafarieh A. 1988: A New Approach to Inference in Approximate Reasoning and its Application to Computer Vision. University of Missouri-Columbia Ph.D. Thesis.

Nagao M., Matsuyama T. & Mori H. 1988:
Structural Analysis of Complex Aerial Photographs. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison-Wesley.

Neapolitan R.E. 1990: Probabilistic Reasoning in Expert Systems. John Wiley & Sons. pp. 1 - 94.

Newell A. & Simon H.A. 1972:
Human Problem Solving. Prentice-Hall.

Nii H.P. & Aiello N. 1988: AGE (Attempt to GEneralize): A Knowledge-Based Program for Building Knowledge-Based Programs. In: Blackboard Systems. Ed. Engelmore & Morgan. Addison-Wesley.

Norman M. 1987: A Prolog Set-theoretic Equation Solver. Loughborough University of Technology M.Sc. Thesis.

Ohta Y. 1985: Knowledge-Based Interpretation of Outdoor Natural Color Scenes. Pitman.

Pearl J. 1984: Heuristics - Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley.

Pearl J. 1988: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann.

Provan G.M. 1987: Efficiency Analysis of Multiple-Context TMSs in Scene Representation. University of Oxford Technical Report OU-RRG-87-9.

Provan G.M. 1990: An Analysis of Knowledge Representation Schemes for High level Vision. Proc. of First European Conference on Computer Vision (ECCV 90), Antibes, France, April 1990. pp. 537 - 541.

Quinlan J.R. 1982: Semi-Autonomous Acquisition of Pattern-Based Knowledge. In: Machine Intelligence 10. Ed. Hayes, Michie & Pao. Ellis Horwood.

Rosin P. 1988: Model-Driven Image Understanding - a Frame-Based Approach. City University Ph.D. Thesis.

Ross P. 1989: Advanced Prolog Techniques and Examples. Addison Wesley.

Sammut C. 1981: Concept Learning by Experiment. IJCAI-'81 Vol. 1. pp. 104 - 105.

Schmucker K.J. 1984: Fuzzy Sets, Natural Language Computations, and Risk Analysis. Computer Science Press.

Schutzer D. 1987: Artificial Intelligence. An Applications-Oriented Approach. Van Nostrand Reinhold. pp. 36 - 42.

Series R.W., Radford C.J., Varga M.J., Fretwell P. & Sleigh A.C. 1989: Comparison of Approaches to Feature Detection. Proc. of 5th Alvey Vision Conference, Reading, 1989.

Shanahan M. & Southwick R. 1989: Search, Inference and Dependencies in Artificial Intelligence. Ellis Horwood.

Shapiro E.Y. 1981: An Algorithm that Infers Theories from Facts. IJCAI-'81 Vol. 1. pp. 446 - 451.

Shapiro E.Y. 1982: Algorithmic Program Debugging. MIT Press.

Shepherd A. & Hinde C.J. 1989: Mimicking the Training Expert: A Basis for Automating Training Needs Analysis. In: Developing Skills with Information Technology. Ed. Bainbridge & Quintanilla. John Wiley & Sons. pp. 153 - 176.

Simon H.A. 1983: Why Should Machines Learn? In: Machine Learning An A.I. Approach. Ed. Michalski, Carbonell & Mitchell. Tioga Press.

Smith E.E. & Medin D.L. 1981: Categories and Concepts. Harvard University Press.

Sobolevitch N. 1985: An Overview of the Battlefield Assessment and Current Holdings (BACH) Expert System. Ist International Expert Systems Conference. London, 1-3 Oct. 1985. pp. 41 - 69.

Sowa J.F. 1984: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley.

Spiegelhalter D.J. 1986: A Statistical View of Uncertainty in Expert Systems. In: Artificial Intelligence and Statistics. Ed. Gale. Addison-Wesley. pp. 17 - 55.

Stallman R. & Sussman G. 1977:
Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis. Artificial Intelligence Vol. 9. p. 135.

Sterling L. & Shapiro E. 1986:
The Art of Prolog: Advanced Programming Techniques. MIT Press.

Varga M.J., Radford C.J., Series R.W. & Sleigh A.C. 1989:
A Two-stage Automatic Object Identification Technique. Private communication.

Varga M.J., Sleigh A.C. & Series R.W. 1989:
The Application of Dynamic Programming to Object Identification. 5th International Conf. on Image Analysis and Processing, Italy, Sept '89.

Walker E.L. & Herman M. 1988:
Geometric Reasoning for Constructing 3D Scene Descriptions from Images. Artificial Intelligence Vol. 37 pp. 275 - 290.

Winston P.H. 1975: Learning Structural descriptions from Examples. In: The Psychology of Computer Vision. Ed. Winston. McGraw-Hill. pp. 157 - 209.

Winston P.H., Binford T.O., Katz B. & Lowry M. 1983:
Learning Physical Descriptions from Functional Definitions, Examples and Precedents. MIT A.I. Memo.

Wise B.P. 1986: An Experimental Comparison of Uncertain Inference Systems. Carnegie-Mellon University Ph.D. Thesis. pp. 1 - 61.

Wysotzki F., Kolbe W. & Selbig J. 1981:
Concept Learning by Structured Examples - an Algebraic Approach. IJCAI-'81 Vol. 1. pp. 153 - 158.

Zadeh L.A. 1986: Is Probability Theory Sufficient for dealing with Uncertainty in AI: A Negative View. In: Uncertainty in Artificial Intelligence. Ed. Kanal & Lemmer. Elsevier. pp. 103 - 116.

APPENDIX A

Listing of the Image Identifier (RECOGNISE2)

```
/* recognise2/master */
/* ********************************************************************** */
/* ****                RECOGNISE2 Master File                        **** */
/* ********************************************************************** */

:-[init].
:-[run].
:-[consult].
:-[enter].
:-[changes].
:-[schedule].
:-[show].
:-[utilities_master].
:-[editor_master].
:-[learner_master].
:-[recogniser_master].
:-[acceptor_master].
:-[selector_master].
:-[feedback_master].
:-[remover_master].
```

```
/* recognise2/init */
/* ********************************************************************** */
/* ****                      Initialisation                          **** */
/* ********************************************************************** */

/* The initialisation process sets up the bid list, list of names  */
/* of learnt objects and no. of object occurrences, reads in the   */
/* duplicates limit, percentage probability thresholds for feature */
/* set acceptance and referral to the user and manual/automatic    */
/* bid execution selector, then calls consult_user.                */

init:-
        assert(bid_list([])),
        assert(recognisebb1(object_names,([]))),
        assert(recognisebb4(occurrences,(total,0))),
        nl,
        write('Duplicates limit? '),
        read_number(Dup,[0,10]),
        asserta(duplicates_limit(Dup)),
        write('Percentage prob. threshold for automatic'),
        write(' acceptance?'),nl,
        write('(For no automatic acceptance, enter 110)  '),
        read_number(Accept,[0,110]),
        asserta(acceptance_threshold(Accept)),
        write('Percentage prob. threshold for referral to user? '),
        read_number(Refer,[0,Accept]),
        asserta(referral_threshold(Refer)),
        write('Bid execution manual (m) or automatic (a)? '),
        read_char(Autorun,[a,m]),
        assert(autorun(Autorun)),
        consult_user.
```

```
/* recognise2/run */
/* ************************************************************** */
/* ****                      Run the system                  **** */
/* ************************************************************** */

/* run removes the top bid from the bid list and executes it.  If  */
/* manual/automatic execution selector is set to automatic it then */
/* runs the next bid; if the selector is set to manual, or if      */
/* the bid list is empty, it calls consult_user.                   */

run:-
        bid_list([]),
        write('Bid list empty'),nl,
        consult_user.

run:-
        bid_list([((Function,Data),Rating)|Rest]),
        retract(bid_list([((Function,Data),Rating)|Rest])),
        assert(bid_list(Rest)),
        execute(Function,Data),
        (autorun(a), !, run; consult_user).
```

```
/* recognise2/consult */
/* ********************************************************************** */
/* ****                    Consult system user                       **** */
/* ********************************************************************** */

/* consult_user is called between bids (manual bid execution) or   */
/* when the bid list is empty (automatic bid execution).  It       */
/* includes a 'help' module.                                       */

consult_user:-
        nl,
        write('What now?'),nl,
        write('(Enter "help." to view options)'),nl,
        read(Command),nl,
        execute_command(Command).

execute_command(help):-
        write('Options available are:'), nl,
        write('Read in a data file: "[filename]."'),nl,
        (       autorun(m),
                write('Execute the next bid: "run."'), nl,
                write('Switch to automatic bid execution: "auto."'),
                nl, !;
                write('Switch to manual bid execution: "man."'),nl),
        write('View the recognition rules for an object: '),
        write('"show_rules(Object)."'), nl,
        write('Alter acceptance/referral thresholds: "alter."'),nl,
        write('Quit the system: "quit."'),
        nl,
        consult_user.

execute_command(quit):- !.

execute_command(run):- run.

execute_command([Filename]):-
        consult(Filename),
        bb_enter,
        (       autorun(a), run;
                consult_user            ).

execute_command(show_rules(Object)):-
        show_rules(Object),
        consult_user.

execute_command(auto):-
        retract(autorun(m)),
        assert(autorun(a)),
        run.

execute_command(man):-
        retract(autorun(a)),
        assert(autorun(m)),
        run.

execute_command(alter):-
        nl,
        acceptance_threshold(A1),
        write('Current automatic acceptance threshold is '),
        write(A1), nl,
        write('New value? '),read_number(A2,[0,110]),
        retract(acceptance_threshold(A1)),
        asserta(acceptance_threshold(A2)),
        referral_threshold(R1),
        write('Current threshold for referral to user is '),
        write(R1), nl,
```

```
        write('New value? '),read_number(R2,[0,100]),
        retract(referral_threshold(R1)),
        asserta(referral_threshold(R2)),
        consult_user.

execute_command(_):-
        nl, write('Unrecognised command'),nl,
        consult_user.
```

```
/* recognise2/enter */
/* ********************************************************** */
/* ****          Enter problem data on the blackboard          **** */
/* ********************************************************** */

/* bb_enter is used to enter data on the problem blackboard.  As   */
/* features and object parts are normally entered in batches, these*/
/* entries are flagged (using bb_entered) so that new_entry can be */
/* called for each batch rather than for each individual entry.    */

bb_enter:-
        feature(PicNo,PatNo,X,Y,Rating),
        assert(problembb1(feature,(PicNo,PatNo,X,Y,Rating))),
        retract(feature(PicNo,PatNo,X,Y,Rating)),
        (       bb_entered(feature,PicNo);
                assert(bb_entered(feature,PicNo))       ),
        !,bb_enter.
bb_enter:-
        object_part(Object,PicNo,Inst,PartNo,X,Y),
        assert(problembb4(object_part,(Object,PicNo,Inst,PartNo,X,Y))),
        retract(object_part(Object,PicNo,Inst,PartNo,X,Y)),
        (       bb_entered(object_part,Object,PicNo);
                assert(bb_entered(object_part,Object,PicNo))    ),
        !,bb_enter.
bb_enter:-
        part_list(Object,List),
        assert(recognisebb1(part_list,(Object,List))),
        retract(part_list(Object,List)),
        !,bb_enter.
bb_enter:-
        rating_threshold(F,T),
        assert(recognisebb1(rating_threshold,(F,T))),
        retract(rating_threshold(F,T)),
        !,bb_enter.

bb_enter:-
        bb_entered(feature,PicNo),
        new_entry([problembb1,feature,(PicNo)]),
        retract(bb_entered(feature,PicNo)),
        !,bb_enter.
bb_enter:-
        bb_entered(object_part,Object,PicNo),
        (       training(PicNo), !; assert(training(PicNo))     ),
        new_entry([problembb4,object_part,(Object,PicNo)]),
        retract(bb_entered(object_part,Object,PicNo)),
        !,bb_enter.

bb_enter.
```

```
/* recognise2/changes */
/* ********************************************************************** */
/* ****      Check blackboard changes, make new bids              **** */
/* ********************************************************************** */

/* new_entries/removed_entries are called when entries are added   */
/* to/removed from the problem blackboard.  The knowledge sources  */
/* which are interested in the changes are then invited to make    */
/* new bids.                                                       */
new_entries([]):- !.

new_entries([First|Rest]):-
        new_entry(First),
        new_entries(Rest).

new_entry([Section,Datatype,Data]):-
        wants(KS,Section,Datatype),
        make_bid(KS,[Section,Datatype,Data]),
        fail.
new_entry(_).

removed_entries([]):- !.

removed_entries([First|Rest]):-
        removed_entry(First),
        removed_entries(Rest).

removed_entry([Section,Datatype,Data]):-
        wants_removed(KS,Section,Datatype),
        make_bid(KS,[Section,Datatype,Data]),
        fail.
removed_entry(_).
```

```
/* recognise2/schedule */
/* ********************************************************************** */
/* ****                Insert new bids in bid list                   **** */
/* ********************************************************************** */

schedule(Function,Data,Rating):-
        bid_list(List),
        (       member(((Function,Data),Rating),List),  !;
                insert1(((Function,Data),Rating),List,NewList),
                retract(bid_list(List)),
                assert(bid_list(NewList))                ).
```

```
/*recognise2/show*/
/* ********************************************************** */
/* ****                        Show                      **** */
/* ********************************************************** */

/* show_rules(Object) can be called by the user to list the        *
/* recognition rules for an Object which has been learnt.          *

show_rules(Object):-
        list_matches(Object),
        list_limits(Object),
        list_sets(Object).

list_matches(Object):-
        nl,
        write('Part/Pattern No. matches:'),
        nl,
        write_matches(Object).
write_matches(Object):-
        recognisebb2(match,(Object,PartNo,PatternNo)),
        recognisebb1(part_list,(Object,Parts)),
        member((PartNo,PartName),Parts),
        write(PartName),write(', Pattern'),write(PatternNo),nl,
        fail.
write_matches(Object).

list_limits(Object):-
        nl,
        write('Distance limits(Part1,Part2,MinX,MaxX,MinY,MaxY):'),
        nl,
        write_limits(Object).
write_limits(Object):-
        recognisebb2(distance_limits,(Object,Part1,Part2,
                                MinX,MaxX,MinY,MaxY)),
        recognisebb1(part_list,(Object,Parts)),
        member((Part1,PartName1),Parts),
        member((Part2,PartName2),Parts),
        write(PartName1),write(','),write(PartName2),
                write(','),write(MinX),write(','),write(MaxX),
                write(','),write(MinY),write(','),write(MaxY),
        nl,
        fail.
write_limits(Object).

list_sets(Object):-
        nl,
        write('Set probabilities(PartSet,PatternSet,FeatureSets,'),
                                write('Matches):'), nl,
        write_sets(Object).
write_sets(Object):-
        recognisebb3(set_probability,(Object,PartSet,PatternSet,
                                        Sets,Matches)),
        write(PartSet),write(','),write(PatternSet),write(','),
        write(Sets),write(','),write(Matches),
        nl,
        fail.
write_sets(Object).
```

```
/*recognise2/utilities_master*/
/* ************************************************************** */
/* ****                  Utilities Master File                 **** */
/* ************************************************************** */

:-[general_utilities].
:-[make_sets].
:-[find].
:-[check_limits].
:-[feature_match].
:-[check_match].
:-[identify].
:-[write_set].
:-[remove_instances].
```

```
/*recognise2/general_utilities*/
/* ********************************************************************** */
/* ****                    General Utilities                         **** */
/* ********************************************************************** */

member(A,[A|_]).
member(A,[_|X]):- member(A,X).

nonmember(A,X):-
        member(A,X), !, fail.
nonmember(A,X).

disjoint(X,Y):-
        member(A,X),
        member(A,Y),
        !,fail.
disjoint(X,Y).

add(A,X,X):-
        member(A,X), !.
add(A,X,[A|X]).

unite([],X,X):- !.
unite([A|X],Y,U):-
        unite(X,Y,U2),
        add(A,U2,U).

near(X,Y):-
        duplicates_limit(Dup),
        Diff is X-Y,
        Diff < Dup,
        Diff > -Dup.

read_char(Input1,List):-
        read(Input2),
        (       member(Input2,List), Input1 = Input2, !;
                write('Unacceptable input - try again? '),
                read_char(Input1,List)          ).

read_number(Input1,[Min,Max]):-
        read(Input2),
        (       Input2 >= Min, Input2 =< Max, Input1 = Input2, !;
                write('Unacceptable input - try again? '),
                read_number(Input1,[Min,Max])           ).

insert1((A,B),[],[(A,B)]):- !.
insert1((A1,B1),[(A2,B2)|R],[(A1,B1),(A2,B2)|R]):-
        B1 >= B2, !.
insert1((A1,B1),[(A2,B2)|R],[(A2,B2)|New]):-
        insert1((A1,B1),R,New).

insert((A,B,C),[],[(A,B,C)]):- !.
insert((A1,B1,C1),[(A2,B2,C2)|R],[(A1,B1,C1),(A2,B2,C2)|R]):-
        C1 >= C2, !.
insert((A1,B1,C1),[(A2,B2,C2)|R],[(A2,B2,C2)|New]):-
        insert((A1,B1,C1),R,New), !.

delete(X,[X|List],List).
delete(X,[Y|List],[Y|NewList]):-
        delete(X,List,NewList).
```

```
/* recognise2/make_sets */
/* ********************************************************************** */
/* ****                        Make Part Sets                          **** */
/* ********************************************************************** */

/* make_sets(Object) finds all the sets of features which could    */
/* represent sets of Object parts, checks to see which of these    */
/* sets match known instances of Object, then counts the no. of    */
/* feature sets and the no. of matches for each set of Object parts*/
/* and corresponding set of feature patterns, recording these in   */
/* the set_probability table on recognisebb3.  The probability that*/
/* a set of features will represent an instance of Object can be   */
/* calculated from the appropriate entry in this table; the        */
/* percentage probability is (100*Matches)/Sets.                   */

make_sets(Object):-
        find_set(Object,PicNo,Parts,Patterns,Features),
        check_match(Object,PicNo,Parts,Features,Inst),
        assert(problembb3(feature_set,(Object,PicNo,0,Parts,
                                Patterns,Features,Inst))),
        update_probability(Object,Parts,Patterns,Inst),
        fail.
make_sets(Object):-
        (       retractall(matched_feature(Object,_,_,_,_,_));true),
        (       retractall(within_limits(Object,_,_,_));true),
        (       retractall(problembb3(feature_set,(Object,_,_,_,_,
                                                _,_))); true).
```

```
/* recognise2/find */
/* ********************************************************** */
/* ****                  Find Feature Sets                 **** */
/* ********************************************************** */

/* find_set(Object,PicNo,Parts,Patterns,Features) returns a list of*/
/* two or more object parts (arranged in descending order of part  */
/* no.) and a list of features in the specified picture which could*/
/* represent these parts, checking for feature pattern/object part */
/* matches and satisfaction of the limits on x and y co-ordinate   */
/* differences between pairs of object parts.                      */
/* Repeated calls to find_set will yield all such part lists/      */
/* feature lists, starting with those with only two elements.      */
/* Longer lists are found by adding higher-numbered parts to the   */
/* start of existing part lists.                                   */

find_set(Object,PicNo,Parts,Patterns,Features):-
        find_pairs(Object,PicNo,Parts,Patterns,Features).
find_set(Object,PicNo,Parts,Patterns,Features):-
        find_multiples(Object,PicNo,Parts,Patterns,Features).

/* Find two-element part lists/feature lists                       */

find_pairs(Object,PicNo,[Part1,Part2],[Pattern1,Pattern2],[F1,F2]):-
        recognisebb1(part_list,(Object,PartList)),
        member((Part2,_),PartList),
        member((Part1,_),PartList),
        Part1 > Part2,
        recognisebb2(match,(Object,Part1,Pattern1)),
        problembb2(feature,(PicNo,F1,Pattern1,X1,Y1)),
        recognisebb2(match,(Object,Part2,Pattern2)),
        problembb2(feature,(PicNo,F2,Pattern2,X2,Y2)),
        check_limits(Object,Part1,X1,Y1,Part2,X2,Y2),
        assert(within_limits(Object,PicNo,[Part1,Part2],[F1,F2])).


/* Find larger feature list                                        */

find_multiples(Object,PicNo,[Part2,Part1|Rest],[Pattern2|Patterns],
                                               [F2|Features]):-
        problembb3(feature_set,(Object,PicNo,_,[Part1|Rest],Patterns,
                                               Features,_)),
        recognisebb1(part_list,(Object,PartList)),
        member((Part2,_),PartList),
        Part2 > Part1,
        recognisebb2(match,(Object,Part2,Pattern2)),
        problembb2(feature,(PicNo,F2,Pattern2,X,Y)),
        check_pair_limits(Object,PicNo,Part2,F2,[Part1|Rest],Features).
```

```
/* recognise2/check_limits */
/* ********************************************************** */
/* ****                Check Distance Limits             **** */
/* ********************************************************** */

/* check_limits(Object,PartNo1,X1,Y1,PartNo2,X2,Y2) succeeds if the*/
/* features whose locations are (X1,Y1) and (X2,Y2) satisfy the    */
/* distance limits between PartNo1 and PartNo2 of Object           */

check_limits(Object,PartNo1,X1,Y1,PartNo2,X2,Y2):-
        recognisebb2(distance_limits,(Object,PartNo1,PartNo2,
                                      MinX,MaxX,MinY,MaxY)),
        DiffX is X1-X2, DiffX >= MinX, DiffX =< MaxX,
        DiffY is Y1-Y2, DiffY >= MinY, DiffY =< MaxY.

/* check_pair_limits(Object,PicNo,PartNo,FNo,PartSet,FSet) succeeds*/
/* if [FNo|FSet] could represent [PartNo|PartSet], i.e. if all the */
/* appropriate distance limits are satisfied.                      */
/* The limits are checked by looking to see if the relevant two-   */
/* element part sets and feature sets have been entered in the     */
/* within_limits table.                                            */

check_pair_limits(Object,PicNo,PartNo,FeatureNo,[],[]).

check_pair_limits(Object,PicNo,PartNo1,FNo1,[PartNo2|RestParts],
                                  [FNo2|RestF]):-
        within_limits(Object,PicNo,[PartNo1,PartNo2],[FNo1,FNo2]),
        check_pair_limits(Object,PicNo,PartNo1,FNo1,RestParts,
                                  RestF).
```

```
/* recognise2/feature_match */
/* ************************************************************** */
/* ****            Match Features to Object Parts              **** */
/* ************************************************************** */

/* feature_match(Object) checks all features against parts of known*/
/* instances of the object for co-ordinate matches; the feature    */
/* details are recorded with the matching part no. and object      */
/* instance no. (or 0,0 if no match) in the matched_feature table. */

feature_match(Object):-
        feature_match(Object,PicNo),
        fail.
feature_match(Object).

feature_match(Object,PicNo):-
        problembb2(feature,(PicNo,FeatureNo,PatternNo,X1,Y1)),
        match_feature(Object,PicNo,FeatureNo,PatternNo,X1,Y1),
        fail.
feature_match(Object,PicNo).

match_feature(Object,PicNo,FeatureNo,PatternNo,X1,Y1):-
        !,
        (       problembb4(object_part,(Object,PicNo,Inst,PartNo,
                                                X2,Y2)),
                X1 == X2, Y1 == Y2,
                problembb2(identified_features,(PicNo,List)),
                retract(problembb2(identified_features,(PicNo,List))),
                assert(problembb2(identified_features,
                                (PicNo,[FeatureNo|List]))),
                !;
                Inst is 0, PartNo is 0),
        assert(matched_feature(Object,PicNo,FeatureNo,PatternNo,
                                                Inst,PartNo)).
```

```
/* recognise2/check_match */
/* ********************************************************** */
/* ****                  Match Feature Sets                  **** */
/* ********************************************************** */

/* check_match(Object,PicNo,PartSet,FeatureSet,Inst) checks to see */
/* if all the features in FeatureSet match the appropriate parts of*/
/* an instance of the Object, and sets the value of Inst to the    */
/* instance number or, if there is no match, to 0.                 */

check_match(Object,PicNo,[Part1,Part2],[Feature1,Feature2],Inst):-
        matched_feature(Object,PicNo,Feature1,_,Inst,Part1),
        matched_feature(Object,PicNo,Feature2,_,Inst,Part2),
        !.
check_match(Object,PicNo,[Part1|RestP],[Feature1|RestF],Inst):-
        matched_feature(Object,PicNo,Feature1,_,Inst,Part1),
        problembb3(feature_set,(Object,PicNo,_,RestP,_,RestF,Inst)),
        !.
check_match(Object,PicNo,Parts,Features,0).


/* update_probability(Object,PartSet,PatternSet,Inst) adds         */
/* information about a feature set which has been checked for an   */
/* object instance match to the appropriate entry in the           */
/* set_probability table.                                          */

update_probability(Object,PartSet,PatternSet,Inst):-
        !,
        (       recognisebb3(set_probability,(Object,PartSet,
                                PatternSet,Sets,Matches)),
                retract(recognisebb3(set_probability,(Object,PartSet,
                                PatternSet,Sets,Matches))),
                !;
                Sets is 0, Matches is 0                          ),
        NewSets is Sets+1,
        (       Inst>0, NewMatches is Matches+1, !;
                NewMatches is Matches                            ),
        assert(recognisebb3(set_probability,(Object,PartSet,
                                PatternSet,NewSets,NewMatches))).
```

```
/* recognise2/identify */
/* ********************************************************************** */
/* ****      Identify a Feature Set as an Object Instance      **** */
/* ********************************************************************** */

identify(Object,PicNo,Parts,Features):-
        problembb2(identified_features,(PicNo,IdList)),
        unite(IdList,Features,NewIdList),
        retract(problembb2(identified_features,(PicNo,IdList))),
        asserta(problembb2(identified_features,(PicNo,NewIdList))),
        record_instance(Object,PicNo,Parts,Features).


record_instance(Object,PicNo,Parts,Features):-
        (       instances(Object,PicNo,Current),
                Next is Current+1,
                retract(instances(Object,PicNo,Current)),
                !;
                Next is 1                               ),
        assert(instances(Object,PicNo,Next)),
        record_parts(Object,PicNo,Next,Parts,Features),
        new_entry([problembb4,object_part,(Object,PicNo)]).

record_parts(Object,PicNo,Inst,[],[]):- !.
record_parts(Object,PicNo,Inst,[PartNo|RestParts],
                                [FeatureNo|RestFeatures]):-
        problembb2(feature,(PicNo,FeatureNo,PatternNo,X,Y)),
        assert(problembb4(object_part,(Object,PicNo,Inst,PartNo,
                                                X,Y))),
        record_parts(Object,PicNo,Inst,RestParts,RestFeatures).
```

```
/* recognise2/write */
/* ********************************************************** */
/* ****    Write an Object Part Set and Candidate Feature Set **** */
/* ********************************************************** */

write_set(Object,PicNo,Parts,Features):-
        nl, write('Object:   '),write(Object),nl,
        fwrite(Object,PicNo,Parts,Features),
        !.

fwrite(Object,PicNo,[],[]):- !.
fwrite(Object,PicNo,[Part1|RestP],[F1|RestF]):-
        recognisebb1(part_list,(Object,Parts)),
        member((Part1,Name),Parts),
        write(Name),write(','),
        problembb2(feature,(PicNo,F1,_,X,Y)),
        write('('),write(X),write(','),write(Y),write(')'),nl,
        fwrite(Object,PicNo,RestP,RestF).
```

```
/* recognise2/remove_instances */
/* ************************************************************** */
/* ****          Count and Remove Object Instances           **** */
/* ************************************************************** */


remove_instances(Object):-
        problembb4(object_part,(Object,PicNo,_,_,_,_)),
        remove_instances(Object,PicNo),
        !,
        remove_instances(Object).
remove_instances(Object).

remove_instances(Object,PicNo):-
        problembb4(object_part,(Object,PicNo,Inst,_,_,_)),
        retractall(problembb4(object_part,(Object,PicNo,Inst,_,_,_))),
        recognisebb4(occurrences,(Object,N)),
        N1 is N+1,
        retract(recognisebb4(occurrences,(Object,N))),
        assert(recognisebb4(occurrences,(Object,N1))),
        recognisebb4(occurrences,(total,M)),
        M1 is M+1,
        retract(recognisebb4(occurrences,(total,M))),
        assert(recognisebb4(occurrences,(total,M1))),
        !,remove_instances(Object,PicNo).
remove_instances(Object,PicNo).
```

```
/* recognise2/editor_master */
/* ************************************************************** */
/* ****                  Editor Master File                  **** */
/* ************************************************************** */

:-[editor_bidder].
:-[edit].
```

```
/* recognise2/editor_bidder */
/* ********************************************************************** */
/* ****                      Editor Bidder                           **** */
/* ********************************************************************** */

wants(editor,problembb1,feature).

make_bid(editor,[problembb1,feature,PicNo]):-
        schedule(editor,(PicNo),100).

execute(editor,(PicNo)):-
        edit(PicNo).
```

```
/* recognise2/edit */
/* ********************************************************** */
/* ****                  Edit Feature Data                 **** */
/* ********************************************************** */

/* edit(PicNo) removes from the list of features for PicNo all     */
/* features whose ratings are below the appropriate threshold value*/
/* and all duplicates (i.e. features whose distance from another,  */
/* higher-rated, feature is less than the duplicates limit), then  */
/* numbers the features remaining in each picture and moves them   */
/* from problembb1 to problembb2.                                  */

edit(PicNo):-
        write('Editing picture:'),write(PicNo),
        nl,
        remove_low_ratings(PicNo),
        remove_duplicates(PicNo),
        count_features(PicNo),
        assert(problembb2(identified_features,(PicNo,[]))),
        new_entry([problembb2,feature,(PicNo)]).



remove_low_ratings(PicNo):-
        problembb1(feature,(PicNo,PatternNo,X,Y,Rating)),
        recognisebb1(rating_threshold,(PatternNo,Threshold)),
        Rating < Threshold,
        retract(problembb1(feature,(PicNo,PatternNo,X,Y,Rating))),
        fail.
remove_low_ratings(PicNo).

remove_duplicates(PicNo):-
        problembb1(feature,(PicNo,Pattern1,X1,Y1,Rating1)),
        problembb1(feature,(PicNo,Pattern2,X2,Y2,Rating2)),
        near(X1,X2),
        near(Y1,Y2),
        Rating1 > Rating2,
        retract(problembb1(feature,(PicNo,Pattern2,X2,Y2,Rating2))),
        fail.
remove_duplicates(PicNo).

count_features(PicNo):-
        problembb1(feature,(PicNo,PatternNo,X,Y,Rating)),
        assign_number(PicNo,PatternNo,X,Y),
        retract(problembb1(feature,(PicNo,PatternNo,X,Y,Rating))),
        fail.
count_features(PicNo):-
        retract(feature_number(PicNo,_)).

assign_number(PicNo,PatternNo,X,Y):-
        (       feature_number(PicNo,No),
                FNo is No+1,
                retract(feature_number(PicNo,No)),
                !;
                FNo is 1                        ),
        assert(problembb2(feature,(PicNo,FNo,PatternNo,X,Y))),
        assert(feature_number(PicNo,FNo)).
```

```
/* recognise2/learner_master */
/* ************************************************************** */
/* ****                Learner Master File                   **** */
/* ************************************************************** */

:-[learner_bidder].
:-[learn].
:-[pattern_match].
:-[limits].
```

```
/* recognise2/learner_bidder */
/* ************************************************************** */
/* ****                    Learner Bidder                      **** */
/* ************************************************************** */

wants(learner,problembb4,object_part).

make_bid(learner,[problembb4,object_part,(Object,PicNo)]):-
        recognisebb1(object_names,Names),
        member(Object,Names),
        !.
make_bid(learner,[problembb4,object_part,(Object,PicNo)]):-
        schedule(learner,(Object),90).

execute(learner,(Object)):-
        learn(Object).
```

```
/* recognise2/learn */
/* ************************************************************** */
/* ****      Learning Recognition Rules for an Object          **** */
/* ************************************************************** */

learn(Object):-
        assert(recognisebb4(occurrences,(Object,0))),
        write('Learning: '), write(Object),nl,
        feature_match(Object),
        pattern_match(Object),
        set_limits(Object),
        make_sets(Object),
        recognisebb1(object_names,(List)),
        retract(recognisebb1(object_names,(List))),
        assert(recognisebb1(object_names,([Object|List]))),
        remove_instances(Object),
        !,
        new_entries([]).
```

```
/* recognise2/pattern_match */
/* ************************************************************** */
/* ****       Find matching pattern no./part no. pairs        **** */
/* ************************************************************** */

/* pattern_match(Object) counts the no. of features of each pattern*/
/* which match each object part, and enters the pattern no. and    */
/* part no. in match table if (no. of features)/(no. of matches)<5,*/
/* i.e. if at least 20% of features of the pattern match the part. */

pattern_match(Object):-
        matched_feature(Object,PicNo,FeatureNo,PatNo,Inst,PartNo),
        increment(Object,PatNo,PartNo),
        fail.
pattern_match(Object):-
        find_good_matches(Object),
        fail.
pattern_match(Object):-
        (retractall(feature_count(_,_));true).


increment(Object,PatNo,PartNo):-
        (       feature_count(PatNo,Features),
                NewFeatures is Features+1,
                retract(feature_count(PatNo,Features)), !;
                NewFeatures is 1                        ),
        assert(feature_count(PatNo,NewFeatures)),
        (       PartNo>0,
                (       match_count(Object,PartNo,PatNo,M),
                        NewM is M+1,
                        retract(match_count(Object,PartNo,PatNo,M)),
                        !;
                        NewM is 1       ),
                assert(match_count(Object,PartNo,PatNo,NewM)), !;
                true                                    ),
        !.


find_good_matches(Object):-
        match_count(Object,PartNo,PatNo,M),
        feature_count(PatNo,Features),
        check_probability(Object,PartNo,PatNo,M,Features),
        fail.
find_good_matches(Object).

check_probability(Object,PartNo,PatNo,M,Features):-
        !,
        InvProb is Features/M,
        (       InvProb<10,
                assert(recognisebb2(match,(Object,PartNo,PatNo)));
                true    ),
        retract(match_count(Object,PartNo,PatNo,M)).
```

```
/* recognise2/limits */
/* ********************************************************** */
/* ****              Set Distance Limits                   **** */
/* ********************************************************** */

/* set_limits(Object) finds the maximum and minimum x and y co-ord.*/
/* differences between each pair of Object parts and records then  */
/* in the distance_limits table on recognisebb2.                   */

set_limits(Object):-
        recognisebb1(part_list,(Object,Parts)),
        member((PartNo1,_),Parts),
        member((PartNo2,_),Parts),
        PartNo1 > PartNo2,
        set_part_limits(Object,PartNo1,PartNo2).
set_limits(Object).

set_part_limits(Object,PartNo1,PartNo2):-
        problembb4(object_part,(Object,PicNo,Inst,PartNo1,X1,Y1)),
        problembb4(object_part,(Object,PicNo,Inst,PartNo2,X2,Y2)),
        DiffX is X1-X2,
        DiffY is Y1-Y2,
        update_limits(Object,PartNo1,PartNo2,DiffX,DiffY),
        fail.

update_limits(Object,PartNo1,PartNo2,DiffX,DiffY):-
        !,
        (       recognisebb2(distance_limits,(Object,PartNo1,PartNo2,
                        MinX,MaxX,MinY,MaxY)),
                (       DiffX<MinX, NMinX is DiffX, NMaxX is MaxX, !;
                        NMinX is MinX,
                        (       DiffX>MaxX, NMaxX is DiffX, !;
                                NMaxX is MaxX                   )),
                (       DiffY<MinY, NMinY is DiffY, NMaxY is MaxY, !;
                        NMinY is MinY,
                        (       DiffY>MaxY, NMaxY is DiffY;
                                NMaxY is MaxY                   )),
                retract(recognisebb2(distance_limits,(Object,PartNo1,
                                PartNo2,MinX,MaxX,MinY,MaxY))),
                !;
                NMinX is DiffX, NMaxX is DiffX,
                NMinY is DiffY, NMaxY is DiffY                  ),
        assert(recognisebb2(distance_limits,(Object,PartNo1,PartNo2,
                                NMinX,NMaxX,NMinY,NMaxY))).
```

```
/* recognise2/recogniser_master */
/* ************************************************************** */
/* ****                Recogniser Master File                  **** */
/* ************************************************************** */

:-[recogniser_bidder].
:-[search].
:-[add].
```

```
/* recognise2/recogniser_bidder */
/* ********************************************************************** */
/* ****                    Recogniser Bidder                          **** */
/* ********************************************************************** */

wants(recogniser,problembb2,feature).

make_bid(recogniser,[problembb2,feature,(PicNo)]):-
        check_recognisers(PicNo).

check_recognisers(PicNo):-
        recognisebb1(object_names,(Names)),
        member(Object,Names),
        check_recogniser(Object,PicNo),
        fail.

check_recogniser(Object,PicNo):-
        (       problembb4(object_part,(Object,PicNo,_,_,_,_)), !;
                recognisebb4(occurrences,(Object,N)),
                recognisebb4(occurrences,(total,M)),
                Rating is (40 + (20*N)/M),
                schedule(recogniser,(Object,PicNo),Rating)
        !.

execute(recogniser,(Object,PicNo)):-
        search(Object,PicNo).
```

```
/* recognise2/search */
/* ********************************************************** */
/* ****                  Search a Picture                  **** */
/* ********************************************************** */

search(Object,PicNo):-
        problembb2(identified_features,(PicNo,IdentList)),
        recognisebb2(match,(Object,Part1,Pattern1)),
        problembb2(feature,(PicNo,F1,Pattern1,_,_)),
        nonmember(F1,IdentList),
        recognisebb2(match,(Object,Part2,Pattern2)),
        Part2 > Part1,
        problembb2(feature,(PicNo,F2,Pattern2,_,_)),
        nonmember(F2,IdentList),
        (F1 > F2; F2 > F1),
        write('Searching picture: '),write(PicNo),write(' for: '),
                                write(Object), nl,
        initialise(Object,PicNo),
        search_sets(Object,PicNo),
        !.

search(Object,PicNo).

initialise(Object,PicNo):-
        (       problembb3(probability_list,(PicNo,List)),!;
                assert(problembb3(probability_list,(PicNo,[])))  ),
        assert(current_num(Object,PicNo,0)).

search_sets(Object,PicNo):-
        find_set(Object,PicNo,Parts,Patterns,Features),
        add_to_list(Object,PicNo,Parts,Patterns,Features),
        fail.
search_sets(Object,PicNo):-
        new_entry([problembb3,probability_list,(Object,PicNo)]).
```

```
/* recognise2/add */
/* ************************************************************** */
/* ****      Add a new feature set to probability list        **** */
/* ************************************************************** */

/* add_to_list assigns a number to a newly-formed feature set,     */
/* finds the probability that it represents an object instance,    */
/* then if this probability exceeds the referral threshold,        */
/* removes any subsets of this feature set from the probability    */
/* list and inserts the object/number/probability into the list.   */

add_to_list(Object,PicNo,Parts,Patterns,Features):-
        (       recognisebb3(set_probability,(Object,Parts,Patterns,
                                        Sets,Matches)),
                Prob is (100*Matches)/Sets;
                Prob is 0       ),
        current_num(Object,PicNo,Current),
        Next is Current+1,
        retract(current_num(Object,PicNo,Current)),
        assert(current_num(Object,PicNo,Next)),
        assert(problembb3(feature_set,(Object,PicNo,Next,Parts,
                                        Patterns,Features,nil))),
        (       referral_threshold(Refer),
                Prob >= Refer,
                problembb3(probability_list,(PicNo,List)),
                remove_subsets(Object,PicNo,Parts,Features,List,List2),
                insert((Object,Next,Prob),List2,List3),
                retract(problembb3(probability_list,(PicNo,List))),
                assert(problembb3(probability_list,(PicNo,List3))),
                !;
                true                                            ),
        !.


remove_subsets(_,_,[P1,P2],_,List,List).
remove_subsets(Object,PicNo,Parts,F,[(Object,No,Prob)|Rest1],Rest2):-
        sub_feature_set(Object,PicNo,Parts,F,No),
        remove_subsets(Object,PicNo,Parts,F,Rest1,Rest2).
remove_subsets(Object1,PicNo,Parts,F,[(Object2,No,Prob)|Rest1],
                                        [(Object2,No,Prob)|Rest2]):-
        remove_subsets(Object1,PicNo,Parts,F,Rest1,Rest2).
remove_subsets(Object,PicNo,Parts,F,[],[]).

sub_feature_set(Object,PicNo,Parts1,F1,No):-
        problembb3(feature_set,(Object,PicNo,No,Parts2,_,F2,_)),
        matching_subsets(Parts2,Parts1,F2,F1).

matching_subsets([],P,[],F).
matching_subsets([P1|RestP1],[P1|RestP2],[F1|RestF1],[F1|RestF2]):-
        matching_subsets(RestP1,RestP2,RestF1,RestF2).
matching_subsets(Parts1,[P1|RestP2],Features1,[F1|RestF2]):-
        matching_subsets(Parts1,RestP2,Features1,RestF2).
```

```
/* recognise2/acceptor_master */
/* ************************************************************** */
/* ****                  Acceptor Master File                  **** */
/* ************************************************************** */

:-[acceptor_bidder].
:-[accept].
```

```
/* recognise2/acceptor_bidder */
/* ********************************************************************** */
/* ****                      Acceptor Bidder                          **** */
/* ********************************************************************** */

wants(acceptor,problembb3,probability_list).

make_bid(acceptor,[problembb3,probability_list,(Object,PicNo)]):-
        acceptance_threshold(Accept),
        problembb3(probability_list,(PicNo,[(_,_,Prob)|_])),
        Prob >= Accept,
        schedule(acceptor,(PicNo),70).

execute(acceptor,(PicNo)):-
        accept(PicNo).
```

```
/* recognise2/accept */
/* ********************************************************************** */
/* ****  Accept feature sets with probabilities above threshold ** */
/* ********************************************************************** */

accept(PicNo):-
        problembb3(probability_list,(PicNo,List)),
        accept_list(PicNo,List).

accept_list(PicNo,[(Object,No,Prob)|Rest]):-
        acceptance_threshold(Accept),
        Prob >= Accept,
        check_set(Object,PicNo,No),
        accept_list(PicNo,Rest).

accept_list(_,_).

check_set(Object,PicNo,No):-
        problembb3(feature_set,(Object,PicNo,No,Parts,_,Features,_)),
        problembb2(identified_features,(PicNo,IdentList)),
        member(X,Features),
        member(X,IdentList),
        !.

check_set(Object,PicNo,No):-
        problembb3(feature_set,(Object,PicNo,No,Parts,_,Features,_)),
        background_check(Object,PicNo,Parts,Features),
        write_set(Object,PicNo,Parts,Features),
        write('Accepted.'),nl,nl,
        identify(Object,PicNo,Parts,Features),
        !.

check_set(_,_,_).
```

```
/* recognise2/selector_master */
/* ************************************************************** */
/* ****                Selector Master File                  **** */
/* ************************************************************** */

:-[selector_bidder].
:-[select].
```

```
/* recognise2/selector_bidder */
/* ************************************************************** */
/* ****                    Selector Bidder                       **** */
/* ************************************************************** */

wants(selector,problembb3,probability_list).

make_bid(selector,[problembb3,probability_list,(Object,PicNo)]):-
        schedule(selector,(PicNo),30).

execute(selector,(PicNo)):-
        select(PicNo).
```

```
/* recognise2/select */
/* ********************************************************************** */
/* ****     Select Sets to be Accepted as Object Instances     **** */
/* ********************************************************************** */

/* select(PicNo) lists, in descending order of probability, sets   */
/* of previously unrecognised features in the specified picture    */
/* which could represent known objects.  The user is asked to      */
/* accept or reject each set; accepted sets are recorded as known  */
/* instances of the appropriate object.                            */

select(PicNo):-
        problembb3(probability_list,(PicNo,List)),
        report_list(PicNo,List),
        retract(problembb3(probability_list,(PicNo,List))),
        nl,
        removed_entry([problembb3,probability_list,(PicNo)]).

report_list(PicNo,[(Object,No,Prob)|Rest]):-
        report_set(Object,PicNo,No,Prob),
        report_list(PicNo,Rest).
report_list(_,_).


report_set(Object,PicNo,No,Prob):-
        problembb3(feature_set,(Object,PicNo,No,Parts,_,Features,_)),
        problembb2(identified_features,(PicNo,IdentList)),
        member(X,Features),
        member(X,IdentList),
        !.
report_set(Object,PicNo,No,Prob):-
        problembb3(feature_set,(Object,PicNo,No,Parts,_,Features,_)),
        (       background_check(Object,PicNo,Parts,Features),
                write_set(Object,PicNo,Parts,Features),
                write('Probability:   '),write(Prob),write('%'),nl,
                !,
                ask_user(Object,PicNo,Parts,Features);
                true                                            ).

ask_user(Object,PicNo,Parts,Features):-
        write('Accept (a) or reject (r)?'),
        read_char(Reply,[a,r]),
        (       Reply = a,
                identify(Object,PicNo,Parts,Features),
                !;
                true                        ).
```

```
/* recognise2/feedback_master */
/* ********************************************************************** */
/* ****                  Feedback Master File                        **** */
/* ********************************************************************** */

:-[feedback_bidder].
:-[feedback].
```

```
/* recognise2/feedback_bidder */
/* ********************************************************************** */
/* ****                    Feedback Bidder                           **** */
/* ********************************************************************** */

wants_removed(feedback,problembb3,probability_list).

make_bid(feedback,[problembb3,probability_list,(PicNo)]):-
        check_objects(PicNo).

check_objects(PicNo):-
        recognisebb1(object_names,(Names)),
        member(Object,Names),
        sought(Object,PicNo),
        schedule(feedback,(Object,PicNo),80),
        fail.
check_objects(PicNo).

sought(Object,PicNo):-
        problembb3(feature_set,(Object,PicNo,_,_,_,_,_)),
        !.

execute(feedback,(Object,PicNo)):-
        feedback(Object,PicNo).
```

```
/* recognise2/feedback */
/* ********************************************************************** */
/* Update probabilities using feedback from the recognition process*/
/* ********************************************************************** */


feedback(Object,PicNo):-
        write('Feedback: '),write(Object),
                        write(', Picture '),write(PicNo),
        nl,
        feature_match(Object,PicNo),
        match_sets(Object,PicNo),
        remove_instances(Object,PicNo).

match_sets(Object,PicNo):-
        problembb3(feature_set,(Object,PicNo,SetNo,Parts,Patterns,
                                        Features,nil)),
        check_match(Object,PicNo,Parts,Features,Inst),
        update_probability(Object,Parts,Patterns,Inst),
        retract(problembb3(feature_set,(Object,PicNo,SetNo,Parts,
                                        Patterns,Features,nil))),
        assert(problembb3(feature_set,(Object,PicNo,SetNo,Parts,
                                        Patterns,Features,Inst))),
        fail.
match_sets(Object,PicNo).
```

```
/* recognise2/remover_master */
/* ************************************************************** */
/* ****                Remover Master File                   **** */
/* ************************************************************** */

:-[remover_bidder].
:-[remove].
```

```
/* recognise2/remove */
/* ********************************************************************** */
/* ****                        Remove Data                           **** */
/* ********************************************************************** */

remove(PicNo):-
        write('Removing: '),write(PicNo),nl,
        retractall(problembb2(feature,(PicNo,_,_,_,_))),
        retract(problembb2(identified_features,(PicNo,List))),
        retractall(problembb3(feature_set,(_,PicNo,_,_,_,_,_))),
        !.
```

APPENDIX B

Edited Listings of Test Runs

```
/* ************************************************************ */
/*                      RECOGNISE1 LISTING                      */
/*  Training set: quad pictures 1,2,3,4.   Test: quad picture 5.  */
/* ************************************************************ */

1 % cprolog
C Prolog version 1.5a.ikbs
% Restoring file /usr/lib/prolog/saved_states.d/Prolog1.5a
| ?-[master,quadtest].
.
.
.
yes
| ?-learn(quad).

match(PartNo,PatternNo),Part name:
match(1,1)    left_circle
match(3,1)    right_circle
match(2,2)    square
match(4,3)    triangle

distance_limits(Part1,Part2,MinX,MaxX,MinY,MaxY):
distance_limits(2,1,40,60,0,20)
distance_limits(3,1,20,60,-30,-10)
distance_limits(3,2,-20,0,-40,-10)
distance_limits(4,1,0,30,-80,-40)
distance_limits(4,2,-60,-10,-80,-50)
distance_limits(4,3,-60,-10,-70,-20)

set_probability(PartSet,PatternSet,FeatureSets,Matches):
set_probability([2,1],[2,1],8,5)
set_probability([3,1],[1,1],7,5)
set_probability([4,1],[3,1],14,5)
set_probability([3,2],[1,2],13,5)
set_probability([4,2],[3,2],11,5)
set_probability([4,3],[3,1],10,5)
set_probability([3,2,1],[1,2,1],6,5)
set_probability([4,2,1],[3,2,1],5,5)
set_probability([4,3,1],[3,1,1],6,5)
set_probability([4,3,2],[3,1,2],8,5)
set_probability([4,3,2,1],[3,1,2,1],5,5)

yes
| ?-[quad5].

quad5 consulted 576 bytes 0.31668 sec.

yes
| ?-edit(5).

yes
| ?-search(quad,5,10).

Object:  quad
PartSet, FeatureSet, Probability:
[4,3,2,1],(3,40,30)(1,80,50)(2,90,80)(1,40,70),100%
Accept (a) or reject (r)?
a.
Object:  quad
PartSet, FeatureSet, Probability:
[4,3],(3,30,90)(1,60,110),50%
Accept (a) or reject (r)?
r.
Object:  quad
PartSet, FeatureSet, Probability:
[4,2],(3,10,20)(2,60,90),45.455%
```

```
Accept (a) or reject (r)?
r.
Object:  quad
PartSet, FeatureSet, Probability:
[4,1],(3,60,70)(1,60,110),35.714%
Accept (a) or reject (r)?
r.

yes
| ?-feedback(quad,5).

set_probability(PartSet,PatternSet,FeatureSets,Matches):
set_probability([2,1],[2,1],9,6)
set_probability([3,1],[1,1],8,6)
set_probability([4,1],[3,1],17,6)
set_probability([3,2],[1,2],15,6)
set_probability([4,2],[3,2],14,6)
set_probability([4,3],[3,1],13,6)
set_probability([3,2,1],[1,2,1],7,6)
set_probability([4,2,1],[3,2,1],6,6)
set_probability([4,3,1],[3,1,1],7,6)
set_probability([4,3,2],[3,1,2],10,6)
set_probability([4,3,2,1],[3,1,2,1],6,6)

yes
| ?-
%Prolog execution halted
```

```
/* ************************************************************ */
/*                      RECOGNISE1 LISTING                      */
/*  Training set: car images 2,3,4,5,6.  Test: car image 1.     */
/* ************************************************************ */

/* Shows the effect of using feedback with incomplete part set.    */
/* Some of the entries in the set_probability table are made more  */
/* accurate, others less so.                                       */

1 % cprolog
C Prolog version 1.5a.ikbs
% Restoring file /usr/lib/prolog/saved_states.d/Prolog1.5a
| ?-[master,ctest1].
.
.
.
.
cargen consulted 372 bytes 0.18334 sec.
cf2 consulted 432 bytes 0.25 sec.
cp2 consulted 260 bytes 0.15 sec.
cf3 consulted 288 bytes 0.18333 sec.
cp3 consulted 260 bytes 0.16667 sec.
cf4 consulted 384 bytes 0.23334 sec.
cp4 consulted 260 bytes 0.15 sec.
cf5 consulted 384 bytes 0.21667 sec.
cp5 consulted 260 bytes 0.15 sec.
cf6 consulted 384 bytes 0.21667 sec.
cp6 consulted 260 bytes 0.15 sec.
ctest1 consulted 3700 bytes 2.5 sec.

yes
| ?-edit.

yes
| ?-learn(car).

match(PartNo,PatternNo),Part name:
match(5,6)    top
match(1,1)    rear_wheel
match(2,1)    front_wheel
match(1,3)    rear_wheel
match(2,3)    front_wheel
match(5,5)    top
match(1,2)    rear_wheel
match(2,2)    front_wheel
match(3,4)    rear_arch
match(4,4)    front_arch

distance_limits(Part1,Part2,MinX,MaxX,MinY,MaxY):
distance_limits(2,1,123,144,-3,6)
distance_limits(3,1,-1,2,-18,-8)
distance_limits(3,2,-145,-121,-24,-9)
distance_limits(4,1,125,143,-16,-8)
distance_limits(4,2,-1,2,-17,-9)
distance_limits(4,3,123,144,-2,7)
distance_limits(5,1,85,97,-51,-43)
distance_limits(5,2,-50,-36,-51,-42)
distance_limits(5,3,83,95,-36,-27)
distance_limits(5,4,-49,-35,-36,-29)

set_probability(PartSet,PatternSet,FeatureSets,Matches):
set_probability([2,1],[1,1],2,1)
set_probability([2,1],[1,3],1,0)
set_probability([2,1],[3,3],1,1)
set_probability([2,1],[2,3],1,0)
set_probability([2,1],[2,2],3,3)
```

```
set_probability([3,1],[4,3],1,0)
set_probability([3,1],[4,1],2,1)
set_probability([3,1],[4,2],6,3)
set_probability([4,1],[4,3],1,0)
set_probability([4,1],[4,1],1,1)
set_probability([4,1],[4,2],3,3)
set_probability([5,1],[6,3],1,0)
set_probability([5,1],[6,2],1,1)
set_probability([5,1],[5,1],4,1)
set_probability([5,1],[5,3],2,1)
set_probability([5,1],[5,2],2,1)
set_probability([3,2],[4,1],1,1)
set_probability([3,2],[4,2],3,3)
set_probability([4,2],[4,3],1,0)
set_probability([4,2],[4,1],2,1)
set_probability([4,2],[4,2],5,3)
set_probability([5,2],[6,2],1,1)
set_probability([5,2],[5,1],3,1)
set_probability([5,2],[5,3],2,1)
set_probability([5,2],[5,2],1,1)
set_probability([4,3],[4,4],4,4)
set_probability([5,3],[6,4],1,1)
set_probability([5,3],[5,4],3,2)
set_probability([5,4],[6,4],1,1)
set_probability([5,4],[5,4],2,2)
set_probability([3,2,1],[4,1,1],1,1)
set_probability([4,2,1],[4,1,1],1,1)
set_probability([5,2,1],[5,1,1],3,1)
set_probability([5,2,1],[5,1,3],1,0)
set_probability([5,2,1],[5,3,3],1,1)
set_probability([3,2,1],[4,2,3],1,0)
set_probability([4,2,1],[4,2,3],1,0)
set_probability([5,2,1],[6,2,3],1,0)
set_probability([5,2,1],[6,2,2],1,1)
set_probability([5,2,1],[5,2,2],1,1)
set_probability([3,2,1],[4,2,2],3,3)
set_probability([4,2,1],[4,2,2],3,3)
set_probability([4,3,1],[4,4,3],1,0)
set_probability([5,3,1],[6,4,3],1,0)
set_probability([5,3,1],[6,4,2],1,1)
set_probability([4,3,1],[4,4,1],1,1)
set_probability([5,3,1],[5,4,1],1,1)
set_probability([5,3,1],[5,4,2],2,1)
set_probability([4,3,1],[4,4,2],3,3)
set_probability([5,4,1],[6,4,3],1,0)
set_probability([5,4,1],[6,4,2],1,1)
set_probability([5,4,1],[5,4,1],1,1)
set_probability([5,4,1],[5,4,2],1,1)
set_probability([5,3,2],[6,4,2],1,1)
set_probability([4,3,2],[4,4,1],1,1)
set_probability([5,3,2],[5,4,1],1,1)
set_probability([5,3,2],[5,4,2],1,1)
set_probability([4,3,2],[4,4,2],3,3)
set_probability([5,4,2],[6,4,2],1,1)
set_probability([5,4,2],[5,4,1],1,1)
set_probability([5,4,2],[5,4,2],1,1)
set_probability([5,4,3],[6,4,4],1,1)
set_probability([5,4,3],[5,4,4],2,2)
set_probability([4,3,2,1],[4,4,1,1],1,1)
set_probability([5,3,2,1],[5,4,1,1],1,1)
set_probability([5,4,2,1],[5,4,1,1],1,1)
set_probability([4,3,2,1],[4,4,2,3],1,0)
set_probability([5,3,2,1],[6,4,2,3],1,0)
set_probability([5,4,2,1],[6,4,2,3],1,0)
set_probability([5,3,2,1],[6,4,2,2],1,1)
set_probability([5,4,2,1],[6,4,2,2],1,1)
```

```
set_probability([5,3,2,1],[5,4,2,2],1,1)
set_probability([5,4,2,1],[5,4,2,2],1,1)
set_probability([4,3,2,1],[4,4,2,2],3,3)
set_probability([5,4,3,1],[6,4,4,3],1,0)
set_probability([5,4,3,1],[6,4,4,2],1,1)
set_probability([5,4,3,1],[5,4,4,1],1,1)
set_probability([5,4,3,1],[5,4,4,2],1,1)
set_probability([5,4,3,2],[6,4,4,2],1,1)
set_probability([5,4,3,2],[5,4,4,1],1,1)
set_probability([5,4,3,2],[5,4,4,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,1,1],1,1)
set_probability([5,4,3,2,1],[6,4,4,2,3],1,0)
set_probability([5,4,3,2,1],[6,4,4,2,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,2,2],1,1)

yes
| ?-[cf1].

cf1 consulted 384 bytes 0.21668 sec.

yes
| ?-edit(1).

yes
| ?-search(car,1,0).

Object:  car
PartSet, FeatureSet, Probability:
[4,2],(4,69,140)(2,69,154),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,1],(4,69,140)(2,69,154),50%
Accept (a) or reject (r)?
a.

yes
| ?-feedback(car,1).

set_probability(PartSet,PatternSet,FeatureSets,Matches):
set_probability([2,1],[1,1],2,1)
set_probability([2,1],[1,3],1,0)
set_probability([2,1],[3,3],1,1)
set_probability([2,1],[2,3],1,0)
set_probability([2,1],[2,2],3,3)
set_probability([3,1],[4,3],1,0)
set_probability([3,1],[4,1],2,1)
set_probability([4,1],[4,3],1,0)
set_probability([4,1],[4,1],1,1)
set_probability([4,1],[4,2],3,3)
set_probability([5,1],[6,3],1,0)
set_probability([5,1],[6,2],1,1)
set_probability([5,1],[5,1],4,1)
set_probability([5,1],[5,3],2,1)
set_probability([5,1],[5,2],2,1)
set_probability([3,2],[4,1],1,1)
set_probability([3,2],[4,2],3,3)
set_probability([4,2],[4,3],1,0)
set_probability([4,2],[4,1],2,1)
set_probability([5,2],[6,2],1,1)
set_probability([5,2],[5,1],3,1)
set_probability([5,2],[5,3],2,1)
set_probability([5,2],[5,2],1,1)
set_probability([4,3],[4,4],4,4)
set_probability([5,3],[6,4],1,1)
```

```
set_probability([5,3],[5,4],3,2)
set_probability([5,4],[6,4],1,1)
set_probability([5,4],[5,4],2,2)
set_probability([3,2,1],[4,1,1],1,1)
set_probability([4,2,1],[4,1,1],1,1)
set_probability([5,2,1],[5,1,1],3,1)
set_probability([5,2,1],[5,1,3],1,0)
set_probability([5,2,1],[5,3,3],1,1)
set_probability([3,2,1],[4,2,3],1,0)
set_probability([4,2,1],[4,2,3],1,0)
set_probability([5,2,1],[6,2,3],1,0)
set_probability([5,2,1],[6,2,2],1,1)
set_probability([5,2,1],[5,2,2],1,1)
set_probability([3,2,1],[4,2,2],3,3)
set_probability([4,2,1],[4,2,2],3,3)
set_probability([4,3,1],[4,4,3],1,0)
set_probability([5,3,1],[6,4,3],1,0)
set_probability([5,3,1],[6,4,2],1,1)
set_probability([4,3,1],[4,4,1],1,1)
set_probability([5,3,1],[5,4,1],1,1)
set_probability([5,3,1],[5,4,2],2,1)
set_probability([4,3,1],[4,4,2],3,3)
set_probability([5,4,1],[6,4,3],1,0)
set_probability([5,4,1],[6,4,2],1,1)
set_probability([5,4,1],[5,4,1],1,1)
set_probability([5,4,1],[5,4,2],1,1)
set_probability([5,3,2],[6,4,2],1,1)
set_probability([4,3,2],[4,4,1],1,1)
set_probability([5,3,2],[5,4,1],1,1)
set_probability([5,3,2],[5,4,2],1,1)
set_probability([4,3,2],[4,4,2],3,3)
set_probability([5,4,2],[6,4,2],1,1)
set_probability([5,4,2],[5,4,1],1,1)
set_probability([5,4,2],[5,4,2],1,1)
set_probability([5,4,3],[6,4,4],1,1)
set_probability([5,4,3],[5,4,4],2,2)
set_probability([4,3,2,1],[4,4,1,1],1,1)
set_probability([5,3,2,1],[5,4,1,1],1,1)
set_probability([5,4,2,1],[5,4,1,1],1,1)
set_probability([4,3,2,1],[4,4,2,3],1,0)
set_probability([5,3,2,1],[6,4,2,3],1,0)
set_probability([5,4,2,1],[6,4,2,3],1,0)
set_probability([5,3,2,1],[6,4,2,2],1,1)
set_probability([5,4,2,1],[6,4,2,2],1,1)
set_probability([5,3,2,1],[5,4,2,2],1,1)
set_probability([5,4,2,1],[5,4,2,2],1,1)
set_probability([4,3,2,1],[4,4,2,2],3,3)
set_probability([5,4,3,1],[6,4,4,3],1,0)
set_probability([5,4,3,1],[6,4,4,2],1,1)
set_probability([5,4,3,1],[5,4,4,1],1,1)
set_probability([5,4,3,1],[5,4,4,2],1,1)
set_probability([5,4,3,2],[6,4,4,2],1,1)
set_probability([5,4,3,2],[5,4,4,1],1,1)
set_probability([5,4,3,2],[5,4,4,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,1,1],1,1)
set_probability([5,4,3,2,1],[6,4,4,2,3],1,0)
set_probability([5,4,3,2,1],[6,4,4,2,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,2,2],1,1)
set_probability([3,1],[4,2],7,4)
set_probability([4,2],[4,2],6,3)

yes
| ?-
% Prolog execution halted
```

```
/* ************************************************************ */
/*                        RECOGNISE1 LISTING                    */
/*  Training set: car images 1,3,4,5,6.  Test: car image 2.     */
/* ************************************************************ */

/* Shows successful recognition with just five training examples   */
.
.
.
| ?-learn(car).

match(PartNo,PatternNo),Part name:
match(5,6)     top
match(1,1)     rear_wheel
match(2,1)     front_wheel
match(1,3)     rear_wheel
match(2,3)     front_wheel
match(5,5)     top
match(1,2)     rear_wheel
match(2,2)     front_wheel
match(3,4)     rear_arch
match(4,4)     front_arch

distance_limits(Part1,Part2,MinX,MaxX,MinY,MaxY):
distance_limits(2,1,123,148,-8,6)
distance_limits(3,1,-1,2,-18,-8)
distance_limits(3,2,-148,-121,-24,-6)
distance_limits(4,1,125,151,-24,-8)
distance_limits(4,2,-1,3,-17,-9)
distance_limits(4,3,123,151,-10,7)
distance_limits(5,1,85,97,-57,-43)
distance_limits(5,2,-54,-36,-51,-42)
distance_limits(5,3,83,95,-43,-27)
distance_limits(5,4,-57,-35,-36,-29)

set_probability(PartSet,PatternSet,FeatureSets, Matches):
set_probability([2,1],[1,1],2,1)
set_probability([2,1],[1,3],1,0)
set_probability([2,1],[3,1],1,0)
set_probability([2,1],[3,3],1,1)
set_probability([2,1],[2,2],3,3)
set_probability([3,1],[4,1],2,1)
set_probability([3,1],[4,2],5,3)
set_probability([4,1],[4,1],1,1)
set_probability([4,1],[4,2],3,3)
set_probability([5,1],[6,2],1,1)
set_probability([5,1],[5,1],4,1)
set_probability([5,1],[5,3],2,1)
set_probability([5,1],[5,2],2,1)
set_probability([3,2],[4,1],1,1)
set_probability([3,2],[4,2],3,3)
set_probability([4,2],[4,1],2,1)
set_probability([4,2],[4,2],5,3)
set_probability([5,2],[6,2],1,1)
set_probability([5,2],[5,1],3,1)
set_probability([5,2],[5,3],2,1)
set_probability([5,2],[5,2],1,1)
set_probability([4,3],[4,4],4,4)
set_probability([5,3],[6,4],1,1)
set_probability([5,3],[5,4],3,2)
set_probability([5,4],[6,4],1,1)
set_probability([5,4],[5,4],2,2)
set_probability([3,2,1],[4,1,1],1,1)
set_probability([4,2,1],[4,1,1],1,1)
set_probability([5,2,1],[5,1,1],3,1)
set_probability([5,2,1],[5,1,3],1,0)
```

```
set_probability([5,2,1],[5,3,1],2,0)
set_probability([5,2,1],[5,3,3],1,1)
set_probability([5,2,1],[6,2,2],1,1)
set_probability([5,2,1],[5,2,2],1,1)
set_probability([3,2,1],[4,2,2],3,3)
set_probability([4,2,1],[4,2,2],3,3)
set_probability([5,3,1],[6,4,2],1,1)
set_probability([4,3,1],[4,4,1],1,1)
set_probability([5,3,1],[5,4,1],1,1)
set_probability([5,3,1],[5,4,2],2,1)
set_probability([4,3,1],[4,4,2],3,3)
set_probability([5,4,1],[6,4,2],1,1)
set_probability([5,4,1],[5,4,1],1,1)
set_probability([5,4,1],[5,4,2],1,1)
set_probability([5,3,2],[6,4,2],1,1)
set_probability([4,3,2],[4,4,1],1,1)
set_probability([5,3,2],[5,4,1],1,1)
set_probability([5,3,2],[5,4,2],1,1)
set_probability([4,3,2],[4,4,2],3,3)
set_probability([5,4,2],[6,4,2],1,1)
set_probability([5,4,2],[5,4,1],1,1)
set_probability([5,4,2],[5,4,2],1,1)
set_probability([5,4,3],[6,4,4],1,1)
set_probability([5,4,3],[5,4,4],2,2)
set_probability([4,3,2,1],[4,4,1,1],1,1)
set_probability([5,3,2,1],[5,4,1,1],1,1)
set_probability([5,4,2,1],[5,4,1,1],1,1)
set_probability([5,3,2,1],[6,4,2,2],1,1)
set_probability([5,4,2,1],[6,4,2,2],1,1)
set_probability([5,3,2,1],[5,4,2,2],1,1)
set_probability([5,4,2,1],[5,4,2,2],1,1)
set_probability([4,3,2,1],[4,4,2,2],3,3)
set_probability([5,4,3,1],[6,4,4,2],1,1)
set_probability([5,4,3,1],[5,4,4,1],1,1)
set_probability([5,4,3,1],[5,4,4,2],1,1)
set_probability([5,4,3,2],[6,4,4,2],1,1)
set_probability([5,4,3,2],[5,4,4,1],1,1)
set_probability([5,4,3,2],[5,4,4,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,1,1],1,1)
set_probability([5,4,3,2,1],[6,4,4,2,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,2,2],1,1)

yes
.
.
| ?-search(car,2,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2,1],(6,155,119)(4,197,154)(4,60,153)(2,197,168)(2,60,170),100%
Accept (a) or reject (r)?
a.

yes
| ?-feedback(car,2).

set_probability(PartSet,PatternSet,FeatureSets, Matches):
set_probability([2,1],[1,1],2,1)
set_probability([2,1],[1,3],1,0)
set_probability([2,1],[3,1],1,0)
set_probability([2,1],[3,3],1,1)
set_probability([3,1],[4,1],2,1)
set_probability([4,1],[4,1],1,1)
set_probability([5,1],[5,1],4,1)
set_probability([5,1],[5,3],2,1)
set_probability([5,1],[5,2],2,1)
```

```
set_probability([3,2],[4,1],1,1)
set_probability([4,2],[4,1],2,1)
set_probability([5,2],[5,1],3,1)
set_probability([5,2],[5,3],2,1)
set_probability([5,2],[5,2],1,1)
set_probability([5,3],[5,4],3,2)
set_probability([5,4],[5,4],2,2)
set_probability([3,2,1],[4,1,1],1,1)
set_probability([4,2,1],[4,1,1],1,1)
set_probability([5,2,1],[5,1,1],3,1)
set_probability([5,2,1],[5,1,3],1,0)
set_probability([5,2,1],[5,3,1],2,0)
set_probability([5,2,1],[5,3,3],1,1)
set_probability([5,2,1],[5,2,2],1,1)
set_probability([4,3,1],[4,4,1],1,1)
set_probability([5,3,1],[5,4,1],1,1)
set_probability([5,3,1],[5,4,2],2,1)
set_probability([5,4,1],[5,4,1],1,1)
set_probability([5,4,1],[5,4,2],1,1)
set_probability([4,3,2],[4,4,1],1,1)
set_probability([5,3,2],[5,4,1],1,1)
set_probability([5,3,2],[5,4,2],1,1)
set_probability([5,4,2],[5,4,1],1,1)
set_probability([5,4,2],[5,4,2],1,1)
set_probability([5,4,3],[5,4,4],2,2)
set_probability([4,3,2,1],[4,4,1,1],1,1)
set_probability([5,3,2,1],[5,4,1,1],1,1)
set_probability([5,4,2,1],[5,4,1,1],1,1)
set_probability([5,3,2,1],[5,4,2,2],1,1)
set_probability([5,4,2,1],[5,4,2,2],1,1)
set_probability([5,4,3,1],[5,4,4,1],1,1)
set_probability([5,4,3,1],[5,4,4,2],1,1)
set_probability([5,4,3,2],[5,4,4,1],1,1)
set_probability([5,4,3,2],[5,4,4,2],1,1)
set_probability([5,4,3,2,1],[5,4,4,1,1],1,1)
set_probability([5,4,3,2,1],[5,4,4,2,2],1,1)
set_probability([2,1],[2,3],1,0)
set_probability([2,1],[2,2],4,4)
set_probability([3,1],[4,3],1,0)
set_probability([3,1],[4,2],7,4)
set_probability([4,1],[4,3],1,0)
set_probability([4,1],[4,2],4,4)
set_probability([5,1],[6,3],1,0)
set_probability([5,1],[6,2],2,2)
set_probability([3,2],[4,2],4,4)
set_probability([4,2],[4,3],1,0)
set_probability([4,2],[4,2],7,4)
set_probability([5,2],[6,2],2,2)
set_probability([4,3],[4,4],5,5)
set_probability([5,3],[6,4],2,2)
set_probability([5,4],[6,4],2,2)
set_probability([3,2,1],[4,2,3],1,0)
set_probability([4,2,1],[4,2,3],1,0)
set_probability([5,2,1],[6,2,3],1,0)
set_probability([3,2,1],[4,2,2],4,4)
set_probability([4,2,1],[4,2,2],4,4)
set_probability([5,2,1],[6,2,2],2,2)
set_probability([4,3,1],[4,4,3],1,0)
set_probability([5,3,1],[6,4,3],1,0)
set_probability([4,3,1],[4,4,2],4,4)
set_probability([5,3,1],[6,4,2],2,2)
set_probability([5,4,1],[6,4,3],1,0)
set_probability([5,4,1],[6,4,2],2,2)
set_probability([4,3,2],[4,4,2],4,4)
set_probability([5,3,2],[6,4,2],2,2)
set_probability([5,4,2],[6,4,2],2,2)
```

```
set_probability([5,4,3],[6,4,4],2,2)
set_probability([4,3,2,1],[4,4,2,3],1,0)
set_probability([5,3,2,1],[6,4,2,3],1,0)
set_probability([5,4,2,1],[6,4,2,3],1,0)
set_probability([4,3,2,1],[4,4,2,2],4,4)
set_probability([5,3,2,1],[6,4,2,2],2,2)
set_probability([5,4,2,1],[6,4,2,2],2,2)
set_probability([5,4,3,1],[6,4,4,3],1,0)
set_probability([5,4,3,1],[6,4,4,2],2,2)
set_probability([5,4,3,2],[6,4,4,2],2,2)
set_probability([5,4,3,2,1],[6,4,4,2,3],1,0)
set_probability([5,4,3,2,1],[6,4,4,2,2],2,2)

yes
| ?-
% Prolog execution halted
```

```
/* Shows the effect of extending the definition of duplicates      */

 /* ************************************************************** */
 /*                    RECOGNISE1 LISTING                          */
 /* Training set: all 21 car images.  Test: car images 4,14,16,17,20*/
 /* ************************************************************** */
 .
 .
 .

 | ?-search(car,4,0).

 Object:  car
        PartSet, FeatureSet, Probability:
[2,1],(1,200,172)(1,70,170),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(5,163,122)(1,200,172),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(5,163,122)(3,199,166),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(5,160,124)(1,200,172),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(5,160,124)(3,199,166),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(5,167,120)(1,70,170),57.143%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(5,163,122)(1,70,170),57.143%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(5,160,124)(1,70,170),57.143%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,163,122)(1,200,172)(1,70,170),50%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,160,124)(1,200,172)(1,70,170),50%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(3,199,166)(1,70,170),50%
Accept (a) or reject (r)?
```

```
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(3,199,166)(3,70,166),50%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,163,122)(3,199,166)(1,70,170),33.333%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,160,124)(3,199,166)(1,70,170),33.333%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,163,122)(3,199,166)(3,70,166),25%
Accept (a) or reject (r)?
a.

yes
.
.

| ?-search(car,14,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(6,160,115)(2,215,166)(2,53,158),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(2,215,166)(2,53,158),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,1],(4,214,147)(2,53,158),91.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,1],(4,216,151)(2,53,158),91.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,2,1],(4,214,147)(2,215,166)(2,53,158),88.889%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,2,1],(4,216,151)(2,215,166)(2,53,158),88.889%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2],(4,53,141)(2,215,166),88.889%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2],(4,52,144)(2,215,166),88.889%
```

```
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2,1],(4,53,141)(2,215,166)(2,53,158),87.5%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2,1],(4,52,144)(2,215,166)(2,53,158),87.5%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(6,160,115)(2,215,166),87.5%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3],(4,214,147)(4,53,141),85.714%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3],(4,214,147)(4,52,144),85.714%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3],(4,216,151)(4,52,144),85.714%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2],(6,160,115)(4,53,141)(2,215,166),83.333%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2],(6,160,115)(4,52,144)(2,215,166),83.333%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,1],(4,214,147)(4,53,141)(2,53,158),81.818%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,1],(4,214,147)(4,52,144)(2,53,158),81.818%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,1],(4,216,151)(4,52,144)(2,53,158),81.818%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,2,1],(6,160,115)(4,214,147)(2,215,166)(2,53,158),80%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,2,1],(6,160,115)(4,216,151)(2,215,166)(2,53,158),80%
Accept (a) or reject (r)?
```

```
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2,1],(6,160,115)(4,53,141)(2,215,166)(2,53,158),80%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2,1],(6,160,115)(4,52,144)(2,215,166)(2,53,158),80%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2],(4,214,147)(4,53,141)(2,215,166),77.778%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2],(4,214,147)(4,52,144)(2,215,166),77.778%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2],(4,216,151)(4,52,144)(2,215,166),77.778%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,160,115)(2,53,158),77.778%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,155,115)(2,53,158),77.778%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2,1],(4,214,147)(4,53,141)(2,215,166)(2,53,158),75%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2,1],(4,214,147)(4,52,144)(2,215,166)(2,53,158),75%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,3,2,1],(4,216,151)(4,52,144)(2,215,166)(2,53,158),75%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,2],(6,160,115)(4,214,147)(2,215,166),75%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,2],(6,160,115)(4,216,151)(2,215,166),75%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4],(6,160,115)(4,214,147),72.727%
Accept (a) or reject (r)?
r.
```

```
Object:  car
PartSet, FeatureSet, Probability:
[5,4],(6,160,115)(4,216,151),72.727%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,1],(6,160,115)(4,214,147)(2,53,158),71.429%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,1],(6,160,115)(4,216,151)(2,53,158),71.429%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2],(6,160,115)(4,214,147)(4,53,141)(2,215,166),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2],(6,160,115)(4,214,147)(4,52,144)(2,215,166),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2],(6,160,115)(4,216,151)(4,52,144)(2,215,166),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3],(6,160,115)(4,214,147)(4,53,141),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3],(6,160,115)(4,214,147)(4,52,144),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3],(6,160,115)(4,216,151)(4,52,144),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3],(6,160,115)(4,53,141),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3],(6,160,115)(4,52,144),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3],(6,155,115)(4,53,141),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3],(6,155,115)(4,52,144),66.667%
Accept (a) or reject (r)?
r.
Object:  car
```

```
PartSet, FeatureSet, Probability:
[5,4,3,2,1],(6,160,115)(4,214,147)(4,53,141)(2,215,166)(2,53,158),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2,1],(6,160,115)(4,214,147)(4,52,144)(2,215,166)(2,53,158),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2,1],(6,160,115)(4,216,151)(4,52,144)(2,215,166)(2,53,158),60%
Accept (a) or reject (r)?
a.

yes
.
.
.
| ?-search(car,16,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,3,2,1],(5,164,116)(4,94,145)(1,200,156)(1,94,164),100%
Accept (a) or reject (r)?
a.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,163,113)(3,94,156),0%
Accept (a) or reject (r)?
r.

yes
.
.
.
| ?-search(car,17,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,1],(5,140,108)(1,75,138),57.143%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,158,103)(1,75,138),33.333%
Accept (a) or reject (r)?
a.

yes
.
.
| ?-search(car,20,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(6,156,123)(2,196,167)(2,65,160),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(2,196,167)(2,65,160),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
```

```
[3,2],(4,65,146)(2,196,167),88.889%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2,1],(4,65,146)(2,196,167)(2,65,160),87.5%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(6,156,123)(2,196,167),87.5%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2],(6,156,123)(4,65,146)(2,196,167),83.333%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,3,2,1],(6,156,123)(4,65,146)(2,196,167)(2,65,160),80%
Accept (a) or reject (r)?
a.

yes
| ?-
% Prolog execution halted


/* ************************************************************** */
/*   RECOGNISE1 LISTING WITH DEFINITION OF DUPLICATES EXTENDED    */
/*   Training set, test as before                                 */
/* ************************************************************** */
.
.
.

| ?-search(car,4,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(5,163,122)(3,199,166)(3,70,166),100%
Accept (a) or reject (r)?
a.

yes
.
.
| ?-search(car,14,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,4,3,2,1],(6,160,115)(4,216,151)(4,52,144)(2,215,166)(2,53,158),100%
Accept (a) or reject (r)?
a.

yes
.
.
| ?-search(car,16,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(6,163,113)(1,200,156)(1,94,164),100%
Accept (a) or reject (r)?
r.
```

```
Object:  car
PartSet, FeatureSet, Probability:
[3,2,1],(4,94,145)(1,200,156)(1,94,164),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2],(6,163,113)(1,200,156),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2],(4,94,145)(1,200,156),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(1,200,156)(1,94,164),100%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,163,113)(1,94,164),66.667%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,2],(4,94,145)(1,94,164),60%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[4,2],(4,94,145)(3,94,156),50%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,1],(4,94,145)(1,94,164),50%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,2,1],(6,163,113)(1,200,156)(3,94,156),0%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,2,1],(4,94,145)(1,200,156)(3,94,156),0%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[5,1],(6,163,113)(3,94,156),0%
Accept (a) or  reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[3,1],(4,94,145)(3,94,156),0%
Accept (a) or reject (r)?
r.
Object:  car
PartSet, FeatureSet, Probability:
[2,1],(1,200,156)(3,94,156),0%
Accept (a) or reject (r)?
r.
```

```
yes
.
.
| ?-search(car,17,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,1],(5,140,108)(1,75,138),100%
Accept (a) or reject (r)?
a.

yes
.
.
| ?-search(car,20,0).

Object:  car
PartSet, FeatureSet, Probability:
[5,3,2,1],(6,156,123)(4,65,146)(2,196,167)(2,65,160),100%
Accept (a) or reject (r)?
a.

yes
| ?-
% Prolog execution halted
```

```
/* recognise2/shapescheck */
/* ************************************************************************
/* ****                    Background Checks on Shapes
/* ************************************************************************

/* No checks on rectangles.

background_check(rectangle1,PicNo,_,_):- !.
background_check(rectangle2,PicNo,_,_):- !.

/* Checks that all other potential shapes have two adjacent sides of
/* equal length.

background_check(big_square,PicNo,[3,1],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(big_square,PicNo,[4,2],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(little_square,PicNo,[3,1],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(little_square,PicNo,[4,2],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(triangle1,PicNo,[3,2],[F1,F1]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(triangle2,PicNo,[3,2],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(triangle3,PicNo,[3,2],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).
background_check(triangle4,PicNo,[3,2],[F1,F2]):-
        !,
        opposite_corners(PicNo,[F1,F2]).

background_check(_,PicNo,_,[F1,F2]):- !.

background_check(_,PicNo,_,[F1,F2|Rest]):-
        member(F3,Rest),
        !,
        equal_sides(PicNo,[F1,F2,F3]).

equal_sides(PicNo,[F1,F2,F3]):-
        problembb2(feature,(PicNo,F1,_,X1,Y1)),
        problembb2(feature,(PicNo,F2,_,X2,Y2)),
        problembb2(feature,(PicNo,F3,_,X3,Y3)),
        L1 is (X1 + Y1 - X3 - Y3),
        L2 is (X2 + Y2 - X1 - Y1),
        L3 is (X3 + Y3 - X2 - Y2),
        (L1 > 0, D1 is L1; D1 is -L1),
        (L2 > 0, D2 is L2; D2 is -L2),
        (L3 > 0, D3 is L3; D3 is -L3),
        equal_length(D1,D2,D3).

opposite_corners(PicNo,[F1,F2]):-
        problembb2(feature,(PicNo,F1,_,X1,Y1)),
        problembb2(feature,(PicNo,F2,_,X2,Y2)),
        (X1 > X2, L1 is X1 - X2; L1 is X2 - X1),
        (Y1 > Y2, L2 is Y1 - Y2; L2 is Y2 - Y1),
        L1 = L2.


equal_length(L1,L2,L3):-
```

```
        L1 = L2, !.
equal_length(L1,L2,L3):-
        L2 = L3, !.
equal_length(L1,L2,L3):-
        L3 = L1, !.
```

```
/* ************************************************************** *
/* ****                  RECOGNISE2 LISTING                      **** *
/* ****  Rectangles training data loaded after first set of tests * *
/* ************************************************************** *

21 % prolog
C Prolog version 1.5a.ikbs
% Restoring file /usr/lib/prolog/saved_states.d/Prolog1.5a
| ?- [master].

. . . . . .

yes
| ?- init.

Duplicates limit? 0.
Percentage prob. threshold for automatic acceptance?
(For no automatic acceptance, enter 110)  110.
Percentage prob. threshold for referral to user? 0.
Bid execution manual (m) or automatic (a)? a.

What now?
(Enter "help." to view options)
|: [test].

shapescheck consulted 2268 bytes 0.11667 sec.
shapesgen consulted 984 bytes 0.083334 sec.
shapes1 consulted 2000 bytes 0.23333 sec.
shapes2 consulted 2000 bytes 0.23333 sec.
test consulted 7252 bytes 0.73333 sec.
Editing picture:2
Editing picture:1
Learning: triangle4
Learning: triangle3
Learning: triangle2
Learning: triangle1
Learning: little_square
Learning: big_square
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes3].

shapes3 consulted 1008 bytes 0.1 sec.
Editing picture:3
Searching picture: 3 for: triangle4
Searching picture: 3 for: triangle2
Searching picture: 3 for: triangle1
Searching picture: 3 for: little_square
Searching picture: 3 for: big_square

Object:   big_square
bottom_left,(10,50)
bottom_right,(50,50)
top_right,(50,10)
top_left,(10,10)
Probability:   100%
Accept (a) or reject (r)?a.
```

```
Object:   big_square
bottom_right,(110,70)
top_right,(110,20)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   little_square
bottom_left,(90,70)
bottom_right,(110,70)
top_right,(110,50)
top_left,(90,50)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   little_square
bottom_left,(30,70)
bottom_right,(50,70)
top_right,(50,50)
top_left,(30,50)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   little_square
bottom_right,(110,50)
top_right,(110,20)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   triangle1
bottom,(50,70)
right,(90,30)
right_angle,(50,30)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   triangle2
left,(80,20)
bottom,(110,50)
right_angle,(110,20)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: triangle2, Picture 3
Feedback: triangle1, Picture 3
Feedback: little_square, Picture 3
Feedback: big_square, Picture 3
Removing: 3
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes4].

shapes4 consulted 1056 bytes 0.11667 sec.
Editing picture:4
Searching picture: 4 for: little_square
Searching picture: 4 for: triangle1
Searching picture: 4 for: big_square
Searching picture: 4 for: triangle3
```

```
Object:   triangle3
top,(115,70)
left,(95,90)
right_angle,(115,90)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   big_square
bottom_left,(55,80)
bottom_right,(115,80)
top_right,(115,20)
top_left,(55,20)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   big_square
top_right,(55,40)
top_left,(5,40)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   triangle1
bottom,(35,70)
right,(65,40)
right_angle,(35,40)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   little_square
bottom_left,(5,70)
bottom_right,(35,70)
top_right,(35,40)
top_left,(5,40)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: triangle3, Picture 4
Feedback: triangle1, Picture 4
Feedback: little_square, Picture 4
Feedback: big_square, Picture 4
Removing: 4
Bid list empty

What now?
(Enter "help." to view options)
|: show_rules(big_square).


Part/Pattern No. matches:
top_left, Pattern1
top_right, Pattern2
bottom_right, Pattern3
bottom_left, Pattern4

Distance limits(Part1,Part2,MinX,MaxX,MinY,MaxY):
top_right,top_left,40,60,0,0
bottom_right,top_left,40,60,40,60
bottom_right,top_right,0,0,40,60
bottom_left,top_left,0,0,40,60
bottom_left,top_right,-60,-40,40,60
```

```
bottom_left,bottom_right,-60,-40,0,0

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],5,4
[3,1],[3,1],9,4
[4,1],[4,1],5,4
[3,2],[3,2],6,4
[4,2],[4,2],5,4
[4,3],[4,3],4,4
[3,2,1],[3,2,1],5,4
[4,2,1],[4,2,1],4,4
[4,3,1],[4,3,1],5,4
[4,3,2],[4,3,2],4,4
[4,3,2,1],[4,3,2,1],4,4

What now?
(Enter "help." to view options)
|: show_rules(little_square).

. . . . . .


What now?
(Enter "help." to view options)
|: alter.


Current automatic acceptance threshold is 110
New value? 100.
Current threshold for referral to user is 0
New value? 0.

What now?
(Enter "help." to view options)
|: [shapes5].

shapes5 consulted 864 bytes 0.1 sec.
Editing picture:5
Searching picture: 5 for: little_square

Object:   little_square
bottom_left,(80,60)
bottom_right,(100,60)
top_right,(100,40)
top_left,(80,40)
Accepted.

Searching picture: 5 for: big_square
Searching picture: 5 for: triangle3

Object:   triangle3
top,(100,60)
left,(60,100)
right_angle,(100,100)
Accepted.

Searching picture: 5 for: triangle2

Object:   triangle2
left,(10,20)
```

```
bottom,(60,70)
right_angle,(60,20)
Accepted.

Searching picture: 5 for: triangle4

Object:   triangle4
right,(60,100)
top,(40,80)
right_angle,(40,100)
Accepted.


Object:   triangle4
right,(50,60)
top,(20,30)
right_angle,(20,60)
Accepted.


Feedback: triangle4, Picture 5
Feedback: triangle3, Picture 5
Feedback: triangle2, Picture 5
Feedback: little_square, Picture 5
Feedback: big_square, Picture 5
Removing: 5
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes6].

. . . . . .

Bid list empty

What now?
(Enter "help." to view options)
|: alter.


Current automatic acceptance threshold is 100
New value? 110.
Current threshold for referral to user is 0
New value? 0.

What now?
(Enter "help." to view options)
|: [rectanglesgen].

rectanglesgen consulted 264 bytes 0.016691 sec.
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes7].

shapes7 consulted 1600 bytes 0.18334 sec.
Editing picture:7
Learning: rectangle2
```

```
Learning: rectangle1
Bid list empty

What now?
(Enter "help." to view options)
|: show_rules(rectangle1).

. . . . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],3,2
[3,1],[3,1],2,2
[4,1],[4,1],4,2
[3,2],[3,2],4,2
[4,2],[4,2],3,2
[4,3],[4,3],3,2
[3,2,1],[3,2,1],2,2
[4,2,1],[4,2,1],2,2
[4,3,1],[4,3,1],2,2
[4,3,2],[4,3,2],2,2
[4,3,2,1],[4,3,2,1],2,2

What now?
(Enter "help." to view options)
|: show_rules(rectangle2).

. . . . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],4,2
[3,1],[3,1],3,2
[4,1],[4,1],2,2
[3,2],[3,2],2,2
[4,2],[4,2],2,2
[4,3],[4,3],4,2
[3,2,1],[3,2,1],2,2
[4,2,1],[4,2,1],2,2
[4,3,1],[4,3,1],2,2
[4,3,2],[4,3,2],2,2
[4,3,2,1],[4,3,2,1],2,2

What now?
(Enter "help." to view options)
|: [shapes8].

shapes8 consulted 2544 bytes 0.26667 sec.
Editing picture:8
Searching picture: 8 for: little_square
Searching picture: 8 for: triangle4
Searching picture: 8 for: triangle3
Searching picture: 8 for: triangle2
Searching picture: 8 for: triangle1
Searching picture: 8 for: big_square
Searching picture: 8 for: rectangle2
Searching picture: 8 for: rectangle1

Object:   rectangle1
bottom_left,(10,40)
bottom_right,(60,40)
top_right,(60,10)
```

```
top_left,(10,10)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   rectangle1
bottom_left,(50,80)
bottom_right,(100,80)
top_left,(50,40)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   rectangle1
bottom_left,(50,60)
top_right,(100,30)
top_left,(50,30)
Probability:   100%
Accept (a) or reject (r)?r.

. . . . . .
(another 25 candidates)
. . . . . .

Feedback: triangle4, Picture 8
Feedback: triangle3, Picture 8
Feedback: triangle2, Picture 8
Feedback: little_square, Picture 8
Feedback: big_square, Picture 8
Feedback: rectangle2, Picture 8
Feedback: rectangle1, Picture 8
Removing: 8
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes9].

. . . . . .

What now?
(Enter "help." to view options)
|: alter.


Current automatic acceptance threshold is 110
New value? 100.
Current threshold for referral to user is 0
New value? 0.

What now?
(Enter "help." to view options)
|: [shapes10].

shapes10 consulted 1824 bytes 0.20006 sec.
Editing picture:10
Searching picture: 10 for: little_square

Object:   little_square
bottom_left,(60,60)
bottom_right,(80,60)
top_right,(80,40)
```

```
top_left,(60,40)
Accepted.

Searching picture: 10 for: triangle4

Object:   triangle4
right,(50,70)
top,(30,50)
Accepted.

Searching picture: 10 for: triangle3
Searching picture: 10 for: triangle2
Searching picture: 10 for: triangle1

Object:   triangle1
right,(80,30)
right_angle,(50,30)
Accepted.

Searching picture: 10 for: big_square
Searching picture: 10 for: rectangle2
Searching picture: 10 for: rectangle1

Object:   rectangle1
bottom_left,(40,90)
bottom_right,(90,90)
top_right,(90,50)
top_left,(40,50)
Accepted.


Object:   triangle3
left,(0,90)
right_angle,(30,90)
Probability:   87.5%
Accept (a) or reject (r)?a.

. . . . . .
(another 7 candidates)
. . . . . .

Feedback: triangle4, Picture 10
Feedback: triangle3, Picture 10
Feedback: triangle1, Picture 10
Feedback: little_square, Picture 10
Feedback: big_square, Picture 10
Feedback: rectangle2, Picture 10
Feedback: rectangle1, Picture 10
Removing: 10
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes11].

shapes11 consulted 2064 bytes 0.23345 sec.
Editing picture:11
Searching picture: 11 for: little_square

Object:   little_square
```

```
bottom_left,(50,80)
bottom_right,(70,80)
top_right,(70,60)
top_left,(50,60)
Accepted.

Searching picture: 11 for: triangle4

Object:   triangle4
right,(90,50)
top,(70,30)
right_angle,(70,50)
Accepted.

Searching picture: 11 for: triangle3

Object:   triangle3
top,(60,40)
left,(30,70)
right_angle,(60,70)
Accepted.

Searching picture: 11 for: triangle1
Searching picture: 11 for: big_square
Searching picture: 11 for: triangle2

Object:   triangle2
left,(70,10)
bottom,(110,50)
right_angle,(110,10)
Accepted.


Object:   triangle2
left,(80,100)
right_angle,(110,100)
Accepted.

Searching picture: 11 for: rectangle1
Searching picture: 11 for: rectangle2

Object:   rectangle2
bottom_left,(20,60)
bottom_right,(40,60)
top_right,(40,10)
top_left,(20,10)
Accepted.

Object:   triangle4
top,(110,80)
right_angle,(110,100)
Probability:   85.714%
Accept (a) or reject (r)?a.

. . . . . .
(another 6 candidates)
. . . . . .

What now?
```

```
(Enter "help." to view options)
|: show_rules(big_square).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],11,5
[3,2,1],[3,2,1],9,5
[4,2,1],[4,2,1],8,5
[4,3,2],[4,3,2],11,6
[4,3,2,1],[4,3,2,1],8,5
[3,1],[3,1],29,5
[4,1],[4,1],15,5
[3,2],[3,2],18,7
[4,2],[4,2],25,6
[4,3],[4,3],16,6
[4,3,1],[4,3,1],13,5

What now?
(Enter "help." to view options)
|: show_rules(little_square).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],19,10
[3,1],[3,1],12,9
[4,1],[4,1],19,10
[3,2],[3,2],19,9
[4,2],[4,2],11,10
[4,3],[4,3],19,10
[3,2,1],[3,2,1],10,9
[4,2,1],[4,2,1],11,10
[4,3,1],[4,3,1],10,9
[4,3,2],[4,3,2],9,9
[4,3,2,1],[4,3,2,1],9,9

What now?
(Enter "help." to view options)
|: show_rules(triangle1).

. . . .

What now?
(Enter "help." to view options)
|: show_rules(rectangle1).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],7,5
[4,1],[4,1],10,4
[3,2],[3,2],9,5
[4,3],[4,3],7,4
[3,2,1],[3,2,1],5,5
[4,2,1],[4,2,1],5,4
[4,3,1],[4,3,1],5,4
[4,3,2],[4,3,2],4,4
[4,3,2,1],[4,3,2,1],4,4
[3,1],[3,1],12,5
```

```
[4,2],[4,2],10,4

What now?
(Enter "help." to view options)
|: show_rules(rectangle2).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],16,4
[3,1],[3,1],11,5
[4,1],[4,1],8,6
[3,2],[3,2],8,4
[4,2],[4,2],7,4
[4,3],[4,3],15,5
[3,2,1],[3,2,1],4,4
[4,2,1],[4,2,1],5,4
[4,3,1],[4,3,1],7,5
[4,3,2],[4,3,2],5,4
[4,3,2,1],[4,3,2,1],4,4

What now?
(Enter "help." to view options)
|: quit.


yes
| ?-
% Prolog execution halted
```

```
/* ************************************************************** */
/* ****                  RECOGNISE2 LISTING                      **** */
/* **** All training data loaded at start of system run         **** */
/* ************************************************************** */

21 % prolog
C Prolog version 1.5a.ikbs
% Restoring file /usr/lib/prolog/saved_states.d/Prolog1.5a
| ?- [master].

. . . . . .

yes
| ?- init.

Duplicates limit? 0.
Percentage prob. threshold for automatic acceptance?
(For no automatic acceptance, enter 110)  110.
Percentage prob. threshold for referral to user? 0.
Bid execution manual (m) or automatic (a)? m.

What now?
(Enter "help." to view options)
|: [test].

shapescheck consulted 2268 bytes 0.13333 sec.
shapesgen consulted 984 bytes 0.083335 sec.
shapes1 consulted 2000 bytes 0.21667 sec.
shapes2 consulted 2000 bytes 0.25 sec.
test consulted 7252 bytes 0.7 sec.

What now?
(Enter "help." to view options)
|: [rectanglesgen].

rectanglesgen consulted 264 bytes 0.016668 sec.

What now?
(Enter "help." to view options)
|: [shapes7].

shapes7 consulted 1600 bytes 0.18333 sec.

What now?
(Enter "help." to view options)
|: help.

Options available are:
Read in a data file: "[filename]."
Execute the next bid: "run."
Switch to automatic bid execution: "auto."
View the recognition rules for an object: "show_rules(Object)."
Alter acceptance/referral thresholds: "alter."
Quit the system: "quit."

What now?
(Enter "help." to view options)
|: auto.

Editing picture:7
```

```
Editing picture:2
Editing picture:1
Learning: rectangle2
Learning: rectangle1
Learning: triangle4
Learning: triangle3
Learning: triangle2
Learning: triangle1
Learning: little_square
Learning: big_square
Bid list empty

What now?
(Enter "help." to view options)
|: show_rules(rectangle1).

. . . . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],3,2
[3,1],[3,1],2,2
[4,1],[4,1],4,2
[3,2],[3,2],4,2
[4,2],[4,2],3,2
[4,3],[4,3],3,2
[3,2,1],[3,2,1],2,2
[4,2,1],[4,2,1],2,2
[4,3,1],[4,3,1],2,2
[4,3,2],[4,3,2],2,2
[4,3,2,1],[4,3,2,1],2,2

What now?
(Enter "help." to view options)
|: show_rules(rectangle2).

. . . . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],4,2
[3,1],[3,1],3,2
[4,1],[4,1],2,2
[3,2],[3,2],2,2
[4,2],[4,2],2,2
[4,3],[4,3],4,2
[3,2,1],[3,2,1],2,2
[4,2,1],[4,2,1],2,2
[4,3,1],[4,3,1],2,2
[4,3,2],[4,3,2],2,2
[4,3,2,1],[4,3,2,1],2,2

What now?
(Enter "help." to view options)
|: [shapes3].

shapes3 consulted 1008 bytes 0.11668 sec.
Editing picture:3
Searching picture: 3 for: rectangle2
Searching picture: 3 for: rectangle1
Searching picture: 3 for: triangle4
Searching picture: 3 for: triangle2
```

```
Searching picture: 3 for: triangle1
Searching picture: 3 for: little_square
Searching picture: 3 for: big_square

Object:   little_square
bottom_left,(90,70)
bottom_right,(110,70)
top_right,(110,50)
top_left,(90,50)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   little_square
bottom_left,(30,70)
bottom_right,(50,70)
top_right,(50,50)
top_left,(30,50)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   triangle1
bottom,(50,70)
right,(90,30)
right_angle,(50,30)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   triangle2
left,(80,20)
bottom,(110,50)
right_angle,(110,20)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   big_square
bottom_left,(10,50)
bottom_right,(50,50)
top_right,(50,10)
top_left,(10,10)
Probability:   66.667%
Accept (a) or reject (r)?a.

Feedback: rectangle2, Picture 3
Feedback: rectangle1, Picture 3
Feedback: triangle2, Picture 3
Feedback: triangle1, Picture 3
Feedback: little_square, Picture 3
Feedback: big_square, Picture 3
Removing: 3
Bid list empty

What now?
(Enter "help." to view options)
|: [shapes4].

shapes4 consulted 1056 bytes 0.11667 sec.
Editing picture:4
Searching picture: 4 for: little_square
Searching picture: 4 for: triangle1
Searching picture: 4 for: big_square
```

```
Searching picture: 4 for: rectangle2
Searching picture: 4 for: rectangle1
Searching picture: 4 for: triangle3

Object:   triangle3
top,(115,70)
left,(95,90)
right_angle,(115,90)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   rectangle1
bottom_left,(55,80)
bottom_right,(115,80)
top_left,(55,40)
Probability:   100%
Accept (a) or reject (r)?r.

. . . . . .
(another 6 candidates)
. . . . . .

What now?
(Enter "help." to view options)
|: [shapes5].

shapes5 consulted 864 bytes 0.11669 sec.
Editing picture:5
Searching picture: 5 for: little_square
Searching picture: 5 for: triangle1
Searching picture: 5 for: big_square
Searching picture: 5 for: triangle3
Searching picture: 5 for: triangle2
Searching picture: 5 for: rectangle2
Searching picture: 5 for: rectangle1
Searching picture: 5 for: triangle4

Object:   triangle4
right,(60,100)
top,(40,80)
right_angle,(40,100)
Probability:   100%
Accept (a) or reject (r)?a.

. . . . . .
(another 4 candidates)
. . . . . .

What now?
(Enter "help." to view options)
|: [shapes6].

shapes6 consulted 1488 bytes 0.15002 sec.
Editing picture:6
Searching picture: 6 . . . . .

Bid list empty

What now?
(Enter "help." to view options)
```

```
|: show_rules(big_square).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],8,5
[3,1],[3,1],11,5
[4,1],[4,1],9,5
[3,2],[3,2],12,5
[4,2],[4,2],10,5
[4,3],[4,3],8,5
[3,2,1],[3,2,1],7,5
[4,2,1],[4,2,1],6,5
[4,3,1],[4,3,1],7,5
[4,3,2],[4,3,2],8,5
[4,3,2,1],[4,3,2,1],6,5

What now?
(Enter "help." to view options)
|: show_rules(little_square).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[2,1],9,6
[3,1],[3,1],6,6
[4,2],[4,2],6,6
[3,2,1],[3,2,1],6,6
[4,2,1],[4,2,1],6,6
[4,3,1],[4,3,1],6,6
[4,3,2],[4,3,2],6,6
[4,3,2,1],[4,3,2,1],6,6
[4,1],[4,1],9,6
[3,2],[3,2],9,6
[4,3],[4,3],10,6

What now?
(Enter "help." to view options)
|: show_rules(triangle1).

. . . .


What now?
(Enter "help." to view options)
|: show_rules(triangle2).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[3,2,1],[3,2,1],2,2
[4,3,2,1],[4,3,2,1],2,2
[3,1],[3,1],4,2
[4,1],[4,1],7,2
[4,2,1],[4,2,1],3,2
[4,3,1],[4,3,1],3,2
[3,2],[3,2],8,2
[4,2],[4,2],5,2
[4,3,2],[4,3,2],3,2
[2,1],[2,1],6,2
```

```
[4,3],[4,3],6,2

What now?
(Enter "help." to view options)
|: show_rules(rectangle2).

. . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[3,2,1],[3,2,1],2,2
[4,2,1],[4,2,1],2,2
[4,3,2,1],[4,3,2,1],2,2
[2,1],[2,1],9,2
[3,1],[3,1],4,2
[4,1],[4,1],3,2
[3,2],[3,2],4,2
[4,2],[4,2],4,2
[4,3],[4,3],10,2
[4,3,1],[4,3,1],3,2
[4,3,2],[4,3,2],4,2

What now?
(Enter "help." to view options)
|: help.

Options available are:
Read in a data file: "[filename]."
Switch to manual bid execution: "man."
View the recognition rules for an object: "show_rules(Object)."
Alter acceptance/referral thresholds: "alter."
Quit the system: "quit."

What now?
(Enter "help." to view options)
|: quit.


yes
| ?-
% Prolog execution halted
```

```
/* ************************************************************** *
/* ****                RECOGNISE2 LISTING                      **** *
/* ****                  Traffic data                          **** *
/* ************************************************************** *

21 % prolog
C Prolog version 1.5a.ikbs
% Restoring file /usr/lib/prolog/saved_states.d/Prolog1.5a
| ?- [master].

. . . . . .

yes
| ?- init.

Duplicates limit? 3.
Percentage prob. threshold for automatic acceptance?
(For no automatic acceptance, enter 110)  110.
Percentage prob. threshold for referral to user? 0.
Bid execution manual (m) or automatic (a)? a.

What now?
(Enter "help." to view options)
|: [ttest].

trafficgen consulted 840 bytes 0.066668 sec.
trafficcheck consulted 176 bytes 0.016667 sec.
traffic1 consulted 4832 bytes 0.5 sec.
traffic2 consulted 4860 bytes 0.53333 sec.
traffic3 consulted 3664 bytes 0.38333 sec.
traffic4 consulted 1920 bytes 0.21667 sec.
ttest consulted 16292 bytes 1.8 sec.
Editing picture:16
Editing picture:15
Editing picture:14
Editing picture:13
Editing picture:12
Editing picture:11
Editing picture:10
Editing picture:9
Editing picture:8
Editing picture:7
Editing picture:6
Editing picture:5
Editing picture:4
Editing picture:3
Editing picture:2
Editing picture:1
Learning: big_lorry
Learning: small_lorry
Learning: van
Learning: car
Bid list empty

What now?
(Enter "help." to view options)
|: shopwrules(car).


Part/Pattern No. matches:
```

```
rear_wheel, Pattern1
front_wheel, Pattern1
top, Pattern4
front_window, Pattern5

Distance limits(Part1,Part2,MinX,MaxX,MinY,MaxY):
rear_wheel,front_wheel,-92,-41,0,1
top,front_wheel,-16,-11,-16,-15
top,rear_wheel,28,79,-17,-16
front_window,front_wheel,-24,-20,-17,-12
front_window,rear_wheel,20,72,-17,-13
front_window,top,-12,-7,-1,3

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[1,1],10,5
[3,1],[4,1],5,5
[4,1],[5,1],5,5
[3,2],[4,1],5,5
[4,2],[5,1],6,5
[4,3],[5,4],6,5
[3,2,1],[4,1,1],5,5
[4,2,1],[5,1,1],5,5
[4,3,1],[5,4,1],5,5
[4,3,2],[5,4,1],5,5
[4,3,2,1],[5,4,1,1],5,5

What now?
(Enter "help." to view options)
|: [traffic101].

traffic101 consulted 720 bytes 0.10001 sec.
Editing picture:101
Searching picture: 101 for: van
Searching picture: 101 for: car
Searching picture: 101 for: small_lorry
Searching picture: 101 for: big_lorry

Object:   big_lorry
wheel3,(139,133)
wheel2,(237,132)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   small_lorry
cab_top,(245,103)
rear_wheel,(139,133)
front_wheel,(237,132)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: big_lorry, Picture 101
Feedback: small_lorry, Picture 101
Removing: 101
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic102].

traffic102 consulted 720 bytes 0.10005 sec.
```

```
Editing picture:102
Searching picture: 102 for: van
Searching picture: 102 for: car

Object:   car
front_window,(194,110)
rear_wheel,(170,123)
front_wheel,(216,123)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: van, Picture 102
Feedback: car, Picture 102
Removing: 102
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic103].

traffic103 consulted 816 bytes 0.1167 sec.
Editing picture:103
Searching picture: 103 for: car
Searching picture: 103 for: small_lorry
Searching picture: 103 for: van
Searching picture: 103 for: big_lorry

Object:   big_lorry
wheel5,(68,145)
wheel2,(212,148)
wheel1,(272,148)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: big_lorry, Picture 103
Removing: 103
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic104].

traffic104 consulted 720 bytes 0.083362 sec.
Editing picture:104
Searching picture: 104 for: car
Searching picture: 104 for: small_lorry
Searching picture: 104 for: van
Searching picture: 104 for: big_lorry

Object:   big_lorry
wheel4,(100,139)
wheel3,(125,139)
wheel1,(286,142)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: big_lorry, Picture 104
Removing: 104
Bid list empty
```

```
What now?
(Enter "help." to view options)
|: [traffic105].

traffic105 consulted 816 bytes 0.083358 sec.
Editing picture:105
Searching picture: 105 for: car
Searching picture: 105 for: van

Object:   car
front_window,(360,110)
top,(367,108)
rear_wheel,(336,125)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   car
top,(52,116)
rear_wheel,(18,133)
Probability:   100%
Accept (a) or reject (r)?r.

Object:   car
front_window,(46,118)
rear_wheel,(18,133)
Probability:   85.714%
Accept (a) or reject (r)?a.

Feedback: van, Picture 105
Feedback: car, Picture 105
Removing: 105
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic106].

traffic106 consulted 480 bytes 0.050057 sec.
Editing picture:106
Searching picture: 106 for: small_lorry
Searching picture: 106 for: big_lorry

Object:   small_lorry
cab_top,(257,123)
rear_wheel,(144,150)
front_wheel,(251,151)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: big_lorry, Picture 106
Feedback: small_lorry, Picture 106
Removing: 106
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic107].

traffic107 consulted 672 bytes 0.083333 sec.
Editing picture:107
```

```
Searching picture: 107 for: car
Searching picture: 107 for: small_lorry
Searching picture: 107 for: van
Searching picture: 107 for: big_lorry

Object:   small_lorry
cab_top,(225,100)
rear_wheel,(142,125)
front_wheel,(221,125)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   car
rear_wheel,(101,80)
front_wheel,(150,80)
Probability:   50%
Accept (a) or reject (r)?r.

Object:   van
rear_wheel,(101,80)
front_wheel,(150,80)
Probability:   28.571%
Accept (a) or reject (r)?r.

Feedback: small_lorry, Picture 107
Feedback: van, Picture 107
Feedback: car, Picture 107
Removing: 107
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic108].

traffic108 consulted 816 bytes 0.083337 sec.
Editing picture:108
Searching picture: 108 for: car
Searching picture: 108 for: small_lorry
Searching picture: 108 for: van
Searching picture: 108 for: big_lorry

Object:   car
front_window,(250,119)
rear_wheel,(225,135)
Probability:   88.889%
Accept (a) or reject (r)?a.

Object:   car
rear_wheel,(214,87)
front_wheel,(262,87)
Probability:   46.154%
Accept (a) or reject (r)?r.

Object:   van
rear_wheel,(214,87)
front_wheel,(262,87)
Probability:   26.667%
Accept (a) or reject (r)?r.

Feedback: van, Picture 108
```

```
Feedback: car, Picture 108
Removing: 108
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic109].

traffic109 consulted 816 bytes 0.083346 sec.
Editing picture:109
Searching picture: 109 for: car
Searching picture: 109 for: van

Object:   car
front_window,(227,105)
top,(235,103)
rear_wheel,(205,120)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   car
rear_wheel,(125,75)
front_wheel,(175,75)
Probability:   40%
Accept (a) or reject (r)?r.

Object:   van
rear_wheel,(125,75)
front_wheel,(175,75)
Probability:   23.529%
Accept (a) or reject (r)?r.

Feedback: van, Picture 109
Feedback: car, Picture 109
Removing: 109
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic110].

traffic110 consulted 816 bytes 0.10001 sec.
Editing picture:110
Searching picture: 110 for: car
Searching picture: 110 for: van

Object:   car
front_window,(192,107)
rear_wheel,(169,123)
Probability:   90.909%
Accept (a) or reject (r)?a.

Object:   car
rear_wheel,(109,78)
front_wheel,(160,78)
Probability:   37.5%
Accept (a) or reject (r)?r.

Object:   van
rear_wheel,(109,78)
```

```
front_wheel,(160,78)
Probability:   21.053%
Accept (a) or reject (r)?r.

Feedback: van, Picture 110
Feedback: car, Picture 110
Removing: 110
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic111].

traffic111 consulted 912 bytes 0.10004 sec.
Editing picture:111
Searching picture: 111 for: car
Searching picture: 111 for: van

Object:   van
front_window,(230,110)
rear_wheel,(183,133)
front_wheel,(240,132)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   car
rear_wheel,(206,86)
front_wheel,(255,86)
Probability:   35.294%
Accept (a) or reject (r)?r.

Object:   van
rear_wheel,(206,86)
front_wheel,(255,86)
Probability:   19.048%
Accept (a) or reject (r)?r.

Feedback: van, Picture 111
Feedback: car, Picture 111
Removing: 111
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic112].

traffic112 consulted 768 bytes 0.083354 sec.
Editing picture:112
Searching picture: 112 for: car
Searching picture: 112 for: van

Object:   van
front_window,(210,110)
rear_wheel,(171,132)
front_wheel,(221,131)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: van, Picture 112
Feedback: car, Picture 112
```

```
Removing: 112
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic112].

traffic113 consulted 624 bytes 0.066687 sec.
Editing picture:113
Searching picture: 113 for: car
Searching picture: 113 for: small_lorry
Searching picture: 113 for: van
Searching picture: 113 for: big_lorry

Removing: 113
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic114].

traffic114 consulted 864 bytes 0.1 sec.
Editing picture:114
Searching picture: 114 for: car
Searching picture: 114 for: van

Object:   car
front_window,(51,133)
rear_wheel,(27,146)
front_wheel,(71,145)
Probability:   100%
Accept (a) or reject (r)?a.

Object:   car
front_window,(351,127)
rear_wheel,(331,140)
Probability:   91.667%
Accept (a) or reject (r)?a.

Feedback: van, Picture 114
Feedback: car, Picture 114
Removing: 114
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic115].

traffic115 consulted 576 bytes 0.066699 sec.
Editing picture:115
Searching picture: 115 for: car
Searching picture: 115 for: van

Object:   car
front_window,(142,108)
top,(149,106)
rear_wheel,(119,123)
front_wheel,(162,122)
Probability:   100%
Accept (a) or reject (r)?a.
```

```
Feedback: van, Picture 115
Feedback: car, Picture 115
Removing: 115
Bid list empty

What now?
(Enter "help." to view options)
|: [traffic116].

traffic116 consulted 528 bytes 0.066691 sec.
Editing picture:116
Searching picture: 116 for: car
Searching picture: 116 for: small_lorry
Searching picture: 116 for: van

Object:   small_lorry
cab_top,(247,118)
front_wheel,(240,145)
Probability:   100%
Accept (a) or reject (r)?a.

Feedback: small_lorry, Picture 116
Removing: 116
Bid list empty

What now?
(Enter "help." to view options)
|: show_rules(car).

. . . . . .

Set probabilities(PartSet,PatternSet,FeatureSets,Matches):
[2,1],[1,1],23,8
[3,1],[4,1],9,6
[4,1],[5,1],10,8
[3,2],[4,1],10,8
[4,2],[5,1],15,14
[4,3],[5,4],12,8
[3,2,1],[4,1,1],6,6
[4,2,1],[5,1,1],8,8
[4,3,1],[5,4,1],8,6
[4,3,2],[5,4,1],8,8
[4,3,2,1],[5,4,1,1],6,6

What now?
(Enter "help." to view options)
|: quit.


yes
| ?-
% Prolog execution halted
22 %
script done on Wed Jan  2 15:48:51 1991
```