A Connectionist Network for Object Recognition

Charles Cordingley

This report is submitted as partial fulfilment of the requirements for the Honours Programme of the Department of Computer Science The University of Western Australia 1995

Abstract

Object recognition is an important aspect of many vision based computer systems. One method of object recognition is to represent an object as a series of parametrised arcs, that can be transformed into a set of points in a metric space. The points can then be used to recognise features common to similar objects under this representation. This thesis examines the parametrised arc representation and how it can be used for the purpose of object recognition. A connectionist network to achieve this task will be described, its weights determined by the common features of the objects to be recognised. Results from experiments on the network, asserting the usefulness of the underlying representation scheme and the speed of the weighting scheme, will be presented.

Keywords: Object Representation, Object Recognition, Connectionist Network
CR Categories: I.2.6 (Learning), I.5 (Pattern Recognition)

Acknowledgements

I would like to thank my my supervisor Paul Hadingham for his guidance and help.

I would also like to thank Jeff Rohl, for setting me straight on context-free grammars.

Furthermore, thanks to all the other Honours students for various pieces of information and help, not to mention their drawing skills.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Object recognition is an important aspect of many vision-based systems, including robotics and image analysis. In this thesis, object recognition is considered to be the process of categorising input objects as members of internally defined object classes. There exist numerous techniques for object recognition. Pavlidis [17] extensively describes the significant underlying theories involved in the discipline, while Suetens, Fua and Hanson [22], give a detailed survey of current and notable techniques for object recognition.

In a number of papers, Hadingham [7, 8, 9, 10] suggests and develops a language for representing static two-dimensional edge-based objects. In this model an object is considered to be a set of semantically disjoint edges (an edgeset). Two lines, which may or may not cross in the image, are semantically disjoint if they belong to different edges in the original scene. Each edgeset consists of a series of connected arcs, which can be represented as a point in a metric space. These arcs are separated by regions of high curvature or by some occlusion. This segmentation of an object into edgesets and arcs has a certain basis in neurophysiology. The research of Hubel [13], demonstrates that the brain contains orientation selective cells, suggesting the capability to detect regions of high curvature, and thus decompose edge-based images into arcs.

This thesis details research where Hadingham's representation scheme was used as the formal basis of a object recognition system, implemented using a connectionist network. The language presented by Hadingham is extended, and the properties of this new representation model and its metric spaces are examined, with particular attention paid to the similarities and differences between certain objects. The results of this comparison are used to construct a measure of distance between objects, and a method of extracting recognisable features from sets of objects. A connectionist network, effectively an artificial neural network, is constructed. The weights on the connections in the network are fixed at initialisation, their values based upon the results obtained from the distance measure and the extracted features. Experiments testing and examining the capabilities of this network were conducted. The findings show that the representation scheme captures the recognisable features of object, that the recognition system produces acceptable results, and that it does so particularly fast.

It should be noted what this project does not entail. Hadingham's representation model is the central object of study, not image segmentation and edge detection techniques. Any such image pre-processing, in particular the formation of semantically disjoint edgesets, will be assumed to have been performed perfectly, before the input data is entered into the system. Although there exist many edge detection techniques (see Gonzales [6] for a fuller discussion), a program capable of fulfilling the image pre-processing required by this project is not currently available. Hence all input data will be manually constructed, rather than being extracted from actual images. The data will be assumed to be initially noise-free; however some inaccuracies in the parameter values may result from manual input. Thus the actual input is assumed to contain minimal noise.

A review of certain object recognition techniques and an examination of the parametrised arc representation developed by Hadingham, is given in Chapter 2. Extensions to the Hadingham representation scheme are discussed in Chapter 3. Chapter 4 details the method used to determine the distance between two objects, and the procedure for generating probalistic intersections, which are used to extract sets of common recognisable features from objects. In Chapter 5 the structure of the connectionist network is presented, along with the scheme to determine its weights from the probalistic intersections of objects representative of particular object classes. Chapter 6 gives the results of a number of experiments performed upon the network, and Chapter 7 summarises the results presented, suggesting avenues for further work.

CHAPTER 2

Literature Review

2.1 Object Recognition

Object recognition is a special case of pattern recognition, where the input data is a visual image, and the pattern to be matched represents an object. Thus to gain an understanding of the theory underlying object recognition, an investigation of the theory of pattern recognition forms a solid foundation. Pavlidis' seminal book *Structural Pattern Recognition* [17] introduced and defined the term it uses as its title [14]. Structural pattern recognition attempts to describe objects, often recursively, by their structure and then design a recognition scheme based upon this representation. This is analogous to the approach taken in this thesis. Pavlidis focuses mainly upon image segmentation, an area extraneous to this project, as all image pre-processing is assumed to have been previously performed.

Miclet [15] details the theoretical structures employed in structural pattern recognition. The fundamental theory surrounding each structure is presented, together with algorithms for determining the distance between objects modeled using the structures. Two structures examined by Miclet and used to represent objects in this project are context-free grammars (Chapter 3) and graphs (Chapter 4). Caelli, Ferraro and Barth [4] expand upon Miclet's work, suggesting four properties a representation structure should possess in order to facilitate 'strong' recognition. The properties are translation invariance, rotation invariance, unique object definition and explicit encoding of transformation states. Due to the pre-condition that only static two-dimensional images can be represented, the last property is inapplicable in this thesis, however the others are essential objectives in designing a representation scheme.

Suetens, Fua and Hanson [22] define the process of object recognition as "...the task of finding and labeling parts of a two-dimensional image that correspond to objects in the scene.". The paper from which the above quote originates contains a comprehensive survey of current and notable techniques for object recognition, each method placed in one of four categories, 'feature vector detection', 'fitting models to photometry', 'fitting models to symbolic structures', or 'composite strategies'. In the 'feature vector detection' technique, objects are represented as a vector of characteristic attributes, each vector corresponding to a point in a multi-dimensional attribute space. Thus recognition becomes the process of determining which vectors from the image, fall within the area in the attribute space representing the object model. 'Fitting models to photometry' involves attempting to match an object directly to the image data (photometry), for example finding circles by searching for arcs. 'Fitting models to symbolic structures' requires recognisable features to be extracted from an image and placed into 'symbolic' models, before attempting to identify them. Any combination of the previous three techniques is classified as a 'composite strategy'.

The recognition method detailed in this thesis falls within the 'composite strategy' category. This is because objects are represented as a set of feature vectors (points in a metric space), although they are first placed into a 'symbolic' structure, namely the extended Hadingham representation scheme (Section 2.2 and Chapter 3). Thus the technique is a combination of 'feature vector detection' and 'fitting models to symbolic structures'. Suetens, Fua and Hanson suggest that this technique is suitable for both complex image data and complex object models. However due to the assumption of perfect input, the complexity of the input image is irrelevant.

In 'feature vector detection' the set of all possible vectors forms a feature space, which is partitioned, during a "learning" phase, into regions representing different objects. This methodology is conducive to a connectionist implementation. Serra and Zanarini [20] demonstrate how a Hopfield network [11] can be 'taught' ASCII characters. Each character, and its complement, is represented by a fixed point within the network, that is if a character equivalent to a fixed point within the network is input, then that character will be output unchanged. This network is guaranteed to converge to only one of these fixed points, for any given input. In this work the feature vectors are the binary bitmaps of the characters. Bergman and Mulgaonkar [2] use a similar technique, with a three layer neural network, to recognise address blocks on envelopes, using position and shape to form the feature vectors.

A recognition system utilising the 'fitting models to symbolic structures' technique must first extract from the image a set of recognisable features. This is usually done using a local operator, hence all contextual information is lost. The features are then placed in 'symbolic' structures, and compared against internal models. The graph is a structure often employed in object representation schemes using this technique. The most basic form of graph matching involves searching for graphs that are identical (isomorphic) to the internal object graphs, Murray [16] uses such a method. However the requirement that the two graphs be isomorphic is an untenable precondition in many applications, as it assumes zero noise in the image data (unlike the minimal noise assumption in this thesis). A more robust procedure is to compare some measure of graph distance; one such method is suggested by Houraud and Skordas [12]. They match stereo images, thereby recovering the three dimensional geometry of the original scene. This is achieved by identifying maximal cliques in a graph where the nodes represent every possible pairing of features from the stereo images, and the arcs represent geometric compatibility between these pairings. The search for maximal cliques is an NP-complete problem, hence the above method was slow, requiring 26 minutes on a 11/780 VAX, when the input scene contained 430 lines.

Ayache and Faugeras [1] use graph matching and heuristic pruning in their HYPER (HYpothesis Predicted and Evaluated Recursively) system, which can identify occluded industrial parts. HYPER extracts linear edges from images and stores them as polynomial approximations to their contour. HYPER works by creating a large number of initial hypotheses about the position of an object. The hypotheses are expanded upon by iteratively matching every remaining image line to successive model lines, thus forming a tree with the hypothesis at the root and (possibly incorrect) matchings between the model and the image at the leaves. At each stage the next line to be added to the hypothesis is the one that results in the minimisation of a dissimilarity measure. The hypothesis resulting in the series of matchings with the least total distance is considered to be correct, and represent the desired object. This work was extended by Beveridge [3], and formalised into a combinatorial search of all possible matchings between an object and an image, employing heuristic shortest-path algorithms to speed the recognition process.

2.2 Hadingham's Representation

In a number of papers Hadingham suggests and develops a method for representing static edge-based two dimensional objects, as a series of parametrised arcs. Under this scheme an edge is decomposed into a series of connected arcs, each arc delineated by areas of high curvature or occlusion [8]. This decomposition constitutes the basis of an object description language, which is proved to be finite for digitised images. Futhermore the language is shown to be a semigroup [7], although the properties of semigroups are yet to be exploited by the language.

The context-free grammar proposed by Hadingham [10] is defined in Table 2.1. The *orientation* of an object is defined to be the angle the first arc makes with respect to the horizontal. Every arc is non-uniquely defined by three parameters: the *knot angle*, which is the angle between an arc and its preceding arc (or in the case of curves, the angle between the tangents); the *sweep angle*, which is the angle subtended by the curve of the arc (0 if the arc is a straight line); and the *length* of the arc. Figure 2.1 illustrates the arc parameters, and the description of the object being: $0((\frac{\pi}{2},0,3)(\frac{\pi}{2},0,2)(\frac{7\pi}{4},\frac{\pi}{2},\pi)(\frac{7\pi}{4},0,2)(\frac{3\pi}{4},0,3\sqrt{2})(\frac{\pi}{4},0,5))$, starting at

object	::=	*orientation*(edgeset)
edgeset	::=	edgeset edge
edge	::=	edge segment
segment	::=	arc
arc	::=	(*knot-angle*, *sweep-angle*, *length*)

Table 2.1: Hadingham's representation scheme

the bottom left corner, and proceeding clockwise. The knot angle is the clockwise change in direction between the previous arc and the current arc, resulting in a value on the interval $[0, 2\pi]$.

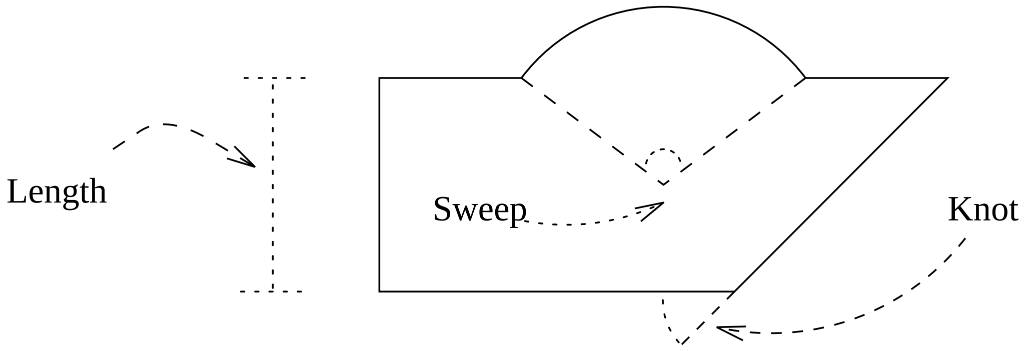


Figure 2.1: An example object displaying parameters

The possible advantages of Hadingham's scheme become apparent if arcs are viewed as elements of a three dimensional metric space (over $\mathbb{R}$) called the 'Arc Parameter' space (AS). This metric space also fulfills the requirements of a vector space, however the properties of vector spaces are not utilised by this project, thus the term metric space is used to avoid confusion. There exists an *onto* function that maps any allowable arc to a point in the AS. This function simply sets the parameters of the point equal to the appropriate parameters of the arc under consideration. However as arcs are not uniquely defined, this function is not *one-to-one*, and many arcs may map to the same point in the AS. Hence a four dimensional metric space ($AS \times \mathbb{R}$, where $\mathbb{R}$ is the set of natural numbers) is required to completely describe an object, this is called the augmented AS. The fourth dimension is defined as the number of arcs mapping to that point, the "mapping multiplicity". Other, higher dimensional, metric spaces may be constructed to represent relationships between an arbitrary number of consecutive arcs. For instance, the binary relationships between consecutive arcs can be expressed in a seven dimensional space ($AS^2 \times \mathbb{R}$), the axes being the six parameters from both arcs combined and the "mapping multiplicity". In general the relationships between n consecutive arcs can be modeled in a metric space of order $AS^n \times \mathbb{R}$.

The augmented AS and binary relationship space of the object in Figure 2.1 are given in Figure 2.2. The seven dimensional binary relationship metric space is illustrated as six, three dimensional graphs where the parameters of adjacent arcs are plotted against each other. In both graphs the "mapping multiplicity" at a point is shown by the size of that point. In order for an object's metric spaces to be invariant to the direction of arc traversal, before mapping arcs to points, the knot-complement of the object is calculated. The knot-complement of an object is the series of arcs in reverse order and with complement knot angles, but otherwise identical to the original object. The required metric spaces are then generated from both the object and its knot-complement. Thus the augmented AS of an object will always be symmetrical around the plane *knot-angle*$=\pi$, however this is not necessarily the case in metric spaces modeling binary or higher relationships between arcs, due to the order of the arcs being reversed. In the binary relationship space graphs, `S` is the sweep-angle (in π), `K` is the knot-angle (in π), and `L` is the length.

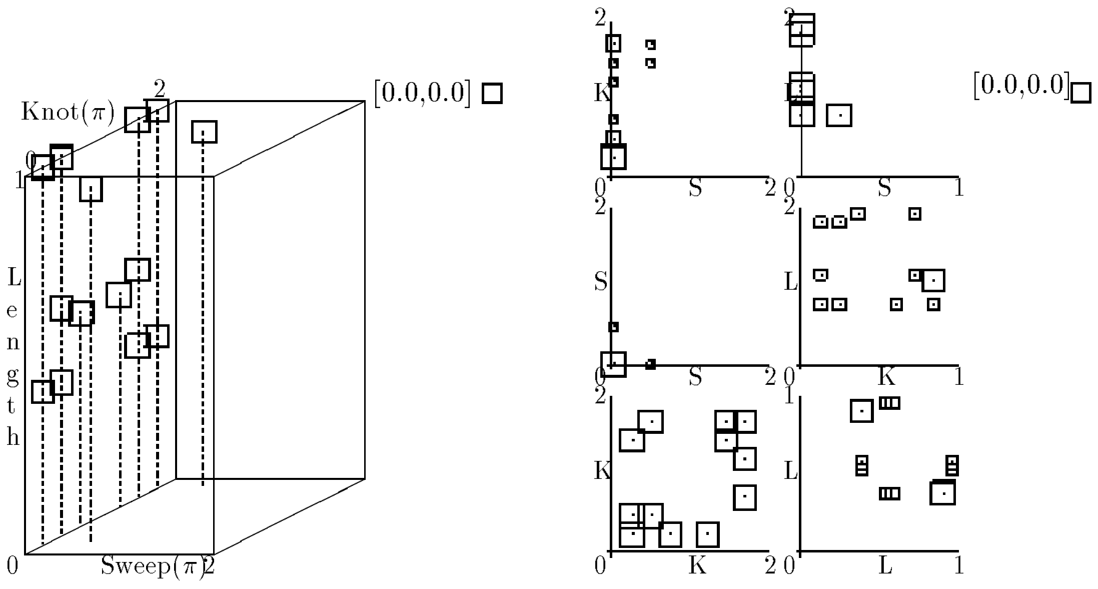


Figure 2.2: AS and binary relationship metric spaces of the object in Fig 2.1

CHAPTER 3

Object Representation

Hadingham's representation scheme [10] has a number of deficiencies associated with it. Firstly the scheme, while translation invariant, is rotation and scale variant. That is objects which are identical in all respects other than their initial orientations or size, will be represented as different objects. This is an unnecessary complication in object recognition: an inverted person is still the same person. The scheme is also unable to model all possible two dimensional edge-based objects. It constrains an object to consist of a single series of connected arcs, with each arc having at most one directly consecutive arc. Objects that comprise multiple disjoint edgesets, or that contain more than two arcs connecting at the same point can not be modeled without the loss of infomation on some of the connections. Figure 3.1 illustrates how a comparatively simple object is impossible to describe using Hadingham's representation scheme.

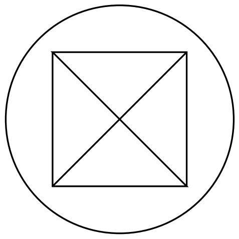


Figure 3.1: An impossible object in Hadingham's representation scheme

The context-free grammar defined in Table 3.1 in extended-BNF form (according to the rules presented by Rohl [19]), is an extension of Hadingham's scheme, and overcomes the shortcomings mentioned above. In the grammar braces signify zero or more occurrences of the enclosed symbol sequence. The arc parameters knot-angle and sweep-angle are determined in a similar manner to the original grammar. However the length of the arcs are normalised by the maximum arc length in the object, resulting in a value on the interval $[0, 1]$. The `position` rule defines the relative position of an edgeset compared to the other edgesets in an object. This is accomplished using polar coordinates, the angle and distance generated from a consistently selected origin. The distance parameter in the `position` rule is also normalised by the maximun arc length.

object	=	"(" edgeset { position "(" edgeset ")" } ")".
position	=	"[" angle "," distance "]".
edgeset	=	edge.
edge	=	arc { "(" edge ")" } {arc}.
arc	=	"(" knot-angle "," sweep-angle "," normalised-length ")".

Table 3.1: The extended representation scheme

The extended representation scheme retains the major attributes of the original. Similarly the method of forming metric spaces from the arc parameters, and the characteristics of these spaces remain equivalent. Advantages over Hadingham's scheme include rotation invariance, due to the removal of the initial orientation, and scale invariance, since the arc lengths are normalised. Recursion in the `edge` rule allows an arc to have an arbitrary number of directly consecutive arcs. Additionally, multiple `edgeset` terms in the `object` rule permits an object to comprise any number of disjoint edgesets. The addition of rotation invariance causes the representation scheme to satisfy the three applicable properties required for 'strong' recognition, as presented by Caelli [4] (Section 2.1). Also the extensions to the representation scheme make it possible to encode all possible static two-dimensional edge-based objects. Thus the object displayed in Figure 3.1 is a valid object in the extended representation scheme, and its arc-based description is shown in Table 3.2. All angles are in multiples of π.

Unfortunately the method of encoding the relative position of edgesets introduces a new constraint. If the position parameters are not consistently measured from corresponding arcs in different objects, then the position of edgesets in those objects will not match. Imagine the disparity between the position of a car's tyres determined first from the bumper, and then from the roof. To partially solve this problem, the positions of edgesets are assumed to have been consistently calculated relative to the first arc in the first object, the first object being the largest edgeset (determined by number of arcs).

In order to facilitate easy use of the extended representation scheme there are two tools `convert` and `graph`. The program `convert` verifies that the input data conforms to the grammar rules in Table 3.1, normalises the length parameters, and converts all the angles to radians. Table 3.2 displays output from `convert`, the first edgeset is the crossed square, encoded starting at the bottom left corner of the square and proceeding clockwise, the circle comprises the second edgeset. The braces and their contents denote an arc cycle (an arc connecting to a previ-

```
((0.500,0.000,0.1667){>1}
    (0.5000,0.0000,0.1667){>2}((0.7500,0.0000,0.2357){1,0.2500})
    (0.5000,0.0000,0.1667)((0.7500,0.0000,0.2357){2,0.2500})
    (0.5000,0.0000,0.1667){1,0.5000}
[0.0000,0.0000] ((0.0000,2.0000,1.0000)))
```

Table 3.2: The arc-based description of the object in Fig 3.1

ously described arc). The first number inside the braces is a label by which the cycle is referenced, and the second number (if present) is the knot angle between the two end arcs. For example, the label `{2,0.2500}` in Table 3.2, denotes that the diagonal originating at the bottom right corner of the square in Figure 3.1 connects with the top left corner, with a knot angle of $\frac{\pi}{4}$. The use of labels to indicate a cycle in an object is purely for simplification of the description process, as the representation scheme completely encodes an object. The `graph` program plots the four dimensional augmented AS and/or the seven dimensional binary relationship space of an object in LaTeX `picture` format. In both output graphs the "mapping multiplicity" dimension is normalised by the ∞-norm and depicted by the size of a point in the object. Thus any points in the representation scheme are constrained to be within the interval $[0, 1]$ for length and "mapping multiplicity", and the interval $[0, 2\pi]$ for knot and sweep angles. Different edgesets are represented in the graphs by different symbols; the relative positions of each edgeset denoted in a symbol key, placed in the top right corner of the graph.

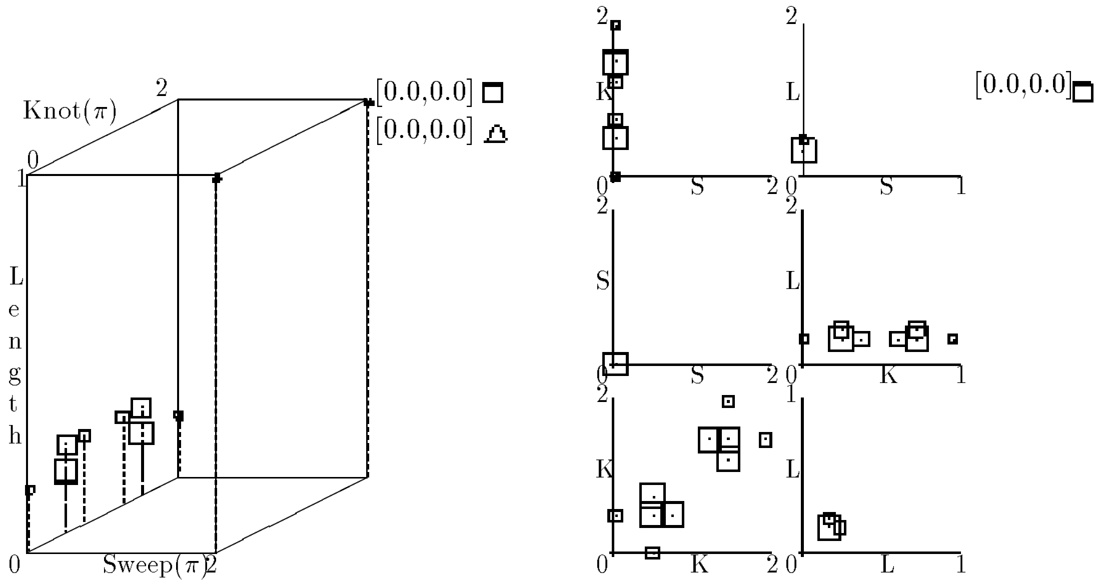


Figure 3.2: AS and binary relationship graphs of the object in Fig 3.1

CHAPTER 4

Object Comparison

4.1 Object Distance

An object recognition system requires the ability to compare the level of similarity between objects. This method of comparison must be able to distinguish between different broad classes of object, for example cars or people, while retaining the capacity to differentiate among members of an object class, such as the objects in Figures 4.1. The ability to convert an object into a set of points in a metric space (Section 2.2), suggests the application of geometric techniques to these metric spaces, in order to determine the level of similarity between objects.

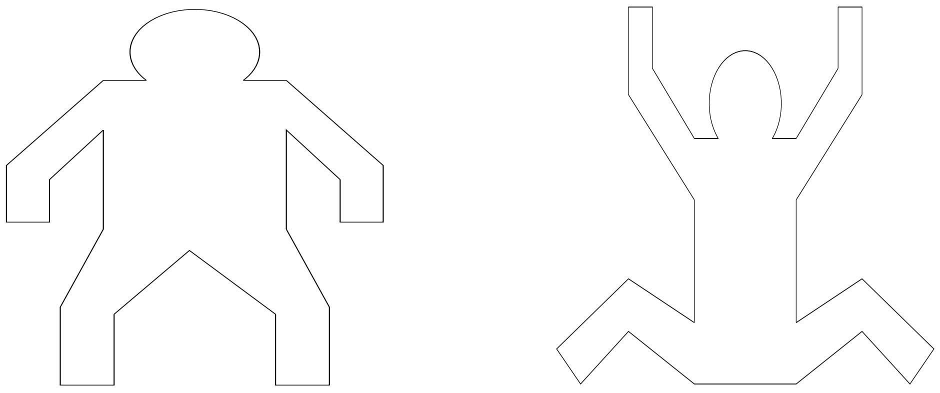
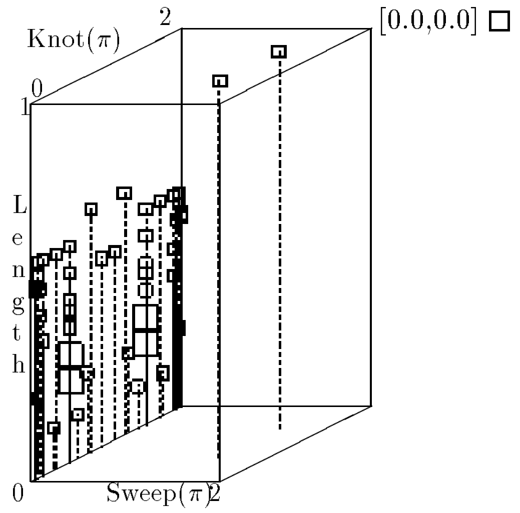


Figure 4.1: Person A and B

In this thesis, the level of object similarity is measured by calculating the cumulative geometric distance between sets of points in a metric space, such as the augmented AS (Figure 4.2). The distance between any two points, p_i and p_j, in a metric space is defined to be $dist(p_i, p_j)$, the length of the line segment that has the two points as its endpoints. This is the usual euclidean metric in $\mathbb{R}^{k+l}$ and can be easily computed for any two points by squaring the differences of their parameters, then taking the square root of the sum of these squares. The n^{th}-partial distance between two objects, O_1 and O_2, is denoted $d_n(O_1, O_2)$ and defined to be the summation of the distances between the n closest points, (p_i, p_j), where $p_i \in O_1$, $p_j \in O_2$, and (p_i, p_j) are removed from O_1 and O_2 respectively after being used in calculating the partial distance. In mathematical notation the partial distance is:

$$d_n(O_1, O_2) = \sum_{k=1}^{n} \left(\min_{0 \leq i < |O_1|, 0 \leq j < |O_2|} dist(p_i, p_j),\ O_1 \setminus p_i,\ O_2 \setminus p_j \right)$$

where $\mid O_1 \mid$ represents the number of points in the set O_1, similarly $\mid O_2 \mid$ is the size of O_2. Note that in the above equation k is simply a counter, ensuring that the distances between the n closest pairs of points are summed. Thus the total distance between two objects is the n^{th}-partial distance where n is the number of points in the smaller object. This distance scheme is not transitive, that is if $dist(p_i, p_j)$ and $dist(p_j, p_k)$ are less than some arbitrary number, it does not necessarily follow that $dist(p_i, p_k)$ is also less than this number. Although it is unreasonable to expect any useful distance measure to satisfy transitivity, this property would be useful when computing intersections. However the scheme is symmetric, $dist(p_i, p_j) = dist(p_j, p_i)$, and the distance from a point to itself is always zero, $dist(p_i, p_i) = 0$.



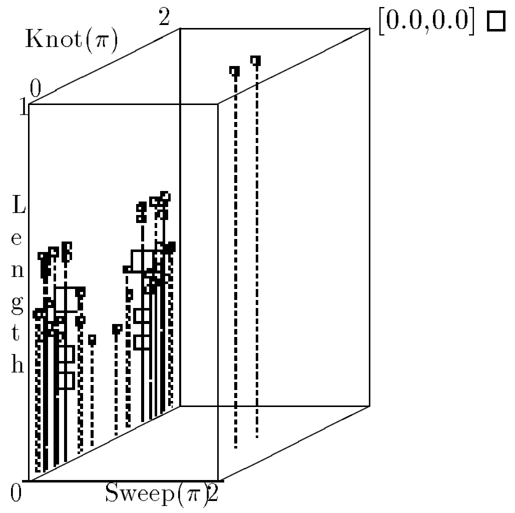


Figure 4.2: The augmented AS's of Person A and B

The program `compare` calculates the partial distances between any two objects. The output of `compare` when the augmented AS's of two people (Figure 4.2) and the augmented AS of the example object (Figure 2.1) are input, is graphed in Figure 4.3. This graph demonstrates how different objects are separated by the distance measure, the distance between persons A and B is clearly less than the distance between either of them and the example object. It should be noted that the person-example object comparisons matches only half the number of points as the person-person comparison. This is because the example object has roughly half the number of arcs of the objects it was compared against. When necessary, the `compare` program can assign values to each unmatched point, based upon a specified function (default value= $\frac{1}{10} \times (number\ of\ unmatched\ points)^2$). Additionally, the `compare` program can ignore one or more of the arc parameters when calculating distance, effectively assigning the value zero to all instances of those parameters.

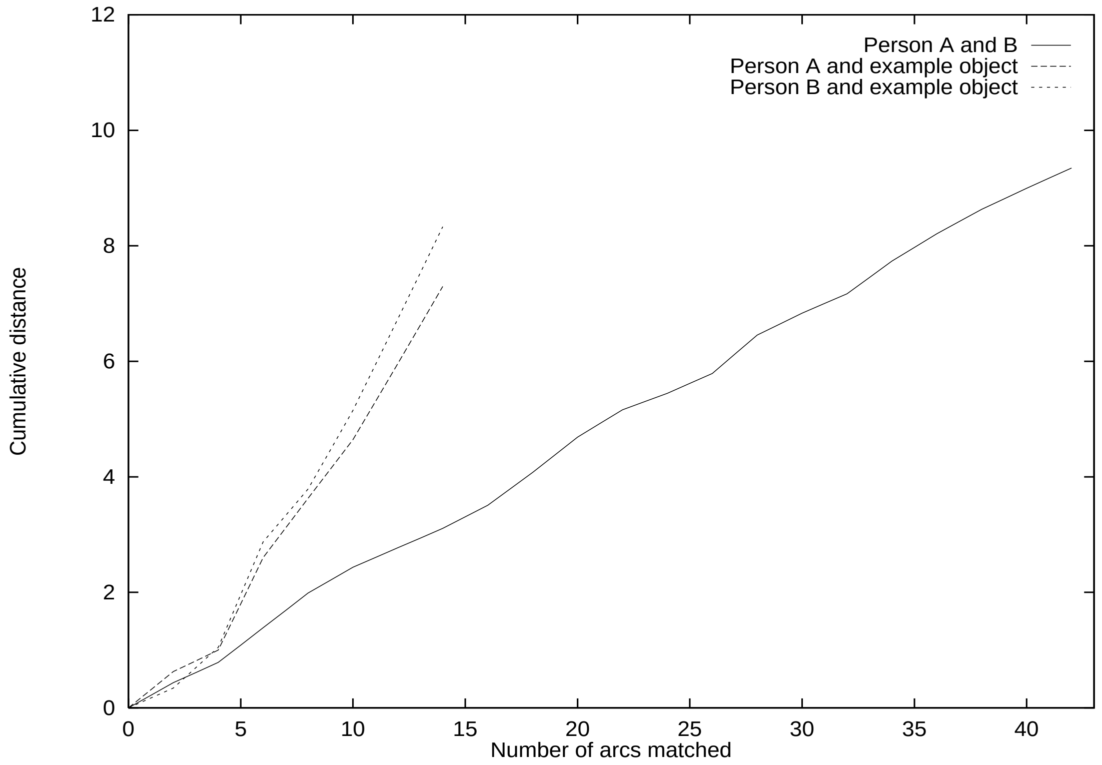


Figure 4.3: Partial distances resulting from comparing Person's A and B, and the example object

4.2 Object Intersection

Object recognition systems generally identify objects by comparing them against an internal set of recognisable features common to a particular object class. Thus a complete object recognition system requires the ability to extract such a set of recognisable features, from a number of objects belonging to the same class. One method of deriving a set of common features from any set of objects is to find their probabilistic intersection, that is those characteristics which are possibly contained by all or some of the objects. Using the extended representation scheme, the probabilistic intersection is found by comparing the points contained in each object, and those within a certain distance of each other, are clustered together to form a single point in the intersection. However since the distance measure is not transitive, there are a number of possible methods for determining which features should be clustered together. This project utilises two clustering algorithms, namely blobbing and maximal cliques.

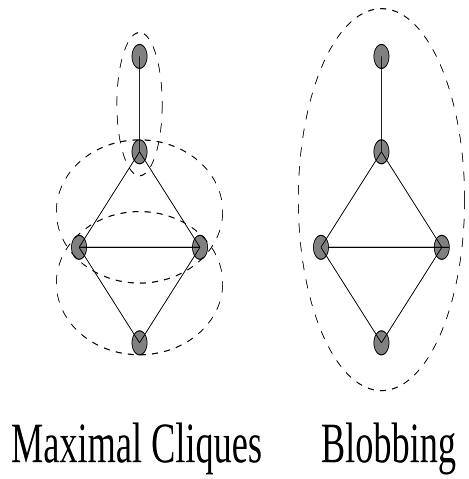


Figure 4.4: Comparison of clusters formed by each clustering algorithm

The first step in determining the intersection of a number of objects is to concatenate the sets of points in each object. Next the distance between every pair of points is calculated, although as geometric distance is symmetric, and the distance between a point and itself must equal zero, only $\frac{p^2-p}{2}$ distances need be calculated, where p is the total number of points. Those pairs greater than a specified threshold value apart (default value=`0.1`) are discarded, resulting in a list containing only the pairs of points which are possibly the same point.

The list of pairs of points returned from the distance function can be thought of as the adjacency list in an undirected graph. In this graph there exists one node for each point in the distance list, and if there exists a pair (p_i, p_j) in the distance list then there is an arc connecting the nodes representing p_i and p_j. Thus the sets of points to be clustered together should form connected subgraphs of this graph. This project utilises two clustering algorithms, namely blobbing and maximal cliques. The blobbing algorithm simply finds the maximal connected subgraphs; that is those subgraphs where there is a path from any node to any other node, and if any node not in the subgraph is added to it, without an appropriate arc, this property is lost. The maximal cliques algorithm finds the maximal cliques in the graph. A maximal clique is a subgraph in which every node is adjacent to every other node, and if any node not in the subgraph is added to it, this property is lost. Unfortunately searching for maximal cliques has been demonstrated to be an NP-Complete problem [5], compared to the polynomial blobbing algorithm; however there exist some heuristics to speed the search. Reingold, Neivergelt and Deo [18] give such an algorithm centred upon the notion of a tree search, with heuristics to prune certain branches; the pseudo-code upon which this implementation is based is presented in Appendix B. Figure 4.4 shows the result of each clustering algorithm on a simple graph. The dashed ellipses surround those points that are placed in the same set, and are thus clustered together.

Once the sets of points to be clustered have been determined, the average values of each parameter within each set are calculated. These averaged points are then sorted, their ranking dependent upon the number of points in the clustering set that was used to form them. This ordered list of averaged points is the probabilistic intersection of the original objects. The position of a point in the list represents, in descending order, the probability that point is in the intersection, relative to the other points. One problem with this method is that the resulting intersection may have more points than any of the objects from which it was generated.

The program `inter` finds the probabilistic intersection of a given set of objects, using either clustering algorithm as required, and employing the distance functions developed for the `compare` program. The program outputs ten intersections, one extracted from the points in the augmented AS, combined with the four derived when the distance function ignores each of the four parameters in turn, the remaining five being formed from the binary relationship space in a similar manner. The average and standard deviation of the number of points in the input objects for each metric space, are also calculated and output by `inter`. Before outputting the intersections, `inter` removes duplicate points between intersections, and ranks the points in each intersection with a call to the standard C library function `qsort`.

Figures 4.5 and 4.6 display the augmented AS intersection of person's A and B, graphed using the `graph` program. It is immediately clear from the figure that the intersection resulting from the maximal cliques algorithm has many more points than the blobbing intersection. In fact the maximal clique intersection has 38 points, compared to the blobbing intersection's 14, while the actual objects themselves contain 42 points. However a visual inspection of both intersections show that they clearly only have points in those areas where points should be expected. To test their accuracy, the intersections were compared against Person A, (Figure 4.2); the partial distances of the result are shown in Figure 4.7. The Person line shows the comparison of the two sets of points used to derive the intersection (Figure 4.2). This is effectively the control. If either of the intersection comparisons give greater partial distances than this line then that intersection

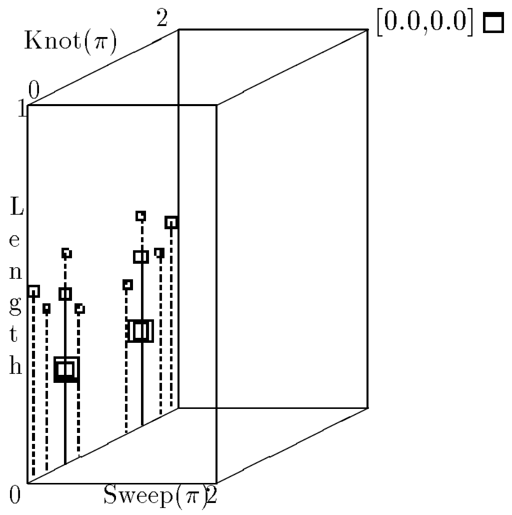


Figure 4.5: Probabilistic intersection formed using blobbing

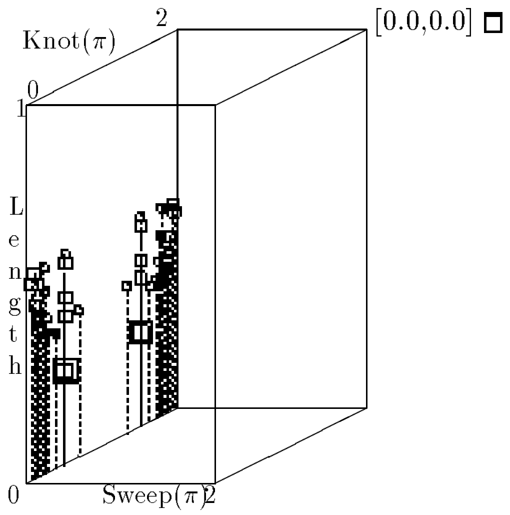


Figure 4.6: Probabilistic intersection formed using maximal cliques

is of minimal use in object recognition, as the result is unlike either of the sets from which it was derived. However, the closer the intersections are to the zero distance line, the better they are at extracting recognisable features. The comparisons of both algorithms with the object give similarly accurate intersections, both well underneath the Person line; the blobbing algorithm giving a slightly better intersection, although over a fewer number of points.

In order to test the time required by each clustering algorithm, they were both tested on the same test data, and with the same value (`0.1`) for the threshold distance. The test data consisted of 20 points with no two points within the threshold distance of each other. The test data was repeatedly input to the program, such that it concatenated the test data onto itself a number of times, before attempting to find the intersection. Thus for each repetition, every point was part of the intersection, a worse case than is normally likely to occur. Figure 4.8 shows the output of the Unix `time` command when the program was executed. Unsurprisingly the NP-Complete clique algorithm required exponentially more time than the polynomial blobbing algorithm.

These results suggests that in object recognition, the blobbing algorithm is useful for quickly determining the broad object category, however in those cases that require precision over a greater number of points, the maximal clique algorithm would be more suitable.

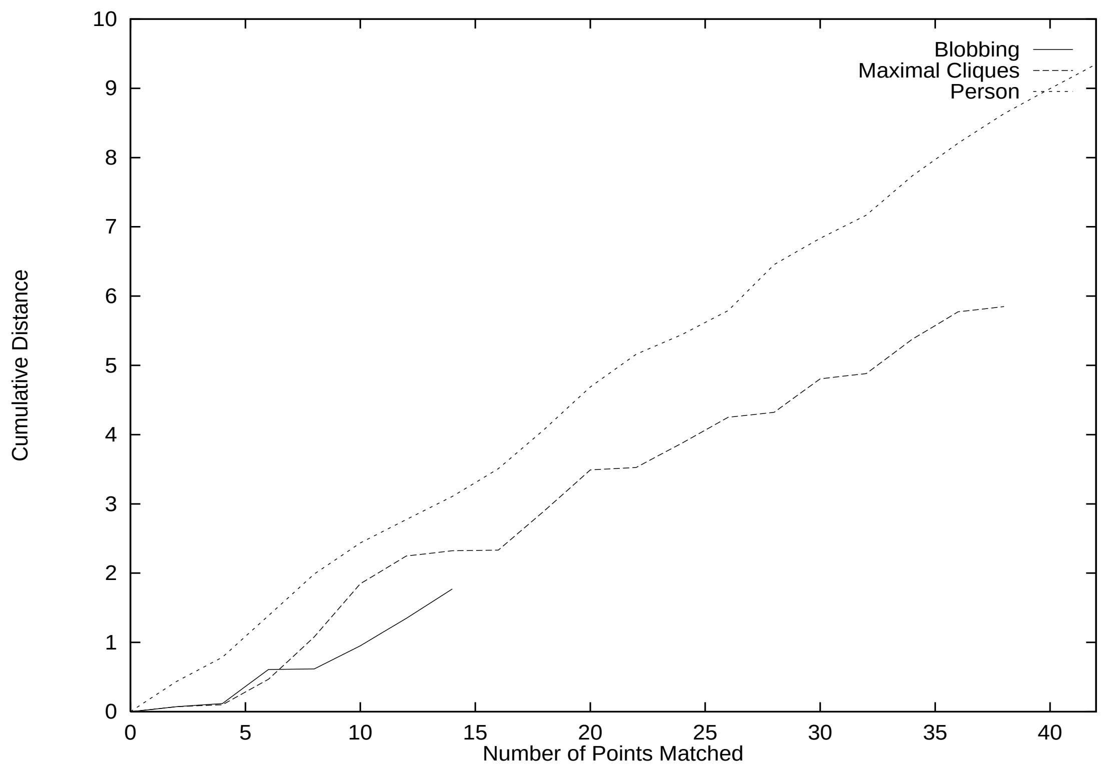


Figure 4.7: Comparison of the accuracy of each clustering algorithm

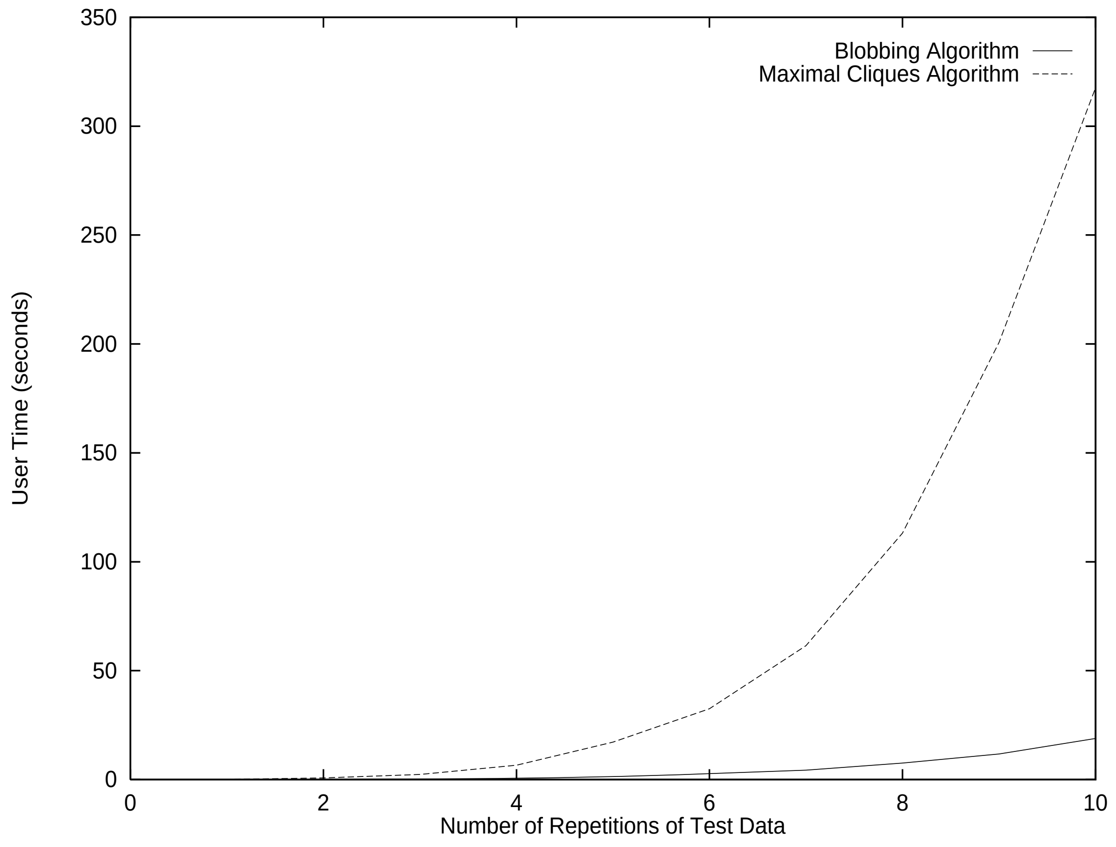


Figure 4.8: Comparison of the time required by each clustering algorithm

CHAPTER 5

Object Recognition

5.1 The Network

The actual object recognition process is achieved in this project through the use of a connectionist network. The network employed consists of three layers: the point (input) layer, which compares individual points; the metric space (middle) layer, which compares metric spaces; and the object (output) layer, which compares objects. Each layer is connected to the layer above, and no other. No traditional learning algorithm is used. The weightings on all connections in the network, both intra-layer and inter-layer, are pre-determined by probabilistic intersections, extracted from objects that are considered representative of the object classes to be learnt. Figure 5.1 presents a basic illustration of the network's structure and connections.

The point layer consists of ten sub-networks. Each sub-network corresponds to one of the ten metric spaces in a probabilistic intersection. That is, the sub-networks have the same dimensions, and the same bounds upon their size as the equivalent metric space. However the sub-networks comprise discrete hypercubes, rather than being continuously valued. The size of these hypercubes is consistent across the sub-networks and is equal to the threshold value that is used in determining probabilistic intersections. This is because a single hypercube forms a basin of attraction for a single point in an intersection. Hence this basin must not be significantly greater than the distance at which two points may be clustered together, otherwise points in an unknown object could be incorrectly recognised as equivalent to a point in the intersection. Similarly, the basin should be as large as possible to facilitate the correct detection of points that have a slightly erroneous position. Thus, with the size of the hypercubes set to `0.1`, the resulting sub-networks are particularly large (160 million hypercubes in the sub-network associated with the full binary relationship space). This represents a large store of memory for learning objects. Hypercubes are binary, either they are active,

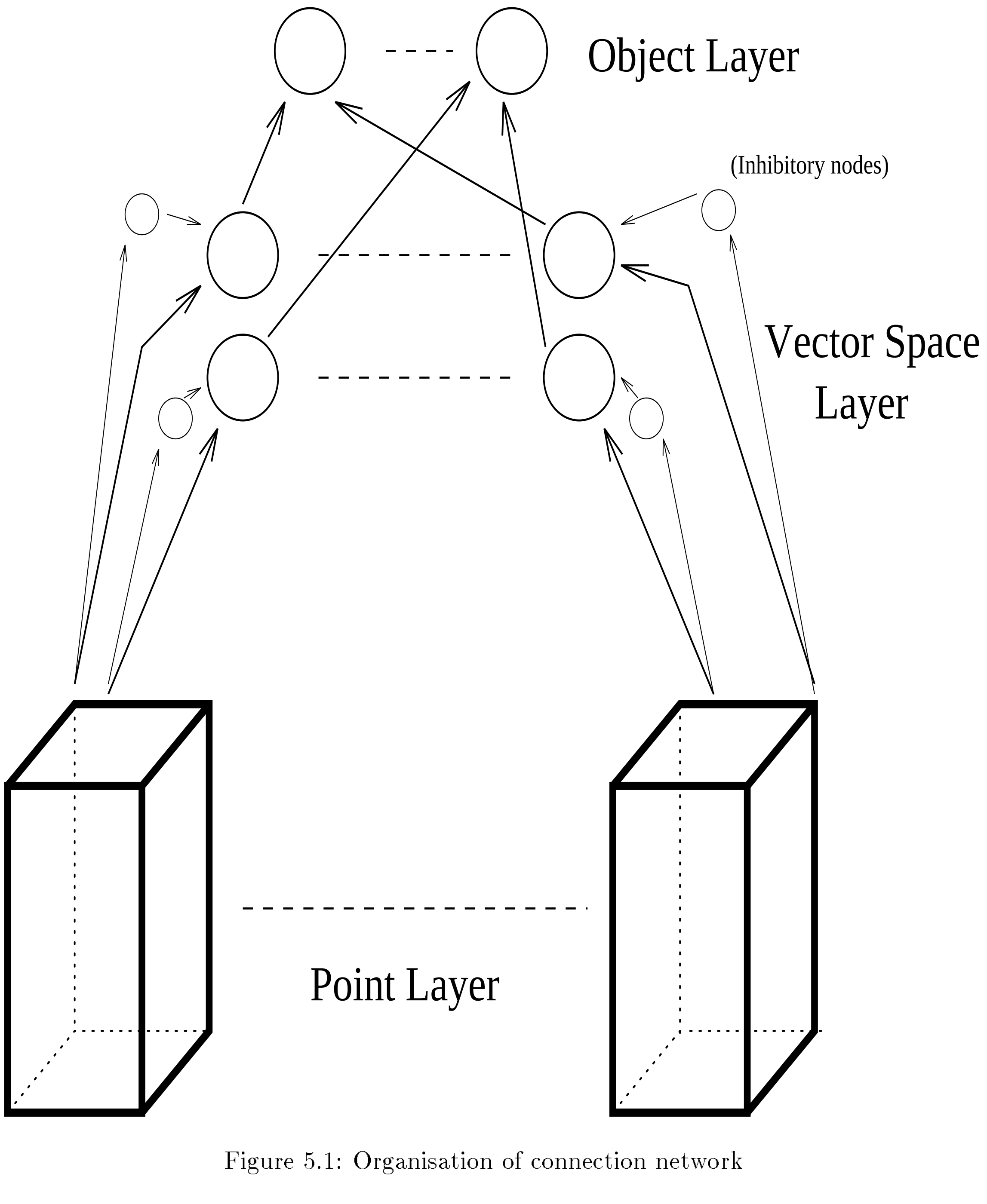


Figure 5.1: Organisation of connection network

and fire values up to nodes in metric space layer, or they do nothing.

The metric space layer contains twenty nodes for each distinct object class that is being learnt. The twenty nodes are made up of two nodes per separate metric space in a probabilistic intersection, that is there are two nodes for every metric space in every object class. One of the nodes simply counts the number of points in the metric space to which it is associated, these are called the inhibitory nodes (shown as small circles in Figure 5.1). The other holds a value signifying the level of similarity between a point layer sub-network, and the associated metric space of the object class that is represented by that node. These nodes are called comparison nodes (shown as large circles in Figure 5.1). The object layer contains one node for every object class learnt, the nodes at this level denote the relative likelihood that an object will be recognised as a member of a particular object class. Nodes in both the metric space and object layers are continuous, their value can be any real number, although their contents normally belong to certain intervals, as explained later in this section.

In the point layer, none of the hypercubes inside a sub-network are interconnected, neither are any of the sub-networks linked with each other; all the connections are made by individual hypercubes to nodes in the metric space layer. Every hypercube has a connection to every inhibitory node associated with the metric space that represents that hypercube's sub-network, regardless of the object class to which the inhibitory node belongs. The weight on the inhibitory link is `1`, thus as hypercubes have a binary value, the input to each inhibitory node will be equal to the number of hypercubes firing, which is equivalent to the number of points in the unknown object. Additionally, any hypercubes that are equivalent to a point in one of the intersections to be learnt, will have a link to a number of comparison nodes in the metric space layer. The comparison node to which a hypercube connects, is the node to which represents the metric space that the hypercube is associated, and the object class from which the equivalent point originated. If there exists more than one object that contains an equivalent point, then the hypercube will have a connection to all the associated comparison nodes.

The weight on a connection from a hypercube to a comparison node, is proportional to the relative importance of that intersection point to which the hypercube is equivalent. This is determined by the ratio between the size of the clustering set of the equivalent point, against the combined size of all the clustering sets in the relevant intersection. The weights are scaled such that if all the hypercubes equivalent to points, representing a single metric space of a particular intersection fire, then the value `10` is sent to the appropriate comparison node. However there is a problem with this method; the process of elucidating probabilistic intersections may result in a larger set of points than any of the individual objects that were used to form it. If this occurs it will place unreasonable demands upon an object to be recognised. To be fully recognised, an object will require more points than could be expected from a member of its object class. To alleviate this problem, only a certain number of the most important points in each metric space from the intersection, are used to initialise hypercubes. Due to the intersection points being ranked , each point can be examined in order until the limit has been reached. This limit is set as the average number of points, in the metric space for the object class in question, plus three times the standard deviation. The limit is set at this level so as not to penalise those objects with a substantially greater number of points than others in its class. Statistics indicate that the size of 99.7% of objects in an object class, will lie within three standard deviations of the average number of points in that class. However results suggest that the limit is currently set too high, and should be lowered (Chapter 6).

In the metric space layer every comparison node is connected to every other comparison node, and to the object layer node that represents the appropriate object. Each inhibition node only has a single link to its associated comparison node. The weight on the link between an inhibitory node and its comparison node, is proportional to the negative inverse of the standard deviation and total number of points that were clustered together, in the metric space that is represented by the nodes in question. This value is scaled by the maximum initial value of a comparison node. If i represents the current metric space, and $contrib_i$ is the total number of clustered points in metric space i, then the inhibition node's weight is:

$$inhibit_i = \frac{-max\ comparison\ node\ value(10)}{(standard\_deviation_i + 1) \times (contrib_i + 1)}$$

The weights on the connection between two comparison nodes is proportional to the relative importance of one of the comparison nodes, and inversely proportional to the importance of the other. The importance of a metric space is measured by comparing the total size of all the clustering sets used to form points in the metric space, divided by the combined size of all the clustering sets used in forming the points belonging to an entire intersection (ten metric spaces). That is the importance of a metric space i, where N is the number of metric spaces (10), is:

$$imp_i = \frac{contrib_i}{\sum_{j=0}^{N} contrib_j}$$

If a metric space has no importance to an object ($imp_i = 0$), then it plays no part in the recognition process. Thus the weights on its connections to all other comparison nodes are set to zero, this also avoids the problem of a division by zero. The weight on the connection from a node to itself is set at one. Assuming two comparison nodes, i and j, have some importance, then the weight w_{ij} on the connection from i to j is scaled depending on the objects and metric spaces the nodes represent. Nodes belonging to the same object reinforce each other, while nodes representing different objects inhibit each other by the same amount when they are associated with different metric spaces, or to a greater degree if they are associated with the same metric space. If the object which a node i represents is denoted by obj_i, and the metric space it represents denoted vs_i, then the weight from any node i to any node j is given by:

$$w_{ij} = \begin{cases} 1 & , if\ i = j \\ \alpha \times \frac{imp_i}{imp_j} & , if\ obj_i = obj_j \\ -\beta \times \frac{imp_i}{imp_j} & , if\ obj_i \neq obj_j\ and\ vs_i = vs_j \\ -\alpha \times \frac{imp_i}{imp_j} & , if\ obj_i \neq obj_j\ and\ vs_i \neq vs_j \end{cases}$$

where both α and β are positive, and $\alpha < \beta$. By default α and β are set to `0.01` and `0.1` respectively. Preliminary experimentation suggested that these values are small enough not to result in a node being overwhelmed, while large enough such that a node's value is significantly influenced by the other nodes. Note that under this scheme the weight on the connection between two nodes is not necessarily the same in both directions, $w_{ij} \neq w_{ji}$. In fact, unless they are zero, the connection weights are inversely symmetric, $w_{ij} = \frac{1}{w_{ji}}$. This causes relatively important metric spaces to overwhelm the the more inconsequential nodes, even if these nodes contain a large value.

Every comparison node in the metric space layer connects to a single node in the object layer. This connection is between nodes representing the same object. Thus each node in the object layer receives input from ten comparison nodes, one for each metric space in an object class' intersection. The weight on this connection is the relative importance of the relevant metric space to the object in total. This is calculated in an identical manner to the importance of a metric space defined in the paragraph above. Hence the weights on the ten connections to a single object node sum to the value 10. Thus, normally, the value of an object node will not exceed 100. None of the object nodes are interconnected.

The program `weight` initialises and assigns weights to the connections in the network defined in this section from a number of probabilistic intersections. Figure 5.2 is a density plot of the weights on the comparison nodes after `weight` has set the weights so as to recognise people and cars. The white boxes denote positive weights, black boxes represent negative weights and grey boxes are weights which are zero. The first ten nodes on each axis (0...9), represent the intersections of the car object class, while the next ten are the person object class. The first five rows/columns in both object classes are the intersections derived from that object class' augmented AS, the binary relationship space derived intersections form the next five. The five intersections from each metric space are in the order, full intersection, knot ignored, sweep ignored, length ignored, and "mapping multiplicity" ignored. Thus the third row/column represents the intersection formed from the car's augmented AS when the sweep parameter is ignored, and the nineteenth row/column is the person object class' binary relationship space with the length ignored. As can be seen, the weight from a node to itself is set to `1`, similarly columns and rows three, eight, thirteen, eighteen and twenty are set completely to zero, thus playing no part in the recognition process. Also visible is the partitioning of the comparison nodes into excitory/inhibitory areas. The two white areas represent comparison nodes connecting to other nodes in the same object and reinforcing each other, the black areas show comparison nodes connecting to different objects and inhibiting them. The inverse symmetry of the nodes across the line $i = j$ can also be discerned. Furthermore those squares representing the connections between nodes in different objects, but from the same metric space (squares along the lines $i = j + 10$ and $i + 10 = j$), are clearly darker than the other squares in the same quadrant.

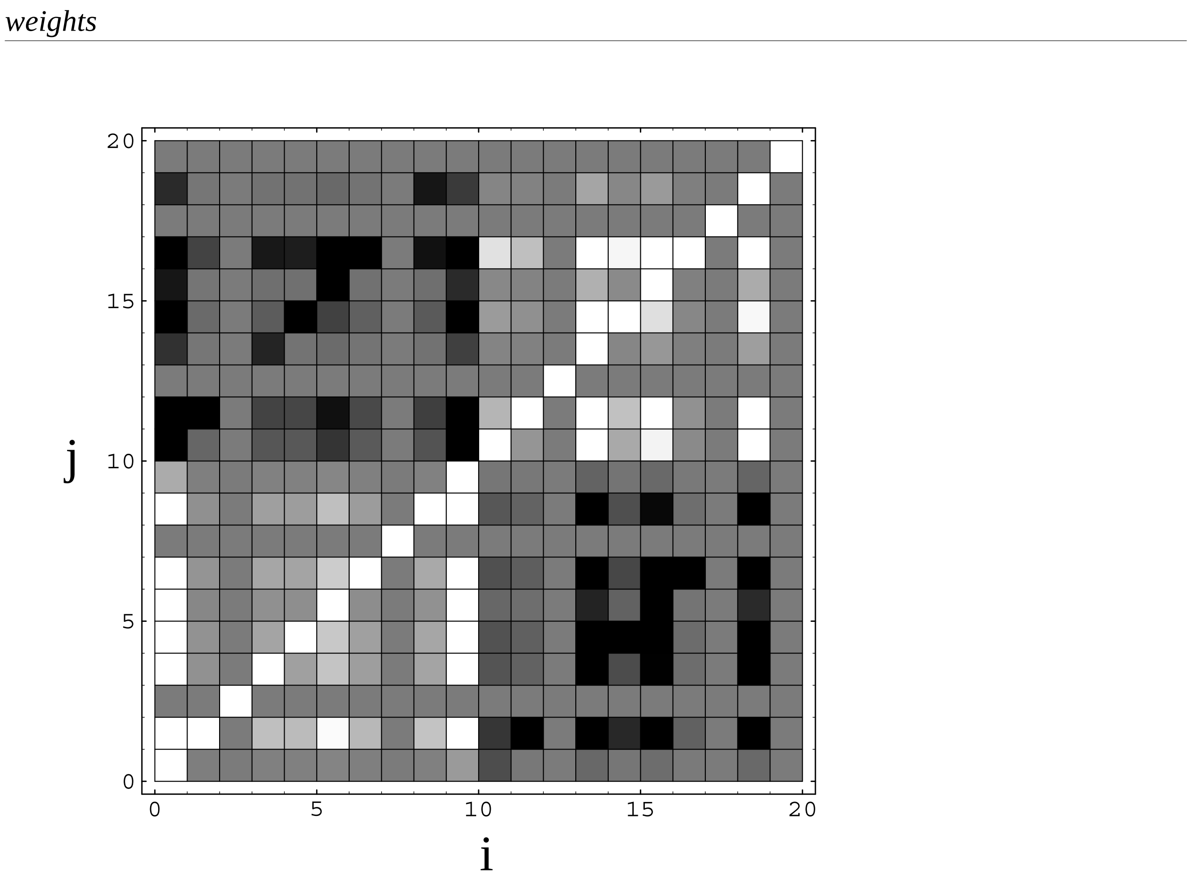


Figure 5.2: Density plot showing weights between comparison nodes i and j

5.2 The Recognition Process

The connectionist network detailed in Section 5.1 is designed to facilitate the fast recognition of an unknown object. When an unknown object is input at the point layer, values propagate through the network to the object layer nodes.

The final value contained within an object layer node will be proportional to the probability that the unknown object is a member of the object class to which that node is associated. Table 5.1 contains the pseudo-code for the recognition process within the network. The update function for nodes synchronously replaces the current value of the node, with the sum of the values in the nodes to which it is connected, multiplied by the weight on the connection. Synchronous updating signifies that the new value of a node should only be placed in the node after all nodes have been updated. That is if the node to be updated is n_i, there are N nodes connected to n_i and the weight on the link between n_i and n_j is w_{ij}, then the new value of node n_i is :

$$n_i = \sum_{j=1}^{N} n_j w_{ij}$$

Prior to an unknown object being placed inside the network, it must be first converted into a set points in its augmented AS and binary relationship space. These points are then put into individual sub-networks within the point layer. If the sub-network into which a set of points is inserted represents a metric space that has one of its parameters set to zero, this parameter is arbitrarily set to zero in the set of input points.

```
activate(direct_matches)
update_nodes(metric_space_nodes)

activate(indirect_matches)
update_nodes(comparison_nodes)

activate(inhibitory_nodes)
update_nodes(comparison_nodes)

repeat
    activate(comparison_nodes)
    update_nodes(comparison_nodes, object_layer)
until (identify(object_layer))
```

Table 5.1: Pseudo-code for the recognition process

The first step in recognising an unknown object is to map the points onto their equivalent sub-network hypercubes. The internal value of these hypercubes is then set to one; that is they are activated, and the metric space layers nodes are updated. This represents a direct match between a point in the unknown object and a point in one of the intersections. After the initial update, any hypercubes that remain unactivated and have a connection to a comparison node are given a second chance to activate. If such a node has an adjacent neighbour that is activated, then it will fire a value up to its comparison node, in the second update. However the value it sends is only half the weight on the connection between them. This indirect match represents a point in the unknown object which may be a point from an intersection, although with a lesser probability than a direct match, so a lesser value is dispatched to the comparison node. To ensure a large number of indirect matches do not overwhelm a comparison node (hypercubes in the sub-network representing the binary relationship space have 3^7-1 adjacent hypercubes), only two indirect matches are allowed per hypercube. Thus the maximum value a hypercube can send to a comparison node during both updates, is the weight on the connection between them.

Following the firing of the point layer hypercubes, and the metric space layer nodes being updated, the inhibitory nodes fire. The value contained by an inhibitory node is the absolute difference between the input it receives, and the average number of points in the metric space and object it represents. When these nodes fire, they inhibit their associated comparison node by an amount proportional to the difference between the number of points in the unknown object and the number of points normally expected in that object the node represents. Thus these nodes penalise an unknown object for being too small or large compared to the object class under consideration.

Once the inhibitory nodes have fired, then the comparison nodes are activated and are updated along with the object layer nodes. Since the weight on the connection from a node to itself is one, the result of updating these nodes is to modify its current value depending upon which object class is most consistent with the unknown object. This is achieved by comparing the importance of the metric space represented by the node, against all the other metric spaces. The metric spaces will strengthen the other nodes from the same object class, by an amount proportional to its relative importance, and weaken those from alternative objects by a similar amount. However if the value of a node is negative it will do the reverse. Additionally, nodes which represent the same metric space, but different object classes will affect each other to a greater degree than other nodes. This forces one node to eventually dominate all the others representing the same metric space. Furthermore, since the comparison nodes are updated synchronously, all the nodes that have any importance to recognition are given an opportunity to affect other nodes before being overwhelmed by nodes of greater importance.

After updating the comparison nodes, the object layer nodes are updated, and then examined to determine whether the conditions for object identification have been attained. Unknown objects are identified by comparing each object layer node's value against a set of cut-off values. This results in the object class represented by the node, being placed into one of four identification categories, relative to the unknown object, namely 'definitely is', 'probably not', 'definitely not' or 'undecided'. The category thresholds affect the strictness of the recognition process. The higher the cut-off values, the greater the required similarity between between an unknown object and an internal object class before identification can take place. Default cut-offs are: nodes with values above `50` are 'definitely is'; between `20-50` are 'undecided'; between `10-20` are 'probably not'; and anything below `10` is 'definitely not'. An object layer node of value `50` signifies that the unknown object has at least half the important recognisable features in common with the node's object class's intersection, while a value of `10` means there is only a tenth of such features in common. The effect of modifying the default values is examined in the experimental section of this thesis, Chapter 6. Although not explicitly referenced by any of the identification criteria, the 'probably not' category is used to separate the 'definitely not' and 'undecided' classifications, thus possibly forcing continued processing if the unknown object is in this category.

If any of the nodes fall into the 'definitely is' category, then the unknown object is identified as a member of that node's object class. However when many nodes fit into the 'definitely is' class, then the recognition process will not attempt to differentiate them, returning that it can't decide between the multiple 'definitely is' object classes. The unknown object is identified as defaulting to an object class when one node is 'undecided', while the remainder are in the 'definitely not' category. The unknown object is considered to be none of the classes learnt by the network, when all of the object nodes are in the 'definitely not' category.

If the values of the object nodes do not fit into any of the identification categories, then the object layer nodes are considered inconclusive, and more processing is required. Hence the process goes into a loop, the comparison nodes are reactivated, then the comparison and object layer nodes are updated again. This continues until the criteria for one of the identification categories has been met, or a set threshold number of iterations is reached. The default threshold value is `10`. Lower than this the comparison nodes may not result in an identification, any higher and an urecognisable object (one not a member of the learnt object classes), may be incorrectly classified.

The program `recog` applies the above scheme in attempting to recognise an unknown object, using a user-specified weights file. The user may also specify the cut-offs for the identification categories, the threshold number of iterations and the verbosity level, or use the internal defaults. Table 5.2 shows the maximum verbosity output from `recog` when the network was taught the object classes 'person' and 'car', then required to recognise a headless person (Appendix C). After naming each object class, `recog` displays the number of iterations performed, the initial value and the final value of the object node. Thus in Table 5.2, the person object class has an initial value of `43.727196`, and after five iterations through the recognition process this becomes `50.225273`. The list of number after each class is the initial and final values of all the comparison nodes associated with an object, only with the node's relative importance in brackets. The comparison nodes are listed in an identical order to those in the density plot of intra comparison node connections, Figure 5.2. Note that the comparison nodes that have a relative importance of zero, match the zero rows/columns in Figure 5.2.

```
Known object classes: car person

car: 5:20.703720->-14.068951

        -5.600000->-25.595201 (0.177778)
        2.973705->0.416750 (2.577778)
        -28.000000->-28.000000 (0.000000)
        5.082352->3.264205 (1.511111)
        3.687689->0.586398 (1.600000)
        -0.421050->-2.676038 (0.800000)
        -0.205130->-6.418312 (1.688889)
        -8.000000->-8.000000 (0.000000)
        1.007580->-0.997563 (1.422222)
        -1.333330->-9.289745 (0.222222)

person: 5:43.727196->50.225273

        2.270250->4.038348 (1.468750)
        4.882958->4.439179 (2.218750)
        -6.666666->-6.666666 (0.000000)
        6.073780->2.429907 (0.343750)
        5.748950->5.483582 (1.218750)
        -1.290324->3.380900 (0.468750)
        4.907302->5.620343 (3.906250)
        -40.000000->-40.000000 (0.000000)
        5.066664->9.031640 (0.375000)
        -40.000000->-40.000000 (0.000000)

person_a.decap.asd is in class person(5:43.727196->50.225273)
```

Table 5.2: Verbose output from the `recog` program

CHAPTER 6

Experimentation

6.1 Results

In order to test the effectiveness of the connectionist network and the recognition process (Chapter 5), numerous experiments were conducted on diverse test data. A number of people drew a collection of 30 objects, which were converted into the extended representation, and classified into one of six categories. The categories were: 1, people, objects that are clearly people; 2, acceptable people, objects that could be people (eg. a decapitated person); 3, almost people, objects not quite acceptable as people (eg. a limbless person); 4, cars; 5, almost cars; and 5, miscellaneous, anything that didn't fit into the other categories. Appendix C shows all 30 input objects.

The `recog` program was executed twelve times on every test object, on each occasion modifying one variable, the clustering algorithm, the size of the intersections, or the thresholds of the identification categories. All other attributes remained constant throughout the tests, including the object classes learnt, which was always cars and people. The size of the probabilistic intersections was increased by including in the intersection more objects representative of the relevant object class. This was done in order to test whether only using the most important points from an intersection is a suitable method, for ensuring the number of recognisable points in an object class is not larger then the members of that class. The effect of increasing the thresholds between identification categories was tested by boosting the cut off between 'definitely not' and 'probably not' to `15`, and raising the thresholds between other categories by `10`. Both clustering algorithms were used to determine whether blobbing is an acceptable approximation to maximal cliques.

The output from the twelve experiments on `recog` are given in Appendix D, and summarised in Tables 6.1 through 6.3. Each table displays the results from different sized intersections, Table 6.1 shows intersections determined from two objects per object class (Person A/B, and Car 1/2), Table 6.2 adds another object to the intersection (Person C and Car 3), and Table 6.3 uses four objects (adding Person D and Car 4). The tables give the algorithm and threshold, before detailing the number of errors made by `recog`. The errors are split into two categories: close, those errors that are almost right, such as not recognising an acceptable person; and wrong, unacceptable errors, such as not recognising a person, or identifying a turtle as a car. The total column is the combined number of wrong and close errors, out of the 30 test objects.

Algorithm	Cut off	Close	Wrong	Total
Cliques	Default	2	1	3
Blobs	Default	1	2	3
Cliques	High	1	2	3
Blobs	High	0	3	3

Table 6.1: 2 object intersection results

Algorithm	Cut off	Close	Wrong	Total
Cliques	Default	1	5	6
Blobs	Default	3	5	8
Cliques	High	2	4	6
Blobs	High	1	3	4

Table 6.2: 3 object intersection results

Algorithm	Cut off	Close	Wrong	Total
Cliques	Default	1	7	8
Blobs	Default	2	8	10
Cliques	High	1	4	5
Blobs	High	1	6	7

Table 6.3: 4 object intersection results

The `recog` program was written in ANSI C and compiled under gcc, using the `-O3` flag, on a SPARCServer5 running SunOS. Table 6.4 shows the output of the Unix `time` command when the various programs used in the recognition process were executed. The size column displays the number of objects used in determining the intersection, the algorithm column shows the clustering algorithm, and the total column is the combined time taken by all programs. It should be noted that the time for the `recog` program is the time required to identify all 30 test objects. All values are in terms of user seconds.

Size	Algorithm	`inter`	`weight`	`recog`	Total
2	Blobs	0.63	0.23	0.74	1.6
2	Cliques	6.13	0.35	0.84	7.32
3	Blobs	1.39	0.32	0.77	2.48
3	Cliques	58.39	0.52	0.78	59.69
4	Blobs	2.25	0.35	0.86	3.46
4	Cliques	129.37	0.63	0.91	130.91

Table 6.4: User time required by each program

6.2 Discussion

The results show that the method devised is particularly fast, and under some conditions reasonably accurate. In the best case the `recog` program requires just a fortieth of a second each to correctly identify 90% of the test objects, with 6.6% of the remainder being close errors. However certain experiments produced up to 30% errors, although still very quickly.

Unsurprisingly maximal cliques proves to be more accurate than blobs, although often requiring orders of magnitude more time. However this extra time cost is only necessary once, as the `weight` and `inter` program's speed are independent of the clustering algorithm. Thus the extra time consumed is probably worthwhile considering the increased accuracy, unless the recognition system is required to learn object classes "on the fly".

A clear trend evident from the results is that the accuracy decreases with the increasing size of the intersection employed. This suggests that the method of simply ignoring all points below a certain threshold position is flawed. The four object clique intersection is the only intersection to have more points in any metric space than three standard deviations above the average. The only intersections below the average number of points are the two object intersections. Thus the problem most probably lies in the threshold being set to high, and therefore there are too many recognisable points each with its relative importance set unnecessarily low.

Raising the cutoff values for the identification classes appears to improve the performance of the recognition process, especially in the larger intersection experiments. This is because the higher thresholds force more stringent requirements upon the unknown object if it is to be identified. However doing this changes the nature of the erroneous identifications. Relatively low thresholds effectively expand the recognition set, so that `recog` will recognise all the objects that are members of the learnt object classes, and also identify some objects which are in none of these classes. Increasing the thresholds causes `recog` to incorrectly identify objects not belonging to any set, but also not recognise some objects which are members of known classes, equivalent to shrinking the recognition set. These effects of changing the thresholds are more discernible in the output contained in Appendix D. The errors in the default threshold experiments are mainly due to incorrectly recognising objects in the almost people, almost cars and miscellaneous categories as people or cars. Errors in the high threshold experiments are frequently the result of not identifying objects in the person or car categories.

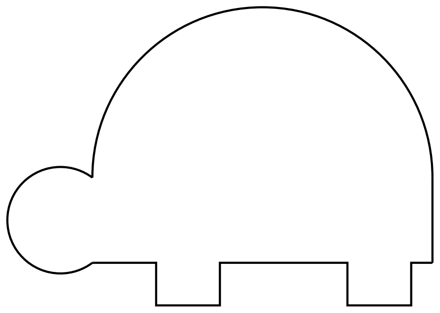


Figure 6.1: A turtle

One object that is consistently incorrectly identified, is the turtle object (Figure 6.1). All experiments recognised the turtle as a car with a reasonable degree of certainty. This is due to the method of recognition. The network will first attempt to identify an unknown object as one of the learnt object classes. Only if `recog` can not force an object into the known classes will the program return that the object is not something it can recognise. In the case of the turtle, `recog` is certain the object is not a person and thus attempts to recognise it as a car, after a number iterations in the metric space layer, it is made to fit into this category. However, if the network is taught the turtle object class, then this object is correctly identified.

CHAPTER 7

Conclusion

7.1 Conclusion

The purpose of this project was the construction of a scheme for object representation, and the devising of a connectionist network for object recognition, based upon this scheme. Experimentation on the connectionist network, described within this thesis (Chapter 5), produces good results, with around 90% of the test data correctly recognised, under acceptable conditions (Chapter 6). Additionally, the simple fact that the recognition scheme correctly identifies the majority of the input objects, suggests that the comparison methods developed (Chapter 4) are able to capture the recognisable features of an object class, and then differentiate between these classes. This also indicates that the object representation scheme devised (Chapter 3), is capable of suitably encoding an object.

Importantly, the recognition process within the network is very fast, requiring hundredths of a second to identify an object (Chapter 6). Although this is after the probalistic intersections of a set of objects representing an object class have been generated. The process of extracting a set of recognisable features using the NP-Complete maximal clique algorithm dominates the time demands of the system, usually by orders of magnitude. However, the much faster polynomial blobbing clustering algorithm can be used, greatly speeding the recognition process, but at the cost of lessened accuracy. The trade-off generally favours the slower, more precise technique of searching for maximal cliques, due to the isolated nature of generating intersections within the system. Furthermore, the design of the connectionist network is inherently parallel, and thus if implemented using the concurrent paradigm, has the capability of requiring even less time. The network is also easily modifiable. To add a new object class, the relevant hypercubes, inhibitory nodes, comparison nodes and object nodes need to be initialised, and their weights set accordingly. None of the existing nodes or weights have to be changed. Deleting an object class is equally easy, only the nodes and weights associated with that object class need to be removed.

At present the system has associated with it certain limitations. The recognition method assumes that the input will be perfectly formed, with no syntactic or semantic noise. Thus the input must be suitably pre-processed, a specification that can not currently be automated, and instead must be performed by "hand". The prerequisite that the position of edgeset be consistently defined is particularly constricting as it forces objects of the same class to conform to a set prototype and necessitates that this template be "remembered" for future reference. Another constraint upon the system is the requirement that a relatively small number of objects be used in determining the recognisable features of an object class. However this can be overcome but adjusting the thresholds on the identification categories, or by utilising the extensive memory capacity of the network and teaching it every possible object class that will likely be encountered.

7.2 Further Work

Pre-eminent among the extensions that should be made to the work detailed in this thesis, is the development of an image pre-processing system. This system would need the capability to detect edges in a real world image. Then convert them into semantically disjoint edgesets, consisting of arcs which could be input by the recognition system. Once combined with the identification system detailed in this thesis, the result would be a complete object recognition system, possibly with real world applications, and greatly accelerating the experimentation process.

Further experimentation and examination should be conducted upon the identification process, performed with extensive test data drawn from more object classes. The aim of this should be the development of methods for automatically setting the thresholds between identification categories, and other system variables, that are dependent on the properties of the learnt object classes. Work should also focus on solving the edgeset position problem, devising a method of encoding them so as to be invariant with respect to the position from which they are measured. Also, the recognition system could only benefit from the examination of higher order relationship spaces and the coding of a concurrent version of the recognition process.

A possible future application could be in the realm of character recognition. Simple image transformations such as dilation, closing and edge detection (See Smith [21]), should produce edges relatively easy to translate into arcs, and semantically disjoint edgesets.

References

1. N. Ayache and O. D. Faugeras. Hyper: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, 1986.

2. A. Bergman and P. G. Mulgaonkar. Neural networks for address block ranking: A comparison with classical techniques. In *Proceedings of USPS third Advanced technology conference*, 1988.

3. J. Ross Beveridge. *Local Search Algorthims for Geometric Object Recognition: Optimal Correspondence and Pose.* PhD thesis, University of Massachusetts at Amherst, 1993.

4. T. Caelli, M. Ferraro, and E. Barth. Aspects of invariant pattern and object recognition. In *Neural Networks for Perception: Human and Machine Perception*, pages 234–247. Academic Press, Boston, 1992.

5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness.* W. H. Freeman and Company, San Fransisco, 1979.

6. Rafael G. Gonzalez and Richard E. Woods. *Digital Image Processing.* Addison-Wesley, Reading, Massachusetts, 1992.

7. P. Hadingham. A regular language for edge description and classification. Technical Report 9, University of Western Australia, Perth, 1988.

8. P. Hadingham. Symbolic description of edges using a geometric relaxation technique. *Pattern Recognition Letters*, 7:173–179, 1988.

9. P. Hadingham. Formal systems in artificial intelligence: an illustration using semigroup, automata and language theory. *Artificial Intelligence Review*, 4:3–19, 1990.

10. P. Hadingham. Visual object representation using parametrized arcs. In *Proceedings of the Australian and New Zealand Conference on Intelligent Infomation Systems*, pages 323–327, Perth, 1993.

11. J. Hopfield and D. Tank. Computing with neural circuits: a model. *Science*, 233:625–633, 1986.

12. R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1168–1180, 1989.

13. D. H. Hubel. *Eye, Brain, Vision.* Scientific American Library, New York, 1988.

14. J. Mantas. Methodologies in pattern recognition and image analysis - a brief survey. *Pattern Recognition*, 20(1):1–6, 1987.

15. L. Miclet. *Structural Methods in Pattern Recognition.* North Oxford Academic, London, 1986.

16. D. W. Murray. Model-based recognition using 3d shape alone. *Computer Vision, Graphics and Image processing*, 40:250–266, 1987.

17. T. Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, Berlin, 1977.

18. Edward M. Reingold, Jurg Nievergelt, and Narsignh Deo. *Combinatorial Algorithms: Theory and Practice*, chapter 8. Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

19. J. S. Rohl. Personal Communication, October 1995.

20. R. Serra and G. Zanarini. *Complex Systems and Cognitive Processes.* Springer-Verlag, Berlin, 1987.

21. R. Smith. Computer processing of line images: A survey. *Pattern Recognition*, 20(1):7–15, 1987.

22. P. Suetens, P. Fua, and A. Hanson. Computational strategies for object recognition. *ACM Computing Surveys*, 24(1):5–61, 1992.

APPENDIX A

Original Honours Proposal

Title: Connectionist Network Architecture For Object Recognition

Student: Charles Robert Tolhurst Cordingley

Supervisor: Dr Paul Hadingham

Aim: Object recognition is an important aspect of many systems in artificial intelligence, such as vision and robotic systems.

Hadingham [10] suggested a scheme for representing edge-based objects as parametrized arcs, a representation which remains untried. Hadingham's model will be extended and used as the formal basis for this project.

The final goal of the project will be to produce a system, based around a neural network structure and utilising the extended representation sheme, capable of recognising two-dimensional planar edge images of objects.

Method: Hadingham's representation scheme describes an edge-based object as a word in a context free grammar. This language converts edges into a set of arcs determined by the parameters knot angle, sweep angle and arc length.

Hence a set of vector spaces defined by the arc parameters can be constructed, with each arc representing a point in the vector space. These vector spaces can be used to express the relationships between the arcs.

The two methods of representing the object, as a set of parameter spaces and in terms of a context free grammar, will be investigated. A neural net will be trained to recognise edge images utilising the representation scheme resulting from the investigation. Where ever possible real image data will be used in preference to manufactured data. However the project will not involve transforming real images into appropriate edge images; rather, the edge images will form the starting point.

The project will be constructed using C or C++ on a UNIX platform.

APPENDIX B

Psuedo-code for Maximal Clique Search

This pseudo-code is taken from the book "Combinatorial Algorithms" by Reingold, Neivergelt and Deo [18].

$adj(f)$ returns all the nodes adjacent to f.
V is the set of all pairs of adjacent nodes.
S is the clique currently being examined.
N is the set of all the nodes that can be added to S.
D is the set of nodes that may not be added to S.

```
$S \leftarrow \emptyset$
CLIQUE($V, \emptyset$)

procedure CLIQUE($N, D$)
    if $N \cup D = \emptyset$ then output $S$, which is a maximal clique
    else
        if $N \neq \emptyset$ then [explore first s ubtree]
            $f \leftarrow$ vertex in $N$
            EXPLORE($f$)
            [explore remaining subtrees]
            while $N \cap (V - adj(f)) \neq \emptyset$ do $v \leftarrow$ vertex in $N \cap (V - adj(f))$
            EXPLORE($v$)
        else [if $N = \emptyset$ and $D \neq \emptyset$ the n no new cliques]
    return

procedure EXPLORE($u$)
    $N \leftarrow N - u$
    $S \leftarrow S \cup u$
    CLIQUE($N \cap adj(u), D \cap adj(u)$)
    $S \leftarrow S - u$
    $D \leftarrow D \cup u$
    return
```

APPENDIX C

Data

C.1 People

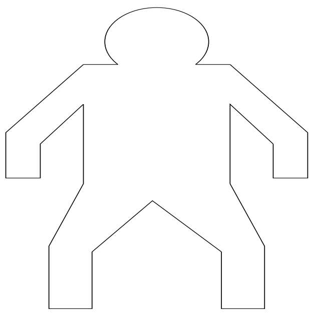


Figure C.1: `person_a`

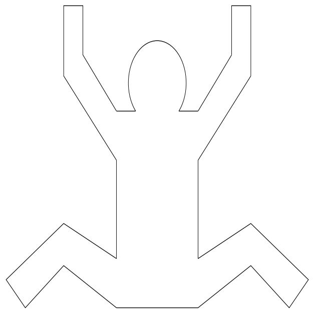


Figure C.2: `person_b`

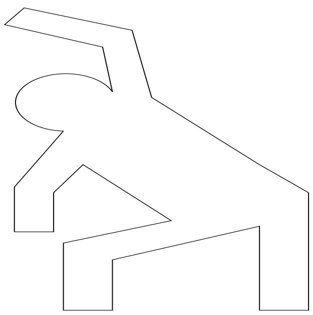


Figure C.3: `person_c`

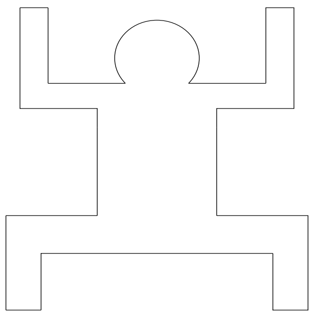


Figure C.4: `person_d`

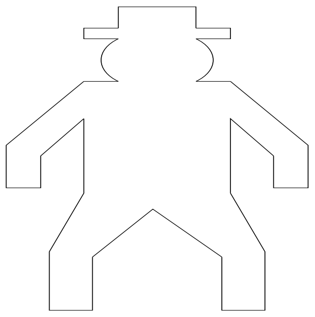


Figure C.5: `person.hat`

C.2 Acceptable People

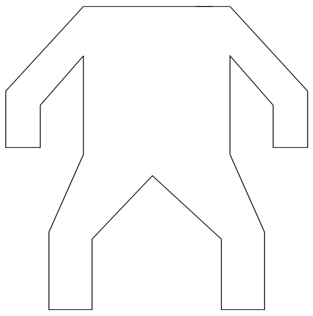


Figure C.6: `person.decap`

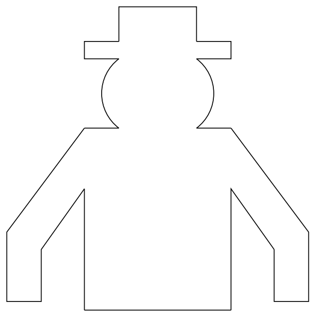


Figure C.7: person.hat.legless

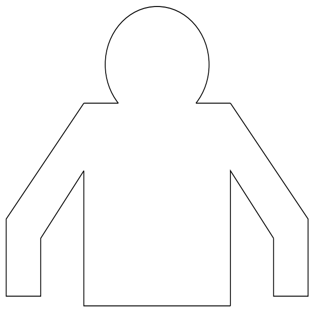


Figure C.8: `person.legless`

C.3 Almost People

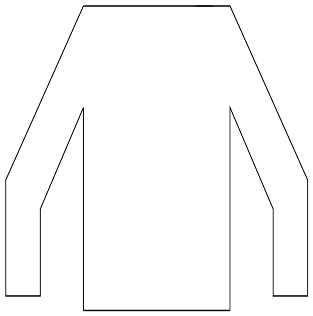


Figure C.9: person.decap.legless

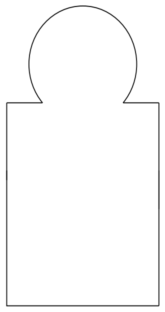


Figure C.10: person.limbless

C.4 Cars

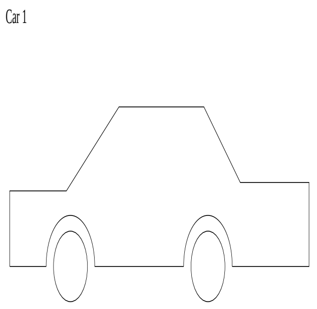


Figure C.11: `car1`

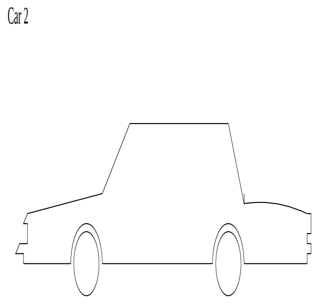


Figure C.12: `car2`

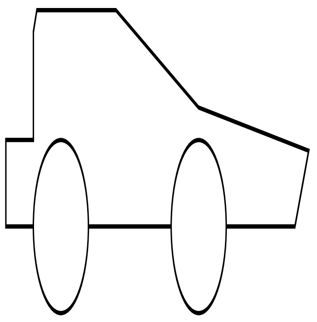


Figure C.13: `car3`

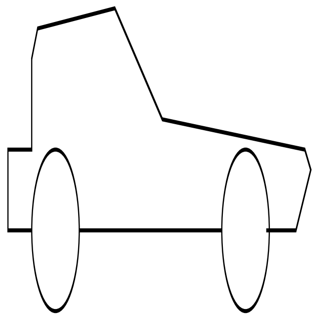


Figure C.14: `car4`

C.5 Almost Cars

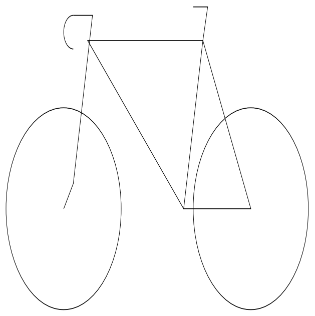


Figure C.15: `bike`

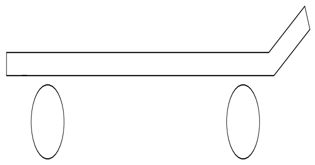


Figure C.16: `skate`

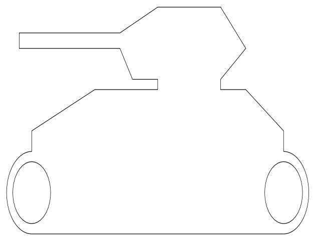


Figure C.17: `tank`

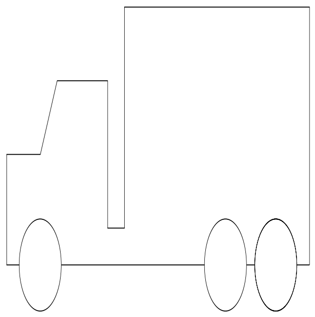


Figure C.18: `truck`

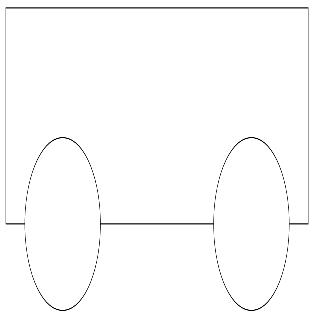


Figure C.19: `box.on.wheels`

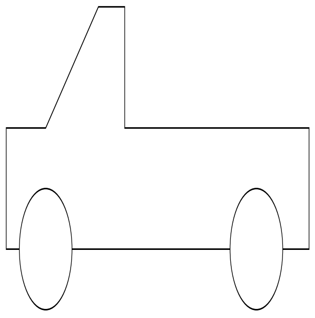


Figure C.20: `ute`

C.6 Misc

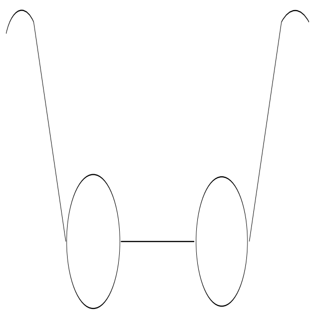


Figure C.21: `glasses`

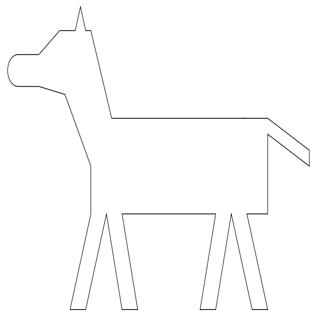


Figure C.22: `horse`

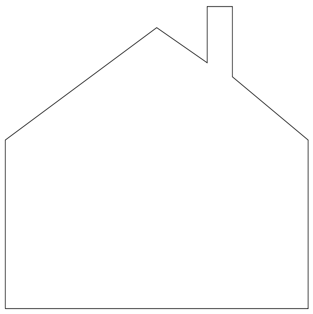


Figure C.23: `house`

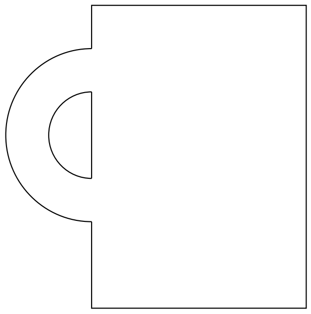


Figure C.24: `mug`

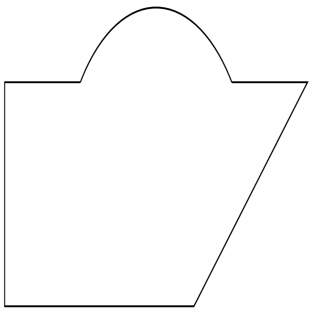


Figure C.25: `example`

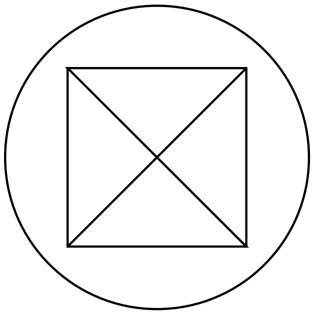


Figure C.26: `np`

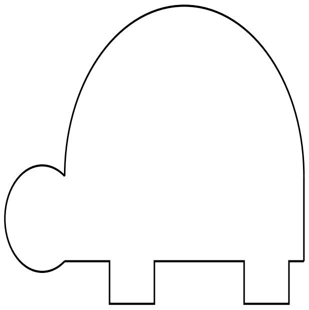


Figure C.27: `turtle`

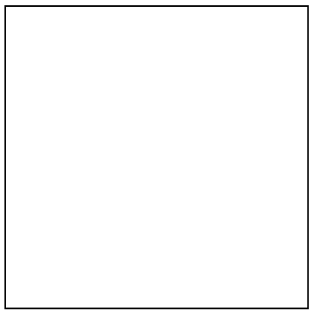


Figure C.28: `square`

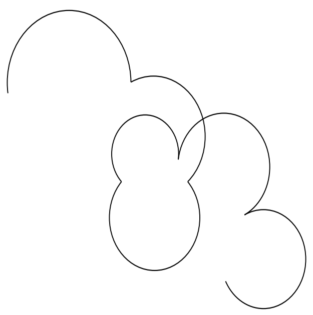


Figure C.29: `test2`

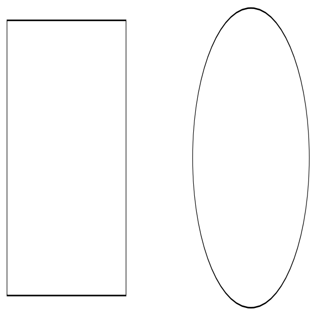


Figure C.30: `test1`

APPENDIX D

Raw Results

D.1 2 Object Blob Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is in class car(10:20.790304->58.249699)
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(3:36.276428->51.297737)
car2.asd is in class car(0:73.398804->73.398804)
car3.asd is in class car(4:29.664864->51.932648)
car4.asd is in class car(4:37.009796->53.386311)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class person(0:58.328518->58.328518)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:77.915894->77.915894)
person_a.decap.asd is in class person(8:36.565948->54.531437)
person_a.decap.legless.asd is in none of the known classes
person_a.hat.asd is in class person(8:33.247131->52.850296)
person_a.hat.legless.asd is in class person(6:44.198997->50.814320)
person_a.legless.asd is in class person(10:37.230747->54.743706)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:72.568298->72.568298)
person_c.asd is in class person(0:67.737350->67.737350)
person_d.asd is in class person(5:42.296661->52.628849)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is undecided, probably none (11)
```

```
turtle.asd is in class car(3:33.507362->59.655163)
ute.asd is in none of the known classes
```

D.2 2 Object Blob Intersection with High Thresholds

```
Known object classes: car person

bike.asd is in none of the known classes
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(5:36.276428->70.235664)
car2.asd is in class car(0:73.398804->73.398804)
car3.asd is in none of the known classes
car4.asd is in class car(5:37.009796->61.453510)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class person(6:58.328518->64.231483)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:77.915894->77.915894)
person_a.decap.asd is in class person(9:36.565948->62.882622)
person_a.decap.legless.asd is in none of the known classes
person_a.hat.asd is in class person(9:33.247131->61.296986)
person_a.hat.legless.asd is in class person(8:44.198997->63.597637)
person_a.legless.asd is in class person(11:37.230747->63.523705)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:72.568298->72.568298)
person_c.asd is in class person(0:67.737350->67.737350)
person_d.asd is in class person(7:42.296661->66.958633)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
turtle.asd is in class car(4:33.507362->72.718231)
ute.asd is in none of the known classes
```

D.3 2 Object Clique Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is in none of the known classes
```

```
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(4:37.085480->53.263283)
car2.asd is in class car(0:70.995682->70.995682)
car3.asd is in class car(5:30.649530->57.969162)
car4.asd is in class car(4:37.132614->51.560436)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is undecided, probably none (11)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:80.841034->80.841034)
person_a.decap.asd is in class person(5:43.727196->50.225273)
person_a.decap.legless.asd is in class person(11:32.208321->54.671303)
person_a.hat.asd is undecided, probably none (11)
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(0:66.077614->66.077614)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:75.841003->75.841003)
person_c.asd is in class person(0:73.315956->73.315956)
person_d.asd is in class person(0:58.747185->58.747185)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
turtle.asd is in class car(3:32.610401->50.246571)
ute.asd is in none of the known classes
```

D.4 2 Object Clique Intersection with High Thresholds

```
Known object classes: car person

bike.asd is in none of the known classes
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(5:37.085480->61.303257)
car2.asd is in class car(0:70.995682->70.995682)
car3.asd is in class car(6:30.649530->68.738571)
car4.asd is in class car(6:37.132614->68.408600)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in none of the known classes
house.asd is in none of the known classes
```

```
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:80.841034->80.841034)
person_a.decap.asd is in class person(7:43.727196->62.155266)
person_a.decap.legless.asd is in none of the known classes
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(0:66.077614->66.077614)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:75.841003->75.841003)
person_c.asd is in class person(0:73.315956->73.315956)
person_d.asd is in class person(2:58.747185->61.225063)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
turtle.asd is in class car(5:32.610401->71.078468)
ute.asd is in none of the known classes
```

D.5 3 Object Blob Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is undecided, probably none (11)
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(7:34.049896->50.191238)
car2.asd is in class car(0:65.820503->65.820503)
car3.asd is in class car(0:54.287174->54.287174)
car4.asd is in class car(0:58.931454->58.931454)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class person(0:63.581692->63.581692)
house.asd is in class person(7:28.937382->50.420223)
mug.asd is in class car(7:20.147560->55.132381)
np.asd is in class car(6:20.272305->50.948353)
person_a.asd is in class person(0:81.766708->81.766708)
person_a.decap.asd is in class person(3:49.007011->50.359756)
person_a.decap.legless.asd is in class person(8:41.565071->53.990631)
person_a.hat.asd is undecided, probably none (11)
person_a.hat.legless.asd is undecided, probably none (11)
person_a.legless.asd is in class person(0:56.596699->56.596699)
person_a.limbless.asd is in none of the known classes
```

```
person_b.asd is in class person(0:61.926476->61.926476)
person_c.asd is in class person(0:82.011330->82.011330)
person_d.asd is in class person(3:48.920895->50.323833)
skate.asd is in class car(6:25.879457->51.938522)
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is undecided, probably none (11)
turtle.asd is in class car(3:35.727524->50.954281)
ute.asd is in none of the known classes
```

D.6 3 Object Blob Intersection with High Thresholds

```
Known object classes: car person

bike.asd is in none of the known classes
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(9:34.049896->67.758911)
car2.asd is in class car(0:65.820503->65.820503)
car3.asd is in class car(4:54.287174->64.619019)
car4.asd is in class car(3:58.931454->61.386753)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class person(0:63.581692->63.581692)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:81.766708->81.766708)
person_a.decap.asd is in class person(6:49.007011->64.081772)
person_a.decap.legless.asd is in class person(9:41.565071->61.655262)
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(5:56.596699->62.856968)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:61.926476->61.926476)
person_c.asd is in class person(0:82.011330->82.011330)
person_d.asd is in class person(6:48.920895->64.181541)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
```

```
turtle.asd is in class car(5:35.727524->70.173035)
ute.asd is in none of the known classes
```

D.7 3 Object Clique Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is in class person(9:33.233257->54.629860)
box.on.wheels.asd is in none of the known classes
car1.asd is in class car(11:22.787630->53.329967)
car2.asd is in class car(0:74.684265->74.684265)
car3.asd is in class car(0:56.374756->56.374756)
car4.asd is in class car(0:58.503597->58.503597)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class car(0:61.416080->61.416080)
house.asd is in class person(9:21.336691->56.192280)
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:88.259048->88.259048)
person_a.decap.asd is in class person(3:47.856052->52.224449)
person_a.decap.legless.asd is in class person(5:43.455360->53.719860)
person_a.hat.asd is undecided, probably none (11)
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(0:75.710846->75.710846)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:71.333931->71.333931)
person_c.asd is in class person(0:85.099655->85.099655)
person_d.asd is in class person(0:64.050941->64.050941)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in class person(9:20.341282->58.250366)
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is undecided, probably none (11)
turtle.asd is in class car(1:46.972878->50.330845)
ute.asd is in none of the known classes
```

D.8 3 Object Clique Intersection with High Thresholds

```
Known object classes: car person

bike.asd is in class person(10:33.233257->63.948765)
```

```
box.on.wheels.asd is in none of the known classes
car1.asd is in none of the known classes
car2.asd is in class car(0:74.684258->74.684258)
car3.asd is in class car(2:56.374756->61.077553)
car4.asd is in class car(3:58.503597->62.438362)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class car(0:61.416080->61.416080)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:88.259048->88.259048)
person_a.decap.asd is in class person(5:47.856052->62.477707)
person_a.decap.legless.asd is in class person(6:43.455360->60.150772)
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(0:75.710846->75.710846)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:71.333931->71.333931)
person_c.asd is in class person(0:85.099655->85.099655)
person_d.asd is in class person(0:64.050941->64.050941)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
turtle.asd is in class car(3:46.972878->61.923248)
ute.asd is in none of the known classes
```

D.9 4 Object Blob Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is in class person(10:41.534039->50.195995)
box.on.wheels.asd is in none of the known classes
car1.asd is undecided, probably none (11)
car2.asd is in class car(0:59.747700->59.747700)
car3.asd is in class car(0:65.807487->65.807487)
car4.asd is in class car(0:69.560051->69.560051)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is one of car(0:51.072868->51.072868) person(0:51.072868->67.005394)
house.asd is in class person(6:35.066650->56.517426)
```

```
mug.asd is in class car(11:22.201128->58.642467)
np.asd is in none of the known classes
person_a.asd is in class person(0:79.612503->79.612503)
person_a.decap.asd is in class person(0:58.586025->58.586025)
person_a.decap.legless.asd is in class person(0:54.815697->54.815697)
person_a.hat.asd is undecided, probably none (11)
person_a.hat.legless.asd is undecided, probably none (11)
person_a.legless.asd is in class person(0:55.543419->55.543419)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:63.597965->63.597965)
person_c.asd is in class person(0:82.422409->82.422409)
person_d.asd is in class person(0:64.839462->64.839462)
skate.asd is in class car(9:28.293728->58.431049)
square.asd is in none of the known classes
tank.asd is in class person(8:25.925007->55.269718)
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in class person(0:56.253597->56.253597)
turtle.asd is in class car(6:32.708576->55.183712)
ute.asd is in none of the known classes
```

D.10 4 Object Blob Intersection with High Thresholds

```
Known object classes: car person

bike.asd is undecided, probably none (11)
box.on.wheels.asd is in none of the known classes
car1.asd is in none of the known classes
car2.asd is in class car(1:59.747700->60.633152)
car3.asd is in class car(0:65.807487->65.807487)
car4.asd is in class car(0:69.560051->69.560051)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class person(0:67.005394->67.005394)
house.asd is in class person(7:35.066650->65.767921)
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:79.612503->79.612503)
person_a.decap.asd is in class person(3:58.586025->61.025509)
person_a.decap.legless.asd is in class person(5:54.815697->61.635639)
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(6:55.543419->61.711365)
person_a.limbless.asd is in none of the known classes
```

```
person_b.asd is in class person(0:63.597965->63.597965)
person_c.asd is in class person(0:82.422409->82.422409)
person_d.asd is in class person(0:64.839462->64.839462)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in class person(8:56.253597->61.156693)
turtle.asd is in class car(7:32.708576->64.441147)
ute.asd is in none of the known classes
```

D.11 4 Object Clique Intersection with Default Thresholds

```
Known object classes: car person

bike.asd is undecided, probably none (11)
box.on.wheels.asd is in none of the known classes
car1.asd is in none of the known classes
car2.asd is in class car(0:69.658394->69.658394)
car3.asd is in class car(0:63.991646->63.991646)
car4.asd is in class car(0:66.394989->66.394989)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class car(0:67.453079->67.453079)
house.asd is in class person(9:20.933527->55.752480)
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:81.302505->81.302505)
person_a.decap.asd is in class person(6:41.342365->51.812717)
person_a.decap.legless.asd is in class person(8:36.947922->52.490696)
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in class car(11:25.046112->50.764919)
person_a.legless.asd is in class person(0:62.072403->62.072403)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:64.088417->64.088417)
person_c.asd is in class person(0:71.794182->71.794182)
person_d.asd is in class person(0:68.680191->68.680191)
skate.asd is in class car(9:21.574013->58.101574)
square.asd is in none of the known classes
tank.asd is in class person(8:21.856445->52.901413)
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is undecided, probably none (11)
```

```
turtle.asd is in class car(4:39.051334->55.495991)
ute.asd is in none of the known classes
```

D.12 4 Object Clique Intersection with High Thresholds

```
Known object classes: car person

bike.asd is in none of the known classes
box.on.wheels.asd is in none of the known classes
car1.asd is in none of the known classes
car2.asd is in class car(0:69.658394->69.658394)
car3.asd is in class car(0:63.991646->63.991646)
car4.asd is in class car(0:66.394989->66.394989)
exp.asd is in none of the known classes
glasses.asd is in none of the known classes
horse.asd is in class car(0:67.453079->67.453079)
house.asd is in none of the known classes
mug.asd is in none of the known classes
np.asd is in none of the known classes
person_a.asd is in class person(0:81.302505->81.302505)
person_a.decap.asd is in class person(8:41.342365->66.961273)
person_a.decap.legless.asd is in class person(9:36.947922->60.495415)
person_a.hat.asd is in none of the known classes
person_a.hat.legless.asd is in none of the known classes
person_a.legless.asd is in class person(0:62.072403->62.072403)
person_a.limbless.asd is in none of the known classes
person_b.asd is in class person(0:64.088417->64.088417)
person_c.asd is in class person(0:71.794182->71.794182)
person_d.asd is in class person(0:68.680191->68.680191)
skate.asd is in none of the known classes
square.asd is in none of the known classes
tank.asd is in none of the known classes
test1.asd is in none of the known classes
test2.asd is in none of the known classes
truck.asd is in none of the known classes
turtle.asd is in class car(5:39.051334->63.807568)
ute.asd is in none of the known classes
```