Formal Hierarchical Object Models for Fast Template Matching

M. C. COOPER*
Department of Computer Science, University of Hull, Hull HU6 7RX

A formal class of object models is defined, called repnets, which take advantage of the repetitive structure of objects, so that repeated sub-objects need not be stored more than once. Since sub-objects may contain repeated sub-sub-objects a hierarchy results. It is shown that the efficiency of template matching can be greatly increased if the template is decomposed into a compact repnet, and potential applications are identified in the inspection of printed circuit boards and in the application of edge-detection operators to pictures.

Received May 1987

1. INTRODUCTION

The concept of a hierarchical object model is very simple: an object is defined in terms of its component parts, which in turn are defined in terms of their components, until some basic level is reached at which further decomposition is not worthwhile. For example a human being may be defined as a head, a body, two arms and two legs in permissible relative positions. An arm may be defined in terms of a hand, an upper arm and a lower arm, with a hand being decomposed further down to the level of the individual bones of the fingers.

Although the description of a jointed object in terms of its rigid component parts is very important in practical computer vision algorithms for the recognition of, for example, a human being in different positions such as sitting, standing, running and jumping,[1] in this paper we are concerned only with the elimination of repetitive calculation when a component part occurs twice or more in the hierarchical model of a rigid object. For example, in a crude model of a human being with two identical arms, it is only necessary to perform one search for arms in the picture, instead of two separate searches for left and right arms. Because of this elimination of repetitive calculation, hierarchical object models may produce a significant and even combinatorial reduction in the time to search for an object in a picture compared with iconic definition of an object, in which a single image of an object is stored in memory.

This paper is a study of a formal class of hierarchical decompositions. In keeping with a research aim of finding provably correct algorithms for computer vision we have concentrated on a simple non-trivial problem, the recognition of two-dimensional rigid objects of a fixed orientation, before considering the full generality of three-dimensional flexible objects subject to arbitrary transformations. A solution to this simple problem using hierarchical object models could lead to practical applications in the automatic inspection of printed circuit boards.

Multi-resolution techniques are effective for the efficient storage and recognition of objects because of the local repetitiveness of object-images: pixels close together in an image have a higher than random probability of being the same or nearly the same. However, a multi-resolution object model cannot efficiently store the repetitive pattern of, for example, a chessboard or a page of text, since the repetitiveness is between non-adjacent parts of the object-image. We will show how an object can be defined by a network, which we call a repnet, capable of characterising non-local as well as local repetitive structure.

* To whom correspondence should be addressed at: Département D'informatique, Université de Bordeaux I, 351 cours de la Libération, 33405 Talance Cedex, France.

2. REPNET: A NETWORK OF REPEATS

A repnet is a data structure which eliminates repeats in an object definition by use of pointers in a hierarchical structure. As a purely illustrative example Fig. 1 shows a black object on a white background. Fig. 2 is a decomposition of it into progressively more basic elements, and Fig. 3 shows how these data may be compressed by the use of pointers labelled with two-dimensional offsets. Those sub-objects which are identical except for translation have been merged to a single node.

More formally, a repnet is a cycle-free network, in which every sink is a pixel and every source is an object. The repnet in Fig. 3 has only one sink because every pixel in the object in Fig. 1 has the same value, and only one source because it represents a single object. A repnet with more than one source represents a set of objects, repeated component-parts between different objects being stored only once. It is intriguing to imagine a repnet storing all the objects a human being can recognise.

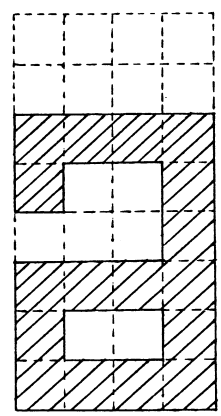

Figure 1. An object.

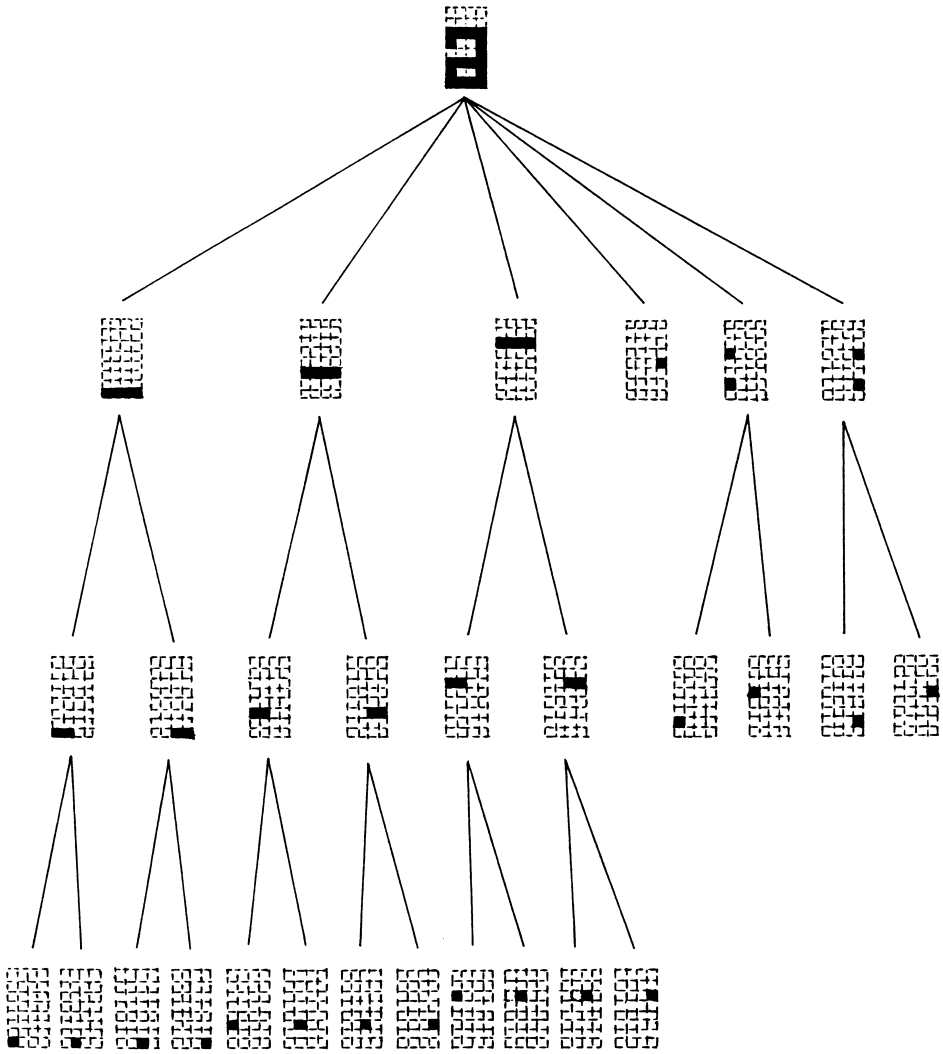

Figure 2. A decomposition of the object in Fig. 1 into sub-objects.

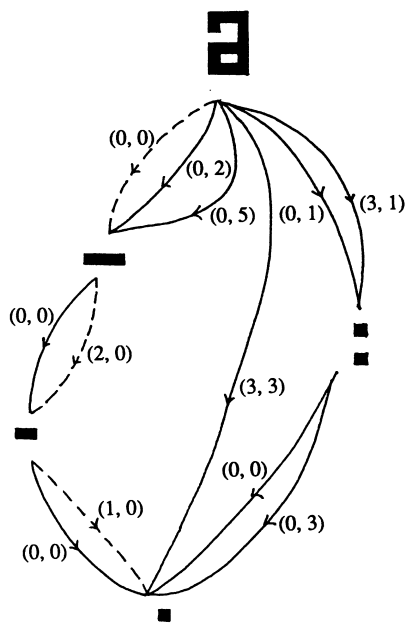

Figure 3. The repnet of the object in Fig. 1 corresponding to its decomposition in Fig. 2.

Associated with every arc from a node A to a node B in a repnet is an offset (x, y), which is the offset of the component part B within the component A. For example, the rightmost arc leaving the source of the repnet in Fig. 3 means that the object contains a copy of the sub-object pointed to by the arc and the bottom left-hand pixel of this copy is at position (3, 1). Every path between a source OBJECT and a sink p means that OBJECT contains a pixel of value p, and it occurs at the coordinates given by the sum of the offsets associated with the arcs on the path. For example, in the repnet in Fig. 3 the dotted path represents the bottom right-hand pixel of the object with coordinates

$$(0,0)+(2,0)+(1,0)=(3,0).$$

3. FAST MATCHING USING REPNET TEMPLATES

Template matching is a commonly employed technique in picture analysis, and its advantages and disadvantages are well documented.[2] It is closely associated with cross-correlation, which has been recognised as being one of the basic algorithms that must be implemented in an

image-processing system.[3] Gemmar[4] has reported that on a 1 MIPS sequential computer, applying an 800-pixel template at only 961 positions of a picture required 15 seconds. Due to computational limitations, matching against, say, a 1024×1024 picture is only practical for very small templates. Various sets of 3×3 templates have been proposed as edge-detection operators.[5–10] Repnets can significantly speed up the application of a template at every position of a picture.

The template matching problem can be expressed as follows.

Given an $MP \times NP$ picture array PICTURE and an $MO \times NO$ template array OBJECT determine those positions $[k, l]$ in the picture for which $S[k, l]$ is less than a threshold t, where

$$S[k,l] = \sum_{\substack{\text{pixels } [x,y] \text{ in OBJECT} \\ \text{such that } [k+x, l+y] \text{ is} \\ \text{in PICTURE}}} d(\text{OBJECT}[x, y], \text{PICTURE}[k+x, l+y])$$

and d is a distance function between pixel values of OBJECT and PICTURE.

By this definition, correlation, in which $d(a,b) = -a^*b$, is just one example of template matching.

The additive nature of the score S ensures that if a template OBJECT is the union of two disjoint templates OBJ1 and OBJ2, then

$$S(\text{OBJECT})[k,l] = S(\text{OBJ1})[k,l] + S(\text{OBJ2})[k,l], \quad (1)$$

where S(OBJ) is the array S obtained when applying the template OBJ to the picture. If OBJ1 is identical to OBJ2 except for a translation (x, y) then the cost of calculating S(OBJECT) is almost halved, because each S(OBJ2) $[k, l]$ does not need to be recalculated since

$$S(\text{OBJ2})[k,l] = S(\text{OBJ1})[k+x, l+y].$$

As a concrete example, consider the template OBJECT in Fig. 1 whose repnet is illustrated in Fig. 3. It is composed of six disjoint components: three copies (OBJ1, OBJ2, OBJ3) of a horizontal bar of length 4 pixels, a single pixel (OBJ4) and two copies (OBJ5, OBJ6) of a template composed of a pair of pixels separated by a vertical distance of 3 pixels. Therefore, for each position $[k, l]$,

$$\begin{aligned} S(\text{OBJECT})[k,l] = {} & S(\text{OBJ1})[k,l] + S(\text{OBJ2})[k,l] \\ & + S(\text{OBJ3})[k,l] \\ & + S(\text{OBJ4})[k+3, l+3] \\ & + S(\text{OBJ5})[k,l+1] \\ & + S(\text{OBJ6})[k,l+1]. \end{aligned}$$

In this example S(OBJ1), S(OBJ4) and S(OBJ5) are the only sums which need to be calculated, since S(OBJ2), S(OBJ3) and S(OBJ6) are copies of these arrays at the following offsets:

$$S(\text{OBJ2})[k,l] = S(\text{OBJ1})[k,l+2]$$
$$S(\text{OBJ3})[k,l] = S(\text{OBJ1})[k,l+5]$$
$$S(\text{OBJ6})[k,l] = S(\text{OBJ5})[k+3,l].$$

OBJ1 can be decomposed further

$$S(\text{OBJ1})[k,l] = S(\text{OBJ7})[k,l] + S(\text{OBJ8})[k,l]$$

into two copies (OBJ7, OBJ8) of a horizontal bar of length 2 pixels, only S(OBJ7) being required to be calculated, since

$$S(\text{OBJ8})[k,l] = S(\text{OBJ7})[k+2,l].$$

Both OBJ7 and OBJ5 are in turn composed of two copies (OBJ9, OBJ10 and OBJ11, OBJ12) of the single-pixel object, OBJ4:

$$S(\text{OBJ7})[k,l] = S(\text{OBJ9})[k,l] + S(\text{OBJ10})[k,l]$$
$$S(\text{OBJ5})[k,l] = S(\text{OBJ11})[k,l] + S(\text{OBJ12})[k,l]$$

where

$$S(\text{OBJ10})[k,l] = S(\text{OBJ9})[k+1,l] = S(\text{OBJ4})[k+1,l]$$
$$S(\text{OBJ12})[k,l] = S(\text{OBJ11})[k,l+3] = S(\text{OBJ4})[k,l+3].$$

The optimal calculation of S(OBJECT) for this template would be

```
FOR ALL [k,l] DO
  S(OBJ4)[k,l]: = d('black', PICTURE[k,l]);
FOR ALL [k,l] DO
  S(OBJ7)[k,l]: = S(OBJ4)[k,l] + S(OBJ4)[k+1,l];
FOR ALL [k,l] DO
  S(OBJ1)[k,l]: = S(OBJ7)[k,l] + S(OBJ7)[k+2,l];
FOR ALL [k,l] DO
  S(OBJ5)[k,l]: = S(OBJ4)[k,l] + S(OBJ4)[k,l+3];
FOR ALL [k,l] DO
  S(OBJECT)[k,l]: = S(OBJ1)[k,l] + S(OBJ1)[k,l+2]
                  + S(OBJ1)[k,l+5]
                  + S(OBJ4)[k+3,l+3]
                  + S(OBJ5)[k,l+1]
                  + S(OBJ5)[k+3,l+1].
```

A loop over all positions $[k, l]$ is performed for each node of the repnet in Fig. 3, beginning with the sink and ending with the source OBJECT. Appendix 1 contains a Pascal program Repnetcalc which performs this calculation for any template OBJECT defined in terms of a repnet. This divide-and-conquer computation of S(OBJECT) has asymptotic time complexity of the order of MP*NP* (number of arcs in the repnet of OBJECT), for a picture of size MP by NP. The optimal form for a repnet, representing the largest template that can be constructed with a given number of arcs, is illustrated in Fig. 4. Thus the minimal asymptotic time complexity for the repnet template application algorithm occurs when the template can be applied by successive doubling (successive calculation of (1)) and is of the order of MP*NP*log (number of pixels in OBJECT). Examples of templates which can be represented by this form of repnet are illustrated in Fig. 5.

Finding the most efficient repnet for a given template would involve a search over a combinatorial number of distinct repnets. We assume that if a template is to be applied many times it is worth the investment of effort to find a compact (though not necessarily optimal) repnet either by machine or by hand.

A variation on the problem of template matching stated above occurs when the template must simply be applied at every position of a picture, as an operator, for

Figure 4. The optimal repnet.

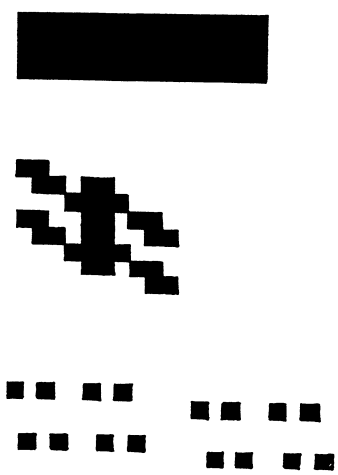

Figure 5. Examples of templates which have optimal repnets.

example in edge and line detection. As a purely illustrative example of the use of a repnet in this case, Fig. 6 shows a repnet for the set of nine 3×3 templates proposed by Frei and Chen[9] for edge and line detection. In this case the speed-up factor would be at most two, but it may be possible to apply larger versions of these templates, say 5×5, using the repnet algorithm with the same computational time required to apply the identical number of 3×3 templates using the direct method, repeats being more likely for larger templates.

4. REPNETS AND OTHER FAST TEMPLATE-MATCHING STRATEGIES

This section is a discussion of how repnets can be combined with other fast matching strategies, and in particular subtemplate matching.[11] It is shown that subtemplates with very compact repnets can be found in templates representing subsections of a digitised picture of a printed circuit board.

Subtemplate matching is a technique by which a subset of the template is applied to the picture, and only at those positions $[k, l]$ at which a rejection threshold $t1$ is not passed by the partially calculated sum $S[k, l]$ is the remainder of the template applied. If a subtemplate with a compact repnet is chosen the cost of the first stage can be greatly reduced. Unfortunately the repnet algorithm cannot then be used on the remainder of the template since most of the positions $[k, l]$ in the picture will have been rejected, and the repnet algorithm is only valid if $S(\mathrm{OBJ})[k, l]$ is calculated for all positions $[k, l]$, and for all the sub-objects OBJ corresponding to the nodes of the repnet for the complete template, OBJECT.

Pixel-by-pixel comparison is best suited to situations where the template represents a solid object without the possibility of distortion. This is often the case in automatic visual inspection[12] and in particular in the inspection of printed circuit boards (PCBs), where the technique is the only reliable method of detecting major

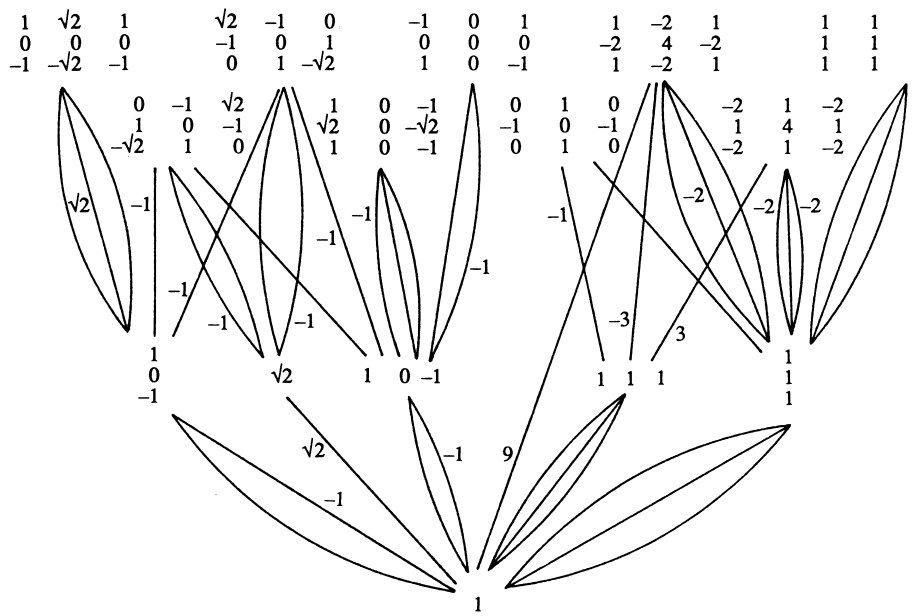

Figure 6. A decomposition of the template boundary-detection operators of Frei and Chen. This repnet takes advantage of the linearity of the distance function d, as well as the repetitiveness of the templates, to eliminate repeated calculation. The numbers shown by the arcs are multiplication factors. Offsets are not shown.

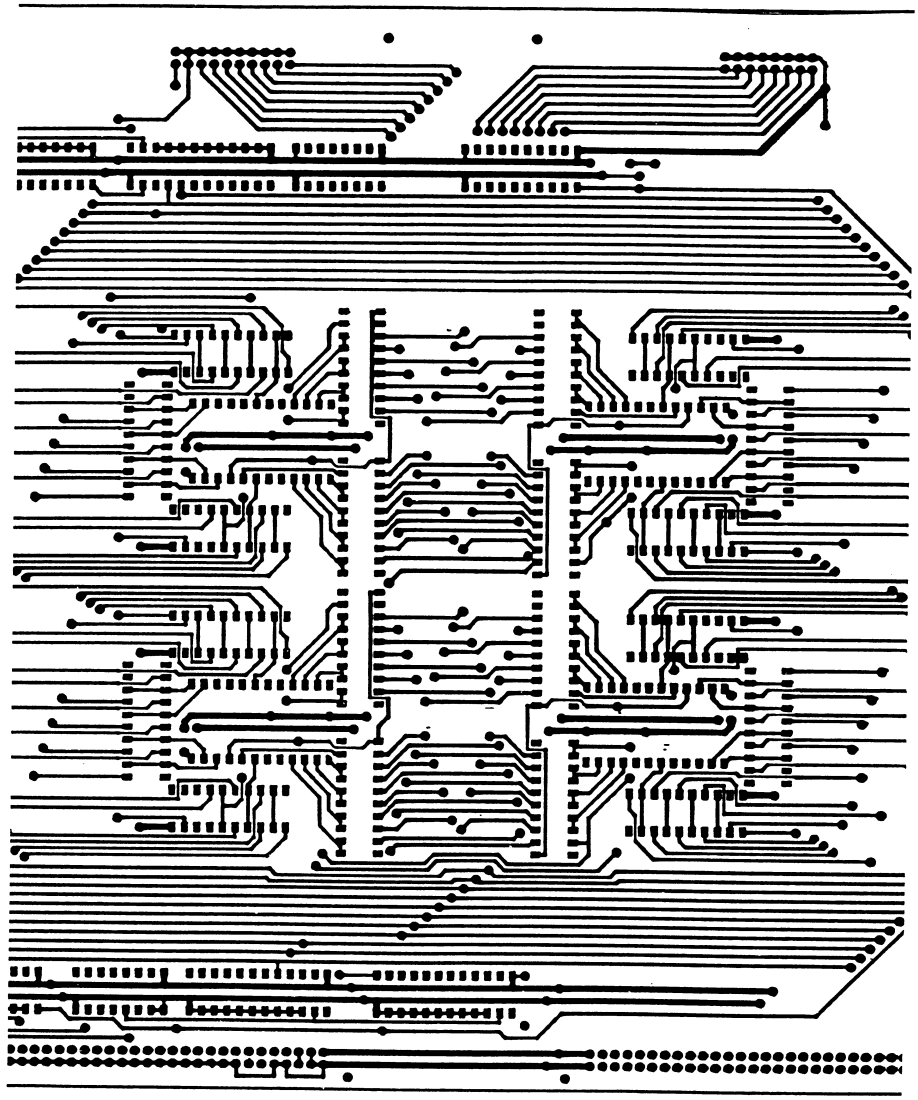

Figure 7. A PCB mask.

faults.[13] With the increasing density of printed circuits and the resulting lower production yields, automatic optical inspection technology is rapidly becoming a major pacing factor in PCB manufacture. In order to test the feasibility of the subtemplate/repnet algorithm a digitised picture of a PCB mask was examined. Fig. 7 shows the complete board, and Fig. 8 shows the detail within a 32×64 rectangle using a fairly coarse digitisation for the whole board.

Repnets were found 'by hand' for subtemplates of various sizes of the template of Fig. 8. Their properties are summarised in Table 1. The number of additions required to apply the subtemplate is proportional to the number of arcs in the repnet representing the subtemplate. The calculation of essential memory requirement to apply the subtemplate is more complicated and is explained below. All the subtemplates were required to have a reasonable balance of black and white pixels to prevent spurious matches against all-black or all-white areas of the picture. For this particular example a subtemplate covering half the pixels of the full template can be applied using only 40 basic operations, an improvement factor of 25 over the naïve template-matching algorithm.

There is often a possible trade-off between space and time, and this is illustrated in the fourth row of Table 1, where the time and space requirements are given for two different repnets of the same subtemplate. Although it is always possible to construct a repnet such that the template-matching calculation requires only 2*MP*NP memory cells, a different repnet may exist for the same template which can be applied faster but with greater memory requirements.

This trade-off only occurs for large templates. There are two distinct algorithms for template matching using a repnet. In the first algorithm the subtotal arrays S(OBJ) are calculated in full for the sub-object OBJ at a node before going on to the next parent node, and the computation of S(OBJ) requires only one pass through the repnet. This is the algorithm given in Appendix 1. The memory requirement is MP*NP*(number of working arrays in use), which depends on the 'breadth' of the repnet. As was stated above, this can always be as low as 2*MP*NP since every template has a decomposition as a union of optimal repnets (i.e. as a union of repnets such as the one in Fig. 4). This is essentially a sequential algorithm.

A second algorithm exists which, as will be demon-

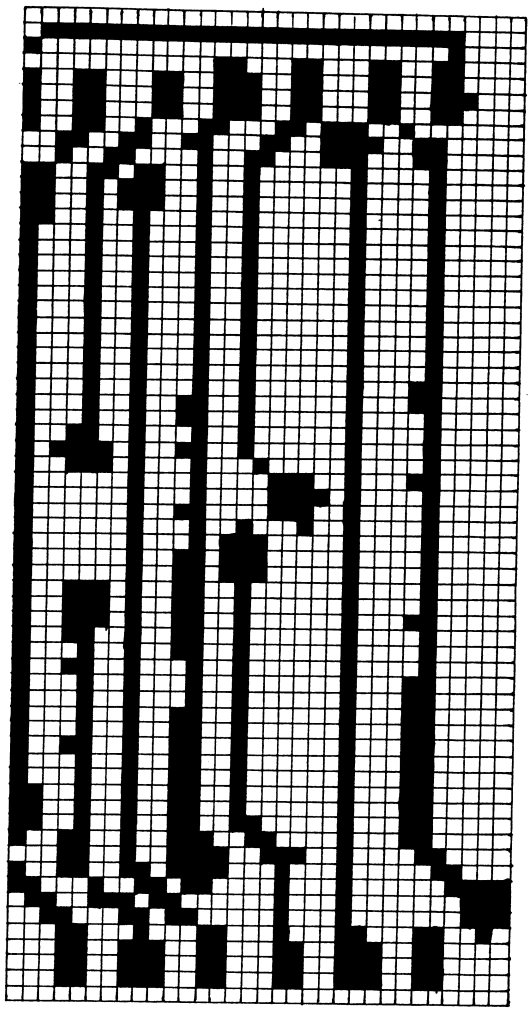

Figure 8. A 32 by 64 template from the PCB mask of Fig. 7.

Table 1. Computational requirements for the application of subtemplates, of various sizes, of the template in Fig. 8

Number of pixels in subtemplate (maximum = 2048)	Number of additions to apply subtemplate (all figures *MP*NP)	Memory requirement excluding PICTURE and OBJECT arrays
256	8	8*MO*NP (=131,072)
512	17	17*MO*NP (=278,528)
1024	40	40*MO*NP (=655,360)
1536	121	4*MP*NP (=1,048,576)
1536	156	3*MP*NP (=786,432)
2048	~400	3*MP*NP (=786,432)

strated in the next section, is more naturally parallelisable. After an initial set-up period, the computation is a succession of passes through the repnet, with as few as one addition being performed at each node. One such pass calculates $S(\text{OBJECT})[k,l]$ for one position $[k,l]$. In this case, for each component template OBJ, values of $S(\text{OBJ})[k,l]$ which will not be referred to again need not be stored. If the object is stored in an MO × NO array, the picture can be scanned in either an MO-deep horizontal scan or an NO-wide vertical scan, with a storage requirement of min{MO*NP, NO*MP}* (number of nodes in repnet). The figures in the table are based on MP = NP = 512, MO = 32, NO = 64.

As a final point it should be observed that the optimisation problem should not be equated with minimising the computational cost. The problems of non-detection and false detection must also be considered.[14] From the theory of matched filters[15] it is known that optimal discrimination is obtained when both PICTURE and OBJECT are 'whitened' before template matching, thus destroying any self-correlation in PICTURE and OBJECT. By selecting the subtemplate with the greatest repetitive structure we may also be choosing the most likely subtemplate to match spuriously with the picture. Thus it is advisable that, for each different repnet, the rejection threshold $t1$ be chosen empirically after trials on real pictures.

It is essential to the nature of the repnet algorithm that the score $S(\text{OBJ})[k,l]$ is calculated for all positions $[k,l]$ and all sub-objects OBJ corresponding to nodes in the repnet. Hence strategies such as Barnea and Silverman's[16] monotonically increasing threshold or Nagel and Rosenfeld's[17] ordering placed on the pixels of OBJECT have very limited application in combination with the repnet algorithm, since both these algorithms work by quickly eliminating the majority of possible positions $[k,l]$ for a template.

5. PARALLEL IMPLEMENTATION OF REPNET TEMPLATE MATCHING

The network of a repnet would seem to be an ideal candidate for parallelisation. However, unless a particular template was permanent it would be extravagant to construct an MIMD (multiple instruction, multiple data) machine with one processing element (PE) for each node of the repnet. However, with the possibility of switchable interconnections it is possible to foresee a multiple processor machine which could be reconfigured to a new repnet. The computation performed at each PE would simply be the calculation of the array $S(\text{OBJ})$, where the corresponding node of the repnet represented the sub-object OBJ, as the sum of the inputs $S(\text{OBJ1})$, $S(\text{OBJ2}), \ldots, S(\text{OBJ}n)$ from connected PE's representing the sub-sub-objects OBJ1, OBJ2, ..., OBJn which constitute OBJ, in accordance with the second algorithm discussed above. Each PE would be required to store an array no larger than MO*NP or NO*MP, except at the PE corresponding to the root of the repnet where the complete array S must be stored. The total processing time in a perfectly pipelined system would be

$$(\text{initial set-up time}) + (\text{number of positions } [k,l])$$

which is of the order of

$$\text{MO*NP} + (\text{MP} + \text{MO} - 1)*(\text{NP} + \text{NO} - 1),$$

which is of the order of MP*NP. With the necessary parallel hardware it is therefore possible to search for an object in a picture in time linear in the number of pixels in the picture.

Since special-purpose or reconfigurable MIMD mach-

ines may be extravagant, it is worth considering an alternative cheaper architecture. It has been demonstrated that a speed-up of N can be achieved for template application when the picture is divided equally between the N processing elements of an SIMD (single instruction, multiple data) machine.[18] This can easily be generalised to the case where each PE executes the repnet algorithm. Each PE could operate with $1/N$th of the memory requirements of the sequential version of the algorithm, which will be at most a small factor times the memory requirements of the direct template-matching algorithm, as is evident from the final column in Table 1.

6. COMPARISON OF THE FOURIER TRANSFORM AND REPNETS FOR TEMPLATE MATCHING

In order for an algorithm for template matching to be practical it should preferably have time complexity no greater than the order of MP*NP*log(MO*NO). Although several fast algorithms have been published,[19] the only one which can guarantee this performance for any OBJECT and PICTURE is the Fourier domain method, which is

(1) transform OBJECT and PICTURE to the Fourier domain using the fast Fourier transform (FFT);

(2) perform a pointwise multiplication of the transform values;

(3) transform the result back to the spatial domain using the FFT.

This is only a sketch of the Fourier domain method of correlation, and the interested reader should consult the textbooks of Rosenfeld and Kak[2] or Pratt[15] for more details.

The repnet algorithm cannot be combined with the Fourier transform method, which is why a direct comparison is being made between the two. (In fact, an attempt to do so only resulted in an algorithm with asymptotic complexity MP*NP*MO*NO, which is as bad as the naïve template-matching algorithm.)

The Fourier domain method has two disadvantages, as follows.

(A) It is less efficient than the direct method for templates less than a certain size. Pratt[15] has calculated this to be between 6×6 and 12×12, for square templates and pictures, depending on the proximity of MP (=NP) to the next power of 2.

(B) It can only be used to calculate the cross-correlation, that is, S, with the specific distance function $d_C(a, b) = -a*b$. This measure is based on the Euclidean metric $d_E(a, b) = (a-b)^2 = a^2 + b^2 - 2*d_C(a, b)$. This is perfectly adequate for binary or grey-level pictures which contain exact copies of the template except for the addition of Gaussian noise.

Given A, above, correlation by the repnet algorithm is certainly worth considering as an alternative to the Fourier transform method for templates of size less than 12×12. Most templates used as operators – as edge or line detectors for example – are smaller than this critical size.

A distance function $d(a, b)$ which could have a different value for each (a, b) and did not depend only on the difference $a-b$ would be a considerable improvement on the Euclidean metric. Let $p(a, b)$ be the probability that pixel value a in the template is garbled to pixel value b when the template occurs in the picture, and let $q(b)$ be the *a priori* probability that a background pixel in the picture (i.e. a pixel which does not correspond to a pixel in the template) has value b. Then $p(a, b)/q(b)$ is proportional to the likelihood that a pixel of value b in PICTURE is a garbled version of a pixel of value a in OBJECT. A proof in Appendix 2, based on Bayes' rule, shows that setting

$$d(a, b) = \log\{q(b)/p(a, b)\} \tag{2}$$

would make $S[k, l]$ equal to a constant minus the log likelihood that OBJECT occurs at position $[k, l]$ of PICTURE, under the following assumptions:

(i) each position $[k, l]$ is *a priori* equally likely;

(ii) the noise at distinct pixels is independent and identically distributed;

(iii) the background pixels of PICTURE are independent and identically distributed.

Under these conditions the position $[k, l]$ at which the minimum value of $S[k, l]$ was attained would be the most likely position for OBJECT in PICTURE. Since the 'background' of PICTURE probably consists of real objects, including copies of OBJECT, the repetitive structure of real pictures is a two-edged sword, providing an efficient repnet algorithm while invalidating (iii) above. Despite this failing (2) is however a less arbitrary distance function than the commonly used measures d_C, d_E and the city-block metric. The repnet algorithm can make use of any additive distance function. A comprehensive list can be found in Cormack.[20] Since d would simply be stored as an array of values $d(a, b)$, it can be used to model any kind of noise (subject to condition (ii), above) and any systematic transformation of pixel values between those of the template OBJECT and its occurrence in PICTURE, such as a uniform increase in lighting.

When the pixel values of OBJECT and PICTURE are actually tri-stimulus colour values the validity of the cross-correlation is even harder to maintain. This statement is based on the fact that the perceptual colour difference between two colours is given by the length of a geodesic in Riemannian space,[15] and it has been shown[21] that it is not possible to map the tri-stimulus values into another three-dimensional space which is Euclidean, such that the colour difference measure is preserved. It is possible however to calculate the perceptual difference between colours, based upon computation of colour geodesics.[22] These values could be stored in the array d and used in the repnet algorithm. Whether a function of the perceived difference between colours is the most appropriate distance function to use in template matching is, of course, another matter.

The essence of the repnet algorithm for template matching is that repeated subtemplates need only be compared with the picture once, and hence repeated computation is avoided by storing intermediate results. This is the trademark of algorithms derived from the 'divide and conquer' strategy. In this sense there is a resemblance between all divide-and-conquer algorithms, and a particularly strong resemblance between the fast Fourier transform and the repnet algorithm. The familiar diagram to illustrate the FFT is almost a repnet for the basis vectors of the transform.

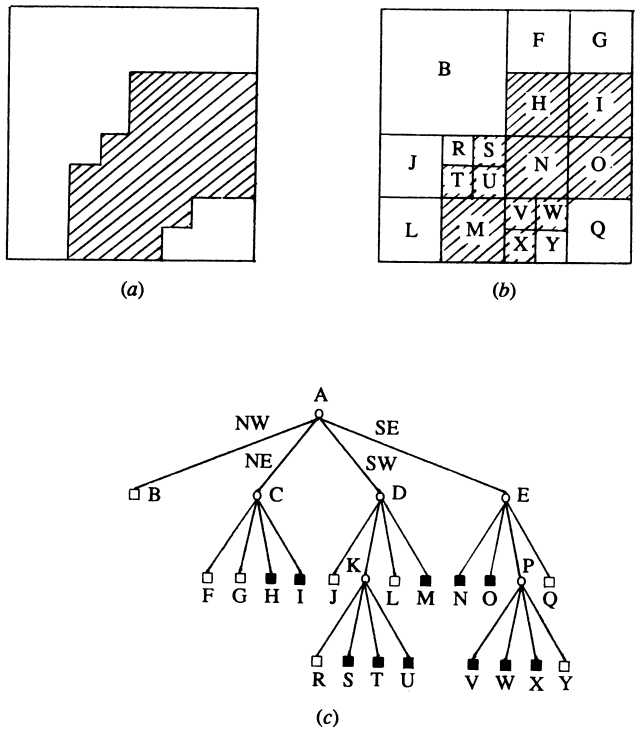

Figure 9. A picture (*a*), its successive division into quadrants (*b*) and the resulting quadtree (*c*).

7. DATA COMPRESSION: QUADTREES AS A SPECIAL CASE OF REPNETS

Decomposing an object-image in the form of a repnet not only provides an efficient algorithm, repnetcalc, for template matching, but also allows the object-image to be stored in a compressed form. This section shows that repnets and quadtrees are close relatives, repnets potentially providing greater data compression but being less easy to calculate.

Quadtree has become a generic term used to describe a class of hierarchical data structures for data compression.[23] The original quadtree was based on a successive division of two-dimensional space into four quadrants,[24] stopping when the quadrant had a constant value. Fig. 9(a) shows a picture, Fig. 9(b) its division into quadrants and Fig. 9(c) the resulting quadtree. Generalisations of this idea include the bintree, formed by successive division of space into two halves instead of four quadrants, the k-tree,[25] which is the extension of quadtrees to k dimensions, and trees in which the division of space is not predetermined but is different for each image so as to reduce the storage requirement further.[26,27] All these forms of quadtree can be expressed as repnets.

Quadtrees are an example of a wider class of data compression and image-analysis techniques which describe an image in terms of a vocabulary of given shapes. In the case of region quadtrees this vocabulary is a set of constant-value rectangles with horizontal and vertical sides each of length a power of 2 pixels. A rectangle with sides 2^r and 2^s can be represented by an optimal repnet, as illustrated in Fig. 4, with $r+s+1$ nodes, since it can be decomposed into two equal rectangles and the subrectangles similarly decomposed until the level of individual pixels is reached. Thus a quadtree can formally be extended to a repnet by replacing the leaf nodes of the quadtree by optimal repnets and labelling the original branches of the quadtree with offsets determined by the position of the corresponding son 'quadrant' in the father 'quadrant'. Thus quadtrees are transformable to, although not exactly equivalent to, a subclass of repnets.

Repnets potentially provide a much greater data compression than quadtrees, because of their far greater generality. Repeated sub-objects of any shape, and not even necessarily consisting of connected pixels, can give rise to data compression if the object is stored as a repnet. Dyer[28] has shown that a square object of size $2^m \times 2^m$ in a $2^n \times 2^n$ image will be encoded as a quadtree ranging in size from the order of $n-m$ nodes to the order of $2^{m+n}+n-m$ nodes as the position of the object varies in the image. The region quadtree is really only efficient for the representation of a small number of regions, and the worst example of inefficiency is the chessboard pattern, whose quadtree, shown in Fig. 10, is less space-efficient than the original image. By contrast, the repnet representation of the same pattern is shown in Fig. 11. Similar examples of regularly repeating patterns with very compact repnets, but inefficient region quadtrees, are patterns consisting of regular stripes, polka dots, checks, etc. Because all quadtrees can be converted to repnets there are no objects which have a compact quadtree but no efficient repnet. However, we are not

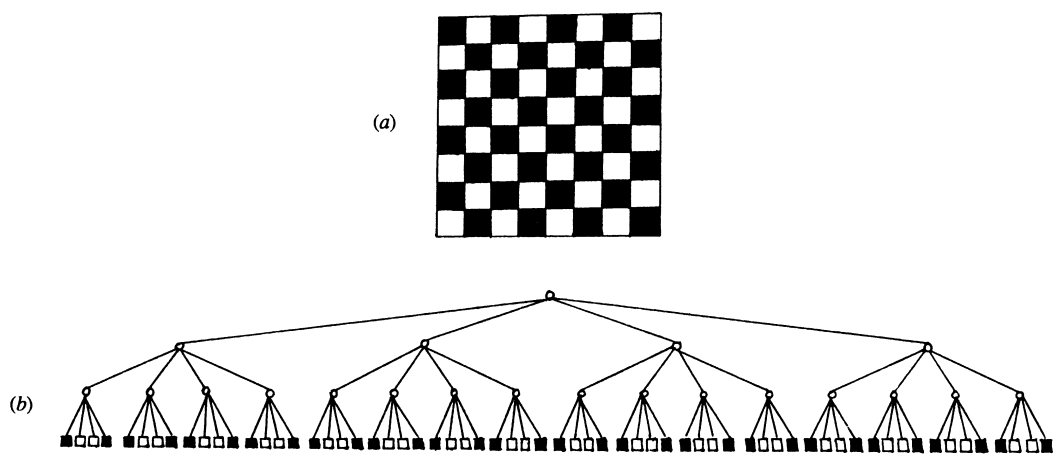

Figure 10. A chessboard patterm (*a*) and its quadtree (*b*).

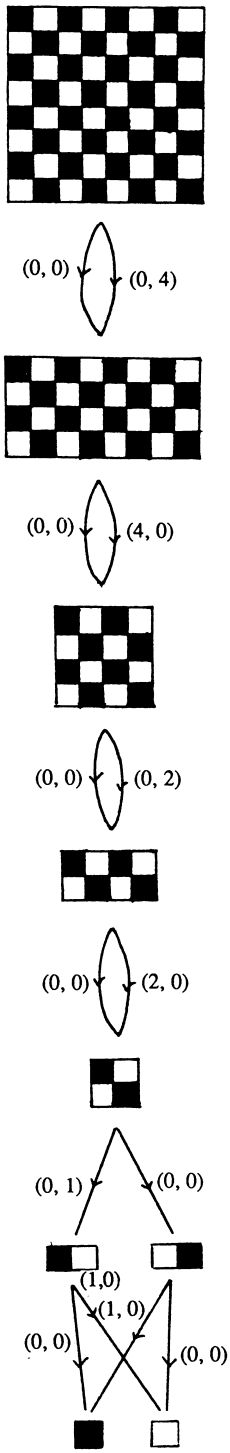

Figure 11. A repnet of the chessboard patter in Fig. 10(*a*).

aware of a non-combinatorial algorithm to find the most concise repnet of an object, and it seems unlikely that such an algorithm exists. The advantage of a quadtree is that it can be constructed in time proportional to the number of pixels in the image.

Further research is required to find a compromise between quadtrees and repnets which are efficient to construct but are not limited to such a restrictive alphabet of sub-objects as constant-value rectangles.

8. DISCUSSION AND FURTHER WORK

A hierarchical data structure has been introduced which is unusual in that non-local repetitiveness can be utilised to store an image efficiently. As introduced above, repnets can only describe exact repeats at the same orientation and dilation. In printed circuit boards the majority of repeats are of this form, and a potential application of repnets has been identified in the automatic visual inspection of PCBs.

However, the ideas behind the repnet can be transferred to any coordinate system and extended to any number of dimensions. That is, if a template is to be applied at different orientations as well as translations, then the repnet of the template can take advantage of repetitive patterns in which the repeated element is rotated as well as translated. The greater the number of dimensions, the larger the probability of repeats. This brief analysis has ignored the problem of the differences in the digitisation of a rotated template and the fact that applying a template at a set of orientations as well as translations would be impractical for all but the smallest templates.

When the range of pixel values is large repeats may be sparse. But if d satisfies the triangle inequality

$$d(a, c) \leqslant d(a, b) + d(b, c)$$

it may be possible to approximate OBJECT by a template with a compact repnet and thus efficiently determine a range of values within which $S[k, l]$ must lie.

Sets of pixels can be subtracted as well as added in a repnet. This would be applicable if an object were regularly shaped apart from small or regularly shaped holes.

Acknowledgements

This work was carried out under the supervision of Professor Julian Ullmann and was funded by U.K. Science and Engineering Research Council grant number GR/C74751. The PCB mask was kindly supplied by the Electronic and Electrical Engineering Department of the University of Sheffield.

REFERENCES

1. R. Nevatia and T. O. Binford, Description and recognition of curved objects. *Artificial Intelligence* **8**, 77–98 (1977).
2. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd edn, vol. 2. Academic Press, London (1982).
3. J. P. Strong, Basic image processing algorithms on the massively parallel processor. In *Multicomputers and Image Processing*, edited K. Preston Jr and L. Uhr. Academic Press, London, pp. 47–85 (1982).
4. P. Gemmar, Image correlation: processing requirements and implementation structures on a flexible image processing system. In *Multicomputers and Image Processing*, edited K. Preston Jr and L. Uhr. Academic Press, London, pp. 87–98 (1982).
5. J. M. Tenenbaum, A. C. Kay, T. Binford, G. Falk, J. Feldman, G. Grape, R. Paul, K. Pingle and I. Sobel, The Stanford hand–eye project. *Proc. IJCAI*, edited D. A. Walker and L. M. Norton, pp. 521–526 (1969).
6. J. M. S. Prewitt, Object enhancement and extraction. In *Picture Processing and Psychopictorics*, edited A. Rosenfeld and B. Lipkin. Academic Press, London, pp. 75–149 (1970).
7. R. Kirsch, Computer determination of the constituent

structure of biological images. *Computers and Biomedical Research* **4**, 315–328 (1971).
8. G. S. Robinson, Edge detection by compass gradient masks. *Computer Graphics and Image Processing* **6** (5), 492–501 (1977).
9. W. Frei and Chung-Ching Chen, Fast boundary detection: a generalised and a new algorithm. *IEEE Trans. Comput.* **C-26** (10), 988–998 (1977).
10. S. Levialdi, Edge extraction techniques. In *Fundamentals in Computer Vision*, edited O. D. Faugeras, pp. 117–144. Cambridge: University Press (1983).
11. G. J. Vanderbrug and A. Rosenfeld, Two-stage template matching. *IEEE Trans. Comput.* **C-26** (4), 384–393 (1977).
12. K. S. Fu, Pictorial pattern recognition for industrial inspection. In *Pictorial Data Analysis*, edited R. M. Haralick, pp. 335–349. Springer, Heidelberg (1983).
13. O. Silven, T. Piironen, M. Elsila and M. Pietikainen, Performance evaluation of algorithms for visual inspection of printed circuit boards. *Proceedings, 7th ICPR*, pp. 1355–1357 (1984).
14. H. K. Ramapriyan, A multilevel approach to sequential detection of pictorial features. *IEEE Trans. Comput.* **C-25** (1), 66–78 (1976).
15. W. K. Pratt, *Digital Image Processing*. Wiley, Chichester (1978).
16. D. I. Barnea and H. F. Silverman, A class of algorithms for fast digital image registration. *IEEE Trans. Comput.* **C-21**, 179–186 (1972).
17. R. N. Nagel and A. Rosenfeld, Ordered search techniques in template matching. *Proc. IEEE* **60**, 242–244 (1972).
18. L. S. Siegel, H. J. Siegel and A. E. Feather, Parallel processing approaches to image correlation. *IEEE Trans. Comput.* **C-31**, 208–218 (1982).
19. J. K. Aggarwal, L. S. Davies and W. N. Martin, Correspondence processes in dynamic scene analysis. *Proc. IEEE* **69**, 562–572 (1981).
20. R. M. Cormack, A review of classification. *Journal of the Royal Statistical Society* A **134**, 321–353 (1971).
21. G. W. Wyszecki and W. S. Stiles, *Color Science*, Wiley, New York, 513 (1967).
22. E. J. Muth and C. G. Persels, Computation of geodesics in color 3-space by dynamic programming. *Proc. 4th Hawaii Conf. on System Sciences*, 155–157 (1971).
23. H. Samet, The quadtree and related hierarchical data structures. *ACM Computing Surveys* **16** (2) (1984).
24. A. Klinger, Patterns and search statistics. In *Optimizing Methods in Statistics*, edited J. S. Rustagi, pp. 303–337. New York, Academic Press (1971).
25. C. Jackins and S. L. Tanimoto, Quadtrees, oct-trees and k-trees – a generalised approach to the recursive decomposition of Euclidean space. *IEEE Trans.* **PAMI-5**, 533–539 (1983).
26. Y. Cohen, M. S. Landy and M. Pavel, Hierarchical coding of binary images. *IEEE Trans.* **PAMI-7** (3), 284–298 (1985).
27. S. N. Srihari, Representation of 3-d images. *Computing Surveys* **13**, 399–424 (1981).
28. C. C. Dyer, The space efficiency of quadtrees. *Computer Graphics and Image Processing* **19**, 335–348 (1982).

APPENDIX 1. PASCAL PROCEDURE REPNETCALC FOR TEMPLATE-MATCHING

The following global declarations are required:

```
TYPE
  pixelvaluerange   ={application-dependent}
  sumptr            =↑sumarray;
  sumarray          =array[1..MP,1..NP]
                     of real;
  repnetptr         =↑repnet;
  listofsubobjects  =↑listitem;
  listitem          =RECORD
                     subobject : repnetptr;
                     xoffset,
                     yoffset : integer;
                     next : listofsubobjects
                     END;
  repnet            =RECORD
                     Sptr : sumptr;
                     subobjects : listofsubobjects;
                     pixel : pixelvaluerange;
                     timesneeded: integer
                     END;

VAR Picture=array[1..MP,1..NP] of pixelvaluerange;
FUNCTION d (templatepixel,picturepixel :
pixelvaluerange) : real
{distance function : application-dependent}
```

The following recursive procedure calculates S(OBJECT) for a template OBJECT whose repnet (or rather a pointer to it) is passed as a parameter, p. After completion of *Repnetcalc*(p), S(OBJECT)$[k,l]$ will be stored in $p\uparrow.Sptr\uparrow[k,l]$. The repnet must be constructed beforehand with the following information at each node:

```
Sptr        = NIL
subobjects  = a list of the subobjects (and
              their offsets) which consti-
              tute the object OBJ repre-
              sented by this node
pixel       = pixel value of OBJ if it is a
              single pixel
timesneeded = number of arcs in the repnet
              which point to this OBJ, i.e.
              number of times S(OBJ) will be
              referred to when Repnetcalc
              is executed for the complete
              template OBJECT

PROCEDURE Repnetcalc (VAR p : repnetptr);
VAR k,l    : integer;
  nextOBJ : listofsubobjects;
  OBJp    : repnetptr;

BEGIN IF p↑.Sptr=NIL THEN
  {Otherwise S has already been calculated
   for this object}
  BEGIN
   IF p↑.subobjects=NIL
   THEN {Calculate S for this single-pixel
     object}
    FOR k: =1 TO MP DO
     FOR 1: =1 TO NP DO
      p↑.Sptr↑[k,l]: = d(p↑.pixel,
       Picture[k,l])
    ELSE {Calculate S for this object as
      the sum of the arrays S(OBJ) for
      its subobjects OBJ}
     BEGIN
       new(p↑.Sptr);
       FOR k: =1 TO MP DO
        FOR 1: =1 TO NP DO
         p↑.Sptr[k,l] : = 0; {initialise}
       next OBJ : = p↑.subobjects;
       WHILE next OBJ <> NIL DO
       BEGIN OBJp : = nextOBJ↑.subobject;
          Repnetcalc (OBJp); {Calculate S
          (OBJ)}

          WITH nextOBJ↑ DO
          WITH OBJp↑ DO
          BEGIN
           FOR k: =1 TO MP DO
            FOR 1: = TO NP DO
             p↑.Sptr↑[k,l] : = p↑.Sptr↑
              [k,l]+Sptr↑[k+xoffset,
              l+yoffset];
           timesneeded : = timesneeded
            - 1;
           IF timesneeded=0
           THEN {S(OBJ) is not needed
              again} dispose(Sptr)
          END;

          {Get next subobject}
          nextOBJ: =nextOBJ↑.next
       END
     END
  END
END;
```

APPENDIX 2. BAYES RULE DERIVATION OF *d*

The following assumptions are made:

(i) each position $[k,l]$ for the template is *a priori* equally likely;

(ii) the noise at distinct pixels is independent and identically distributed;

(iii) the background pixels of PICTURE are independent and identically distributed.

Let X be the event that the object occurs at the position corresponding to $[k,l]$ in the picture array. Let A be the event that the picture has the pixel values given by PICTURE.

Bayes' rule says that

$$Pr(X \mid A) = \frac{Pr(A \mid X) * Pr(X)}{Pr(A)} \tag{3}$$

Under the assumptions (ii) and (iii) above,

$$Pr(A \mid X) = \prod_{\substack{\text{pixels}[i,j] \\ \text{in PICTURE}}} Pr(A(i,j) \mid X)$$

where $A(i,j)$ is the event that pixel $[i,j]$ in the picture array has the value PICTURE$[i,j]$.

Let $p(a,b)$ be the probability that a pixel of value a in the template is garbled to a pixel of value b in the picture, and let $q(b)$ be the *a priori* probability that a background pixel of the picture has value b. Then, for all pixels $[k+x, l+y]$ of the picture,

$$Pr(A(k+x,l+y) \mid X) = \begin{cases} p(\text{OBJECT}[x,y], \text{PICTURE}[k+x,l+y]) \\ \text{if } [x,y] \text{ is a pixel in the template} \\ q(\text{PICTURE}[k+x,l+y]) \\ \text{otherwise} \end{cases}$$

Therefore

$$Pr(A \mid X) =$$

$$\prod_{\substack{\text{pixels}[x,y] \text{ in} \\ \text{OBJECT such that} \\ [k+x,l+y] \text{ is in PICTURE}}} \frac{p(\text{OBJECT}[x,y], \text{PICTURE}[k+x,l+y])}{q(\text{PICTURE}[k+x,l+y])} * Q$$

where

$$Q = \prod_{\substack{\text{pixels}[k+x,l+y] \\ \text{in PICTURE}}} q(\text{PICTURE}[k+x,l+y])$$

$$= \prod_{\substack{\text{pixels}[i,j] \\ \text{in PICTURE}}} q(\text{PICTURE}[i,j]$$

Q, $Pr(X)$ and $Pr(A)$ are all independent of $[k,l]$, under assumption (*i*). From (3) we can see that, by defining

$$d(a,b) = \log\{q(b)/p(a,b)\} = -\log\{p(a,b)/q(b)\},$$

$S[k,l]$, as defined in the text, will be equal to a constant minus $\log\{Pr(X \mid A)\}$. Hence the position $[k,l]$ at which $S[k,l]$ attains its minimum value (including positions at which the template only partially overlaps the picture), will be the most likely position for the object.