Multiconstraint shape analysis

Tom Henderson and Ashok Samal

The paper presents a method for applying multiple semantic constraints based on discrete relaxation. A separate graph is maintained for each constraint relation and used in parallel to achieve consistent labelling. This permits both local and global analysis without recourse to complete graphs. Here, the term 'local' is used with respect to a particular constraint graph, and thus includes global spatial relations on the features, eg parallel edges on an object will be neighbours in the parallel constraint graph even though they are far apart in Euclidean space. Another major result is a technique for handling occlusion by incorporating the use of spatially local feature sets in the relaxation-type updating method.

Keywords: shape analysis, discrete relaxation, constraint relation

One of the problems in computer vision is to identify the set of objects present in a given image. This is, in essence, the scene labelling problem. The problem can be mapped into what has been variously called the consistent labelling problem[1], the satisfying assignment problem[2], the constraint satisfaction problem[3], Waltz filtering[4] etc. We will refer to it as the consistent labelling problem (CLP). It has been shown this problem is NP-complete. There have been several approaches to solving the problem, including backtracking, graph matching and relaxation.

The approach used here is based on discrete relaxation; however, there are some major philosophical differences. As explained later, we distinguish between different types of constraints during the process of relaxation. Many relaxation-based approaches make use of only the local constraints as opposed to global constraints. They tend to ignore the global constraints because occlusion sometimes prevents their use. We argue that global constraints can be useful and, under certain circumstances, they may be extremely advantageous. The model proposed in this paper uses both local and global constraints. Also, with the advent of parallel computers it is imperative to look at the problem again and see if the approaches are suitable for parallel processing or not. This is especially important because the problem at hand has an exponential growth rate. For an example of the use of multiple semantic constraints with stochastic relaxation, see Faugeras and Price[5].

This paper gives a formalism to describe the 2D scene analysis problem as a consistent labelling problem, and explores the suitability of the approach for parallel processing.

CONSISTENT LABELLING PROBLEM (CLP)

Although there are several variations, the CLP can be formulated as follows, given

- a set of items or units, $\mathbf{U} = \{u_1, u_2, \ldots, u_n\}$
- that each unit u_i has a domain D_i, which is the set of acceptable labels; often the units all have the same domain, in which case $D_1 = D_2 = \ldots = D_n = D$, $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$
- a labelling $\mathbf{L} = \{L_1, L_2, \ldots, L_k\}$, where $k \leqslant n$, $L_i = (u_i, l_i)$, $u_i \in \mathbf{U}$ and $l_i \in D_i$; L_is are called unit labels

Without loss of generality, we can assume that all u_is are distinct. If $k < n$, then the labelling is a partial labelling and, if $k = n$, it is called a complete labelling.

A unit can have any label which is in its domain. Usually, however, there are restrictions on the labels a set of units can have simultaneously in order to be consistent. These constraints are expressed by a constraint relation **R.** Potentially **R** can be an n-ary relation.

A pair of unit labels $L_i = (u_i, l_i)$, $L_j = (u_j, l_j)$ are consistent if and only if $(u_i, l_i, u_j, l_j) \in \mathbf{R}$. A given labelling $\mathbf{L} = (L_1, L_2, \ldots, L_k)$ is consistent if unit labels L_i and L_j are consistent for all $i, j \leqslant k$.

The goal of the labelling problem is to find a complete consistent labelling, given a set of units **U**, the domains of these units **D** and the constraint relation **R**.

Computer Science Department, University of Utah, Salt Lake City, UT 84112, USA

0262–8856/86/02084–13 $03.00

⟨CLP⟩ : : = (**U, D, R**)

Other formulations of the problem ask for all solutions, but the nature of the problem is not changed. If a complete consistent labelling cannot be obtained, then the largest (or 'best') partial labelling should be found.

Solutions to CLP

The consistent labelling problem can be solved in many ways. The most straightforward method is what is called the generate-and-test method. Here, we list all the possible configurations and then select those which are consistent. It should be obvious that this method is going to be extremely slow for problems where **U** and **D** are large. For example, if |**U**| = 10 and |**D**| = 10, then the number of possible configurations is 10^{10}. In many cases the labels assigned to the first few units make the whole labelling inconsistent, and this can be detected early, during the configuration process. This observation could save a lot of computation time.

A better method, which takes advantage of the above observation, is standard backtracking. Here we start with a single unit and assign a label to it from its domain. Then we select another unit from the rest of the units and assign a label to it from its domain such that the partial labelling built so far is consistent. If at any point we cannot find such a label, then we go back one step and give the next possible label to the unit which was last assigned a consistent label, and continue the process. If we manage to assign labels to all the units consistently then we have found a solution, and if we run out of labels then there is no solution.

Although standard backtracking is more efficient than the generate-and-test method, it is still not good enough for many practical applications. There have been several approaches to overcome this problem. Gaschnig[2] attempted to solve it within the backtracking framework and gave two new backtrack-type algorithms: Backmark, where all redundant pair-tests are eliminated; and Backjump, where it is possible to backtrack across multiple levels, instead of just one level. These algorithms do indeed show better performance than the standard backtrack algorithm, as shown by Gaschnig. Haralick and Elliot[6] gave another algorithm called 'forward checking' to improve backtracking; this performs better than the other two algorithms in some cases. However, the results are only known to apply to problems where the labelling of each unit is constrained by all other units.

Haralick *et al*[7] have described two 'look-ahead' operators, Φ and Ψ, to reduce the computation during the backtracking process. Haralick and Shapiro[1,8] generalized these operators to 'look ahead' arbitrarily.

Waltz[4] took another approach to solve the problem. The idea was to initially assign all possible labels to all the units and remove a label from the label set of a unit if it was found that the label was not compatible with any of the labels of the others. Removing a label from a unit, in turn, makes some labels of some other units inconsistent. This process continues until there is no label of any unit that can be removed or the label set of one unit becomes empty. In the former case, it is still necessary to search for an unambiguous solution, while in the latter situation, no solution exists. Convergence to a consistent set of labels depends on the nature of the problem and the constraint set **R**. Rosenfeld *et al*[9] have described a modified version of Waltz's filtering algorithm which is a parallel iterative procedure, and has been generalized to allow probabilities to be associated with the labels.

Mackworth[3] and others use yet another approach to reduce computation during the backtracking process. For binary constraints, the problem can be formulated using graphs, where the nodes correspond to the units and the arcs represent the constraints between the units. Each node also has an associated label set, which gives the possible labels for the unit. Montanari[10] explores the fundamental properties of such networks and their applications. Mackworth[3] gives three consistency tests, node consistency, arc consistency and path consistency, which prevents the thrashing behaviour of the backtrack algorithms. He also gives several algorithms to achieve the above three consistencies in networks. These consistency checks are done first, and then the backtracking process is applied. Recently, Mohr and Henderson have given an optimal algorithm for arc consistency and an improved algorithm for path consistency[11].

Complexity of CLP

In this section we analyse the complexity of the CLP. As Knuth[12] points out, it is difficult to analyse the backtracking algorithms without actually running programs. Even then, the results are not very convincing since it is not obvious either why or if they should hold in a totally different problem domain. Gaschnig[2] took the experimental approach to study the complexity of different algorithms.

Since it is known that CLP is NP-complete[10], it is not important to find the complexity of the various algorithms used to solve the CLP, since the worst-case complexity will be exponential anyway. What is more important is to determine whether a particular algorithm is consistently better than the others. Even this is not a straightforward problem. Gaschnig[2] and Haralick and Elliot[6] give experimental results for the efficiency of some of these algorithms.

Application of relaxation to scene analysis

Relaxation techniques have been successfully applied to several problems. Gaschnig investigated the application of relaxation to the *n*-queens problem, criptarithmetic problems, the Soma Cube and Instant Insanity. Henderson and Davis used it for syntactic shape analysis[13,14]. The application with which we are concerned here is scene analysis. The problem is to locate and identify all the objects in a given scene. There are several methods that can be used to locate the objects. If we locate the boundary of the object then it unam-

biguously determines the object. Also, a set of features, if located in the right configuration, can sometimes determine the location and orientation of the object. So, to locate the objects, we must locate the various features (or the boundary edges). Once these features are obtained from the image, they can be assigned a set of possible labels depending on the problem at hand. Then the relaxation procedure can be applied to produce a consistent set of labels and backtracking is used to find a solution.

Parallel relaxation

Since the worst-case execution time for the problem at hand is exponential, it might be worthwhile to explore whether the problem lends itself to a parallel execution model. Also, now that multiprocessors are available, we can actually test these models, not just theoretically analyse them. In this paper, we describe a model which exploits the parallelism in the relaxation process.

Local *versus* global constraints

One of the first steps in locating an object is to locate its features. We can recognize objects on the basis of global features, like number of holes, size of various segments, total area of the segments, perimeter etc. Alternatively we can also use local features to locate objects. Here we use local features like corners, holes etc. We look for certain structure with respect to these local features in the image, and if we can find such a structure then we can locate the object.

Both methods have their advantages and disadvantages. A system based on global feature matching is prone to mistakes, particularly when the object is occluded or even partially defective. On the other hand, if the object is completely isolated in the scene under consideration then the method is very straightforward and efficient. A system using local features is probably more sophisticated and is more robust for a general scene, but it also takes a long time.

A method based on local features can use only the local constraints, ie constraints between nearby features. This is useful in many circumstances, when the object is partially occluded and it is likely that some of the features would be visible and they can be used to identify the object. However, there are instances when it would be very useful to have global constraints also, and in certain instances it is necessary to use global constraints to identify an object. For example, the object in Figure 1a, as it appears in the scene in Figure 1b, can never be identified using local features alone since all the holes are occluded. However, by using global constraints like the parallelism and perpendicularity of sides, we can easily identify the object.

Scene analysis using local features

Recently, several successful 2D scene analysis systems have been proposed based on the use of local features[15–17]. Of course, many of the very first systems proposed were based on global features, eg Fourier coefficients and moment invariants. To give a basis of comparison, we briefly describe a system which uses local features to recognize objects in a scene; this is described in detail in Bolles and Cain[18]. (Most of the terms used in this section are from that reference.) The algorithm is then analysed for its complexity.

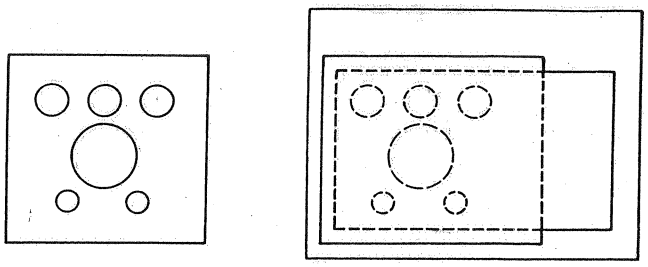

*Figure 1. **a** an object and **b** a scene with the object*

LFF method

The local feature focus (LFF) method has two major components: a training system and a runtime system. During the training phase, the object models are fed into the system and the system comes up with a set of strategies for recognition of each type of object. For each object it gives a list of features, which are termed focus features along with a set of neighbouring features. The idea is that if the focus feature is found in the image along with the neighbouring features, in the correct configuration with respect to the distance and orientation, then the object's location and orientation can be unambiguously determined.

The steps during run time can be summarized as follows

- Step 1: locate all potentially useful local features in the image
- Step 2: locate a focus feature; if there are none available in the image, then quit
- Step 3: get the set of nearby features for the focus feature
- Step 4: select those features which satisfy certain criteria, ie those features which have a corresponding occurrence in the actual image
- Step 5: make object-feature to image-feature assignments and transform these into a graph
- Step 6: find all the maximal cliques of the graph and hypothesize the object using them
- Step 7: verify the hypotheses; if a hypothesis succeeds, mark all the visible features of the object in the image as explained; go to step 2.

Analysis of the LFF method

Let:

- the total number of features in the image be N_i
- the number of line segments defining the boundary of the object be N_s

- the total number of features in the object be N_f
- the number of focus features for the object be N_F
- the average number of nearby features for each focus feature be N_n
- the average number of possible labels for a nearby feature in the image by N_a

Now we analyse the algorithm defined in the previous section, step by step. We are ignoring the complexity of step 1, since the time it takes is proportional to the size of the image and is (most often) independent of the scheme used.

- Step 1: O(image size), since features must be extracted from the entire image
- Step 2: O(constant), since the list of focus features can be built when the local features are being extracted from the image
- Step 3: O(constant), since the nearby features can be put in some kind of property list of the focus feature; an indexing scheme could also be used, with the same effect
- Step 4: $O(N_nN_l)$, since we essentially have to go through the whole list of the image features for each nearby feature, which takes, on average, $(N_nN_l)/2$ steps
- Step 5: $O(N_nN_a)^2$, since the number of vertices in the graph is (N_nN_a) and for building the graph we have to check for compatibility of each pair of vertices of the graph
- Step 6: This is a slightly complex step to analyse. The general problem of finding the maximal cliques of a graph is known to be NP-complete[19,20]. However, if the graph is planar then it can be done in linear time with respect to the number of vertices[21]. Using Kuratowski's theorem[22], a planar graph cannot have K_5 or $K_{3,3}$ as a subgraph. This reduces the problem to finding only K_4s, and K_3s in the graph, which can be done in linear time. So, the best case for this step is $O(N_nN_a)$ and the worst-case complexity is $O(k^{f(x)})$, where x is (N_nN_a) and k is a constant
- Step 7: $O(N_s + N_f)$, since we have to loop over the list of segments of the object and then we also have to loop over the list of the features for that object

So the total complexity for each iteration through steps 2 to 7 is the sum of all the above complexities, and the complexity of the whole algorithm is N_F times this factor, since we have to loop over the list of focus features.

Since step 6 in the worst case takes exponential time, the worst-case complexity of the algorithm is also exponential.

SCENE ANALYSIS USING BOTH LOCAL AND GLOBAL CONSTRAINTS

In this section we decribe a scheme to do scene analysis which uses both local and global constraints. It also lends itself well to a parallel excution model. We analyse the model and compare it with some other models, especially the LFF method described in the previous section.

The first step in scene analysis, before any attempt to locate objects in the scene, is to identify what are the possible objects which can be in the scene. We also need to extract a list of features from the image and their locations. They form the basis for all computations done in order to identify the objects which are actually in the scene.

Models for the objects

A model for an object is defined using the location and orientation of features, eg holes, corners, boundary edges, with respect to a fixed coordinate frame. These features are named or numbered and constitute the label space for the CLP. We also have constraints between these features. The constraints could in general be n-ary, where n is the number of features in the model. However, it is often sufficient to consider only binary constraints.

Let the set of models be $\mathbf{M}$ and the number of models be n. So, $\mathbf{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n\}$. Each of these models $\mathbf{M}_i$ is defined by, $\mathbf{M}_i = (\mathbf{F}^m_i, \mathbf{C}^m_i)$, where $\mathbf{F}^m_i$ is the set of features and $\mathbf{C}^m_i$ is the set of binary constraints for model $\mathbf{M}_i$.

Although all the constraints in $\mathbf{C}^m_i$ are binary, there are differences between them. For example, consider two boundary edges E_1 and E_2. Also, assume that E_1 is both parallel to and longer than E_2. The two constraints can be expressed as predicates parallel(E_1, E_2) and longer(E_1, E_2) respectively. Here, even though both are binary constraints, they are of different types. For one thing the parallel constraint is symmetric, while the longer constraint is not. Also, they have different semantics.

A constraint can equivalently be represented as a relation. In the above example, (E_1, E_2) belongs to both parallel and longer relations. So the constraint set for each model can be expressed as a set of relations, each with different meanings

$$\mathbf{C}^m_i = \{\mathbf{R}^m_{i,1}, \mathbf{R}^m_{i,2}, \ldots, \mathbf{R}^m_{i,k_i}\},$$

where $\mathbf{R}^m_{i,j} \subseteq \mathbf{F}^m_i \mathbf{X} \mathbf{F}^m_i$, $1 \leqslant j \leqslant k_i$

Let $\mathbf{C}^m = \cup^n_{i=1} \mathbf{C}^m_i$

The relations here are not just abstract relations. Here the relations also have a physical meaning associated with them. For example, if (a, b) $\in$ longer relation, then it also means a is physically longer than b.

Information from the image

After an image is acquired, the different feature instances are extracted from it. After that all the computation and analysis is based on these extracted features. Now the main problem is to associate these feature instances with the features of some models. The features thus obtained from the image are related to each other in some ways. For example, the image may have two holes and a corner, and we can have constraints like above(Corner,$Hole_1$), or bigger($Hole_1$,$Hole_2$). Obviously we could form many relations, each with

different physical meaning, to relate these features. However, we choose only those which are used to define the constraints between the features in the models. So the scene can be represented as

$$\mathbf{S} = (\mathbf{F}^s, \mathbf{C}^s)$$

where $\mathbf{F}_s$ is the set of features in the image and

$$\mathbf{C}^s = \{\mathbf{R}^s_1, \mathbf{R}^s_2, \ldots, \mathbf{R}^s_k\}, \mathbf{R}^s_i \subseteq \mathbf{F}^s \mathbf{X} \mathbf{F}^s$$

There is a direct correspondence between the two sets $\mathbf{C}^m$ and $\mathbf{C}^s$. For each relation $\mathbf{R}^s_i$ in $\mathbf{C}^s$ there is an equivalent relation $\mathbf{R}^m_j$ in $\mathbf{C}^m$ which has the same physical interpretation, although the domains of both relations are different.

Scene analysis as CLP

Now we formulate scene analysis (SA) as a consistent labelling problem (CLP) as defined in an earlier section. The set of units which needs to be labelled is the set of feature instances found in the image. For a simpler analysis, we assume that the domain of each of these units is the union of the features in all the models. In practice, however, it would be the set of features in all models which are of the same type as the unit (feature instance). For example, if the given feature instance is of type 'hole', then only those features in the models of type 'hole' can be in its domain. Obviously, the constraints in the models should be included in the constraint set for the CLP. We also have to include the constraints in the image since they also constrain the labelling process.

Features can have the same name in different models. To differentiate between them each feature in the model is prefixed by the model name. For example, feature 'Hole' in model 'Part' will be referred to as Part_hole. We assume that no two models have the same name and the set of features $\mathbf{F}^m_i$ for each model is augmented to remove any ambiguities. So the scene analysis problem in terms of a CLP is

$$SA = (\mathbf{U}, \mathbf{D}, \mathbf{R}),$$

where

$$\mathbf{U} = \mathbf{F}^s; \mathbf{D} = \cup^n_{i=1}\mathbf{F}^m_i; \mathbf{R} = (\cup^n_{i=1}\mathbf{C}^m_i) \cup \mathbf{C}^s_i$$

Network model for relaxation

This section briefly describes a graph/network model which is the underlying basis for the relaxation process explained above. This is similar to the network model used by Mackworth[3] and others in the sense that the nodes represent the units to be labelled and the constraints are represented by the arcs in the graph. However, there are several differences. What we have here is conceptually closer to a set of graphs than a single graph.

Instead of one graph, we have a set of graphs, one for each of the relations in the image. We also have a set of graphs, corresponding to the object models. This set of graphs is used to represent the constraints in the model. Let $\mathbf{G}^m$ be the composite graph to represent the set of model graphs, and let $\mathbf{G}^s$ denote the same for the image graph.

Model graphs

$$\mathbf{G}^m = \{\mathbf{G}^m_1, \mathbf{G}^m_2, \ldots, \mathbf{G}^m_n\}$$

where n is the number of models being considered. Each of the $\mathbf{G}^m_i$ also consists of a set of graphs corresponding to the various relations in the constraints

$$\mathbf{G}^m_i = \{\mathbf{G}^m_{i,1}, \mathbf{G}^m_{i,2}, \ldots, \mathbf{G}^m_{i,k_i}\}$$

where k_i is the number of relations in $\mathbf{C}^m_i$. Each $\mathbf{G}^m_{i,j}$ is a graph and can be represented by

$$\mathbf{G}^m_{i,j} = < \mathbf{F}^m_i, \mathbf{E}^m_{i,j} >$$

where edge (x,y) is in $\mathbf{E}^m_{i,j}$ iff (x,y) $\mathbf{R}^m_{i,j}$. The nodes of the graph are the features in the image which need to be labelled and the arcs represent a relation.

Image graphs

Unlike the model graphs, we have only one composite image graph, since we are concerned about only one image at a time. However, the composite image graph, $\mathbf{G}^s$, is again composed of a set of graphs, one for each of the relations in the image. So

$$\mathbf{G}^s = \{\mathbf{G}^s_1, \mathbf{G}^s_2, \ldots, \mathbf{G}^s_{k_s}\}$$

where k_s is the number of relations used in the image graphs. Each of these graphs corresponds to a relation in the image. So

$$\mathbf{G}^s_i = < \mathbf{F}^s, \mathbf{E}^s_i >$$

where edge (x, y) is in $\mathbf{E}^s_i$ iff $(x, y) \in \mathbf{R}^s_i$.

Example

The following example is simpler than the scenes one would normally find in actual cases, but is only used to explain the concepts. Figure 2 shows two simple

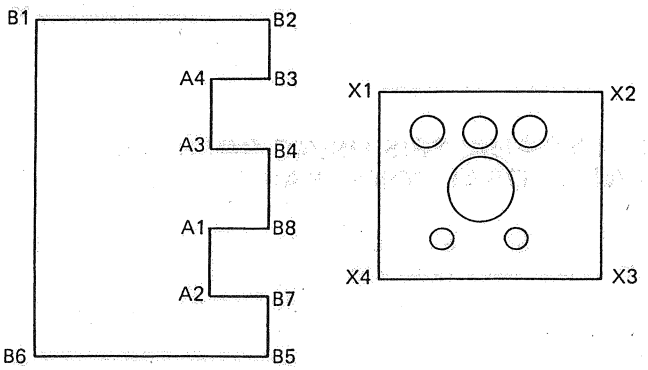

Figure 2. Part$_1$ (left) and Part$_2$

industrial parts, taken from Nitzan *et al*[23]; we refer to the parts as $Part_1$ and $Part_2$. The set of features for each part consists of only its boundary edges.

$\mathbf{M} = \{Part_1, Part_2\}$
$Part_1 = (\mathbf{F}_1^m, \mathbf{C}_1^m)$, $Part_2 = (\mathbf{F}_2^m, \mathbf{C}_2^m)$

$\mathbf{F}_1^m$ = {B1B2, B2B3, A4B3, A3A4, A3B4, . . . , A2B7, B5B7, B5B6, B1B6, B1B2}
$\mathbf{C}_1^m$ = {parallel, perpendicular, longer-than, equal-length}

$\mathbf{F}_2^m$ = {X1X2, X2X3, X3X4, X4X1}
$\mathbf{C}_2^m$ = {parallel, perpendicular, longer-than, equal-length}

We have four relations to express the constraints in the model for $Part_1$. These four relations are given in Table 1. The relations are not complete, but an idea is given of what these relations look like. $Part_2$ has similar relations.

Relaxation process

In this section we describe how the actual relaxation process is executed, and how it fits in a parallel processing framework. The first step is to build all the graphs, ie all the model graphs and the image graphs. We associate a set of labels with each node in the image graph, which represents the set of features it could be. It should be pointed out that, although we treated the graphs separately in the previous sections, they need not be really disjoint in actual implementation. Since the node set of all the image graphs is the same, they could be shared. The same applies to the model graphs, too. The topology of all these graphs remains unchanged during the relaxation process. What changes is the label set attached to each of the nodes in the image graph. The model graphs remain completely unchanged.

Once the graphs are constructed, the next step is to enforce the node, arc and path consistencies in the image graphs. This is where the system lends itself to parallel execution. We have graphs which are independent in the sense that they represent totally different types of constraints. We may have one graph to incorporate the size constraints, eg $Edge_1$ is longer than $Edge_2$. We may have another graph to represent the directional constraints, eg $Edge_1$ is parallel to $Edge_2$. The size and directional constraints are independent. However, they share the same set of nodes. To exploit this parallelism we create a process for each of the constraint types. So, in the above example, we will have four processes, trying to enforce consistencies in the parallel, perpendicular, longer-than and equal-length graphs. Hence the relaxation process is now distributed or split into subprocesses or levels which work on individual constraint types. The subprocesses structure is given in Appendix 1. The controlling process is described in Appendix 2.

Table 1. Constraint relations in $Part_1$

Parallel	Perpendicular	Longer-than	Equal-length
(A3B4 A1B8)	(B1B2 B2B3)	(B1B6 B1B2)	(B1B2 B5B6)
(A1B8 A2B7)	(B2B3 A4B3)	(B1B6 A3A4)	(A3A4 A1A2)
(A2B7 B5B6)	(A4B3 A3A4)	(B1B2 B2B3)	(A4B3 A3B4)
(B1B6 A3A4)	(A3A4 A3B4)	(A3A4 B2B3)	(A1B8 A2B7)

After the consistencies are enforced, we find all the solutions using standard backtracking or one of the modified schemes described above. It is hoped that the amount of bactracking necessary would be much less now that the networks are arc and path consistent.

To prove the correctness of the algorithms in the previous section, we state the following proposition. If a label **L** is removed from the label set of a node **N** in any graph, then the node **N** cannot have the label **L** in any complete consistent labelling.

The proof of this proposition is that, for a labelling to be consistent, it must be consistent in all the underlying graph types. Once a label in a node is removed by a process relaxG working on graph *i*, it is not consistent with respect to constraint type *i*, ie it cannot be part of a globally consistent labelling. So no solutions are lost by the above procedure.

The correctness of the algorithm is obvious. After consistency checks are enforced, we use backtracking to find all the solutions and reject those labels which do not lead to complete consistent solutions.

Efficiency considerations

In the previous section we described an algorithm to solve the scene analysis problem, and proved that it is both correct and complete. However, the performance of the given algorithm is very poor. In this section we analyse its efficiency and give some improvements to make it more efficient.

Complexity

The inefficiency of the algorithm can be seen very clearly at the step where we assign the initial label set to each of the nodes in the image graph. We assign $\mathbf{L}_{all}$ to label sets of all the nodes. $\mathbf{L}_{all}$ is the union of all the features of all the models which can be in the scene.

$$\mathbf{L}_{all} = \cup_{i=1}^{n} \cup_{j=1}^{m_i} \mathbf{F}_{i,j}^m$$

Clearly, this is a very large set. Even though many of the labels would be removed during the first few iterations of the relaxation process, this is definitely a major source of inefficiency.

Improvements

One simple and obvious improvement to the algorithm can be made by deleting those labels which are not of the same type from the label set of a feature. For example, if the feature in the image is a hole, it makes little sense to assign a label which is a corner or a boundary edge. Although this is a very simple modification, it should considerably reduce the initial size of the label set of the nodes.

However, it does not solve the problem entirely. One of the reasons the above method is so inefficient is that we are trying to do too many things at the same time. Although we are enforcing the constraints in different graphs in parallel, which is obviously useful, we are still, in essence, trying to label all the features of the image at the same time. Since this, in general, is going to be a very tough problem to handle — a scene may have a hundred features — it might be worthwhile to break the problem into smaller problems and try to solve them (the 'divide-and-conquer' paradigm). We should perhaps note again that we are dealing with an NP-complete problem. Since in the worst case it takes exponential time anyway, all we are trying to do is to gain speed wherever possible.

One way to break the problem into smaller (simpler) ones is to divide the scene into a set of smaller subscenes and try to analyse these. Although it might sound very simple, there are problems with this scheme. The most obvious one is how to perform the subdivision.

One simple way is to divide the scene into an m by n rectangular grid. Here, the problem is how to choose optimal values for m and n. There might also be duplication of effort, since an object may be spread over more than one rectangle.

The problem with the rectangular grid scheme is that we have no basis for performing the subdivision. We are essentially using heuristics to choose the values of m and n. Instead, we should be using the image and the models to guide our subdivision process. In the next section we propose a method which is based on the understanding of the models and the image.

New model

As mentioned in the previous section, the basic idea is to let the models and the image direct the process of dividing the scene into subscenes. There are two key observations which should be noted:

- a feature instance in the image can belong to only a few objects; for example, if the feature under consideration is a hole, then it can only belong to objects which have holes
- if a feature instance in the image is hypothesized to belong to a particular object, then the other features of the object should be located within a certain distance of it

What the first observation means, is that a feature can only belong to certain objects, and it should only have the labels corresponding to the features of those objects. The second observation implies that it is useless to consider the portion of the scene which could not include any features of the object under consideration. The above two observations form the basis of the subdivision of the scene into subscenes.

The algorithm which uses these ideas is given in Appendices 3 and 4. The process relaxM is the top-level process which spawns processes to work on subscenes. The process relaxSS works on the subscene, and for each possible object it tries to perform relaxation and find if the hypothesis is true.

Table 2. A set of objects

Object set	Features
$\mathbf{M} = \{\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \mathbf{M}_4, \mathbf{M}_5\}$	$\mathbf{F}_1$ = {round-hole, square-hole}
	$\mathbf{F}_2$ = {square-hole, triangular-hole}
	$\mathbf{F}_3$ = {round-hole, square-hole, triangular-hole}
	$\mathbf{F}_4$ = {round-hole}
	$\mathbf{F}_5$ = {square-hole}

Process relaxM controls the top-level division of the scene into subscenes. Function 'find-objects' returns all the objects which can have a particular feature. A very simple indexing scheme can be used to do this (see next section). Once a feature is chosen, and the objects it can belong to are associated with it, process relaxSS works to see if any of the objects actually are in the scene. This could be done in parallel as shown in the processes relaxS and relaxG. The process finds all the features in the bounding circle of the feature. The radius of the bounding circle is determined beforehand and is available (see below). The process then marks off these features, since they could potentially be explained. Then actual relaxation is done on this subscene using the object's model. If it turns out that the object is in the subscene, then all the features which have been labelled by the relaxation process are marked as explained, and all the other features are unmarked. This process also restarts the main process if necessary.

Simple indexing scheme

Function 'find-objects' in the process relaxM takes a feature and returns a set of objects of which it can be a part. This can be done by a brute force method: by searching each object for its feature list and checking whether it has the given feature as a member. It is a simple but very inefficient method. A simple indexing scheme is very effective at making it faster. The objects are indexed using the features as the keys, and finding the objects for a given feature is a one-step operation. The computation needed for making the index tables is done before run time and is a one-time step. Also, new objects can be added to the list incrementally, without totally reconstructing the index tables.

Table 2 shows a hypothetical set of objects and their features. From it we construct the index tables which are given in Table 3.

This indexing can be extended further on the basis of certain measures of the features. For example, if $\mathbf{M}_1$ has round holes of only a certain size, which is different from those of $\mathbf{M}_3$ and $\mathbf{M}_4$, then the round

Table 3. Index tables for the objects in Table 2

Feature	Object with feature
Triangular-hole	$\{\mathbf{M}_2, \mathbf{M}_3\}$
Square-hole	$\{\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_4, \mathbf{M}_5\}$
Round-hole	$\{\mathbf{M}_1, \mathbf{M}_3, \mathbf{M}_4\}$

holes can be further indexed with the size as the key. However, not all features may have such a measure.

Bounding circle

Here we describe a simple method to determine the bounding circle for a feature of an object. This helps in breaking up a scene into subscenes on which the relaxation is done. Like the index tables, this is done before run time and is a one-time computation. We associate with each feature of the object a position in two dimensions. So each pair of features f_i and f_j has a distance measure $d_{i,j}$ associated with it. We choose the Euclidean distance as the measure of distance. The radius of the bounding circle, r_i for the feature f_i, is the maximum of the distances associated with it.

$$r_i = \text{Max}\ (d_{i,j})\ \forall\, j : f_j \in \mathbf{F}$$

Appendix 5 describes the computation. The function 'distance' computes the Euclidean distance between the positions of the two features.

Comments

The approach we have proposed in this section works better in certain situations than others. Our subdivision is based on the scene itself and for the bounding circle we use the worst-case estimate. If the objects are very close together, we would use features of both objects for the relaxation process and this would slow down the process. On the other hand, if the two objects are close together, then the chances are that many features of one (or some of both) would not be visible. So the number of features we have to deal with may not be as large as it may seem at first sight.

If, however, the objects are spread out without much overlap, the relaxation process would reach a solution very quickly. Also, even if the objects are overlapping in physical space but fairly disjoint in the feature space, then, again, the convergence to the solution is speedy since those features which cannot belong to the object under consideration are ignored, thereby effectively reducing the number of initial labels the node in the graph can have. For example, if one object has only square holes and another has only round holes, then even if the square and the round holes may lie close to each other in the image, we would not include the round holes in the list of features in the bounding circle for a square hole.

EXTENDING RELAXATION FOR OCCLUDED SCENES

So far we have not given the details of how the relaxation process actually works for occluded scenes. In this section it is described in detail, since it has been transformed into a very different form. Although the relaxation process works fine for nonoccluded scenes, it does not work well if the scene is occluded. The basic reason is that, if a constraint is missing in the image, it does not mean that the constraint actually does not hold. It is possible that a constraint is not satisfied because some or all of the associated units are occluded. So the constraints do not have the same discriminating power in occluded scenes. In nonoccluded scenes, we use the boundary edges directly to describe the constraints, but this cannot be done in occluded scenes.

Also, some of the constraints have to be used or interpreted differently. For example, if a boundary edge X is longer than another edge Y in the model, it does not follow that the corresponding edge for X in the image will be longer than the corresponding edge for Y. In fact, if a boundary edge W is longer than another boundary edge Z in an image, no definite conclusion can be made about them from this information alone.

However, some of the constraints can be used under certain circumstances. For example, if there are only a few boundary edges which are longer than some absolute amount, and we find an edge in the image which is longer than this threshold, then it has to be one of the above edges. But this does not constrain the problem enough to reduce the solution set of the other labels.

One way to get around this problem is to use only local features like holes, corners etc and the constraints between them. Instead of using constraints between the sides (or boundary edges), we use constraints between the lines between the locations of features. We refer to these vectors as interfeature vectors (IVs). The advantage of using IVs instead of sides is that, unlike the sides, they are either present or absent, ie they cannot be partially missing or broken up into different parts because of occlusion. If both the features constituting an IV are present in the image, then the IV is defined; otherwise it is not. We can use the same constraints as before, but they are now between the IVs instead of the sides. Before they were binary constraints, but now they are essentially quarternary constraints.

However, the constraints still do not have the discriminating power to drive the relaxation process, since the constraint set in the image is incomplete. Initially each unit has all the labels of the model in the label set. We delete a label at a node if a certain constraint is not satisfied. But now we are not sure if the constraint is unable to be satisfied or is just not satisfied because of occlusion. So to drive the relaxation process we need to seed the label set of some units and then let relaxation take over. The goal is to get some positive information from the scene and then propagate it. The next two subsections describe how the IVs are created and how the initial seeding is done.

Creation of interfeature vectors

An interfeature vector is defined to be the vector between two feature locations. The magnitude of the vector is the distance between the two feature locations. If there are N features under consideration, then there could be as many as $N(N-1)$ IVs. This could be too large to be manageable, particularly in an image where there could be a large number of features. To avoid this problem, we connect only the

features which are within a certain distance of each other. This distance is not arbitrarily chosen; rather it is computed for each model separately so as to keep the number of IVs small. For example, for the object described in Figure 3, there are 28 IVs for a distance threshold of 2.1

Initial seeding of label sets

The goal of the seeding process is to reduce the label sets of some units, drastically if possible, so that it will then drive the relaxation engine. This means we need to find the peculiar properties of some features, their locations or their relationships. Since we are using the IVs as the basis for the constraints, we decided to look for peculiarities among these IVs. Two such quantities are used here and we found them very useful for the seeding process; they are

- magnitude of the interfeature vector (or the distance between the corresponding two feature locations)
- angle between two interfeature vectors; the two IVs are chosen such that they have exactly one common end point, ie they share one feature

We group these quantities, and try to find quantities which are very unusual. In our case we say a quantity is unusual, if it does not occur more than a certain number of times in the above groups. We determine this by histogram; Appendix 6 gives a brief outline of how the process works. (It should be noted that threshold distance, interfeature vectors, histograms etc are created for the models beforehand. This computation is performed only once.)

The function 'details' collects some more information about the interfeature vectors which correspond to the ith histogram entry (the associated features, etc). Function 'GetSize' computes the size of the histogram. It is computed such that two consecutive entries in the histogram are at least 'error' apart. All the special distances are collected in a list and become a part of the model. Special angles are also computed by a similar procedure and are stored in the model. A special distance is actually not a single quantity, but it has a range associated with it. The idea is that if the distance between two units in the image is in this range, then the two units should correspond to the two features in the model.

In the case of the objects given in the section on nonoccluded object recognition, we found that there are a few special lengths and angles for each of the models that are fairly unusual in the sense that there is practically no other quantity which is close to these special quantities in terms of magnitude. This information can be used to seed the label set of units in the image.

Labelling process

The process of labelling starts with type checking. Initially each unit (primitive) in the image is given all the labels of the desired model. Then, if the types of the unit and any of its labels are not consistent, those labels are removed from the label set (as above). The next step is the seeding process. Before the actual seeding can be done, the interfeature vectors and the angles are constructed for the image. The distance used is the threshold distance used by the model. After the lengths and angles are constructed, they are searched for the special angles and lengths. If any of the distances (or angles) match a special distance (or angle) for the model, then the label sets of the associated units are updated.

The control is then passed on to the relaxation process, and finally to the backtracking operator. Due to the changed nature of the problem, the structure of the relaxation process has been changed considerably too. In the next section, we give the motivation and the structure of the new version of the relaxation process.

Split-level relaxation process

The main reason to change the structure of the relaxation process is that there is no way to use positive information in relaxation. For example, if during the seeding process we infer that the primitive X can be either A or B, then we cannot use this information in any way more than if the label set of X had A and B without seeding. In relaxation, all nodes are treated equally and the information from seeding, which is very powerful and positive, cannot be used. As a matter of fact, it is possible for node X to lose all its labels during relaxation just because it lacks support at some other primitive which may not even belong to the model.

We divide the nodes (primitives) into two groups: nodes whose labels are fixed during the seeding process are called the strong nodes and others are called weak nodes. The strong nodes signify positive information. The strong nodes always remain strong, while the weak nodes may be elevated to strong status. During relaxation, a strong node can affect the label set of another (either strong or weak) node. This prevents the nodes which are not a part of the model from affecting the label sets of other nodes. Those units which survive the first iteration, ie whose label set is not empty at the end of the iteration, are then changed to strong nodes. Those nodes which do not survive the first iteration are not considered further and are not considered to be part of the model. After that, the relaxation process works as usual.

Another way to solve the problem of weak nodes deleting the labels of strong nodes is to allow a label at a node to remain if there is at least one support for it from any other node. (In standard discrete relaxation, a label remains if it has support from all neighbouring nodes.) This is motivated by the fact that some of the constraints may be unsatisfied due to occlusion. It should be noted, however, that there has to be support for the label in each of the possible graph types. This is fairly effective, but takes more time, since the labels are deleted after all the constraints are processed. The effect of change can only be perceived during the next iteration. In split-level relaxation, we delete

a label once it is determined that there is no support, and the change can be used by the next constraint during the same iteration.

Discussion

In this section we point out some of the advantages and disadvantages of this approach. Clearly the success of the method depends on the special lengths and angles. If they are occluded, then the method may take a longer time than usual. During the process, it might lose all the labels in which case it would not recognize the object. Or it may reduce the label sets of some nodes and leave the rest of the work to the backtracking procedure, which can be expensive.

However, if there are enough special distances and angles then, even if some are missing, the method will work very well. Our experience is that it will work well if we can reduce the label sets of two or three nodes to about two or three. Also, it is not unusual to find about four or five special distances and angles. If these are not enough, we could increase the threshold distance (for connection) to include more interfeature vectors, thereby increasing the possibility of finding more special distances and angles. Of course, this increases the cost of computation, both for the model and the image. The positive side of this is the fact that these computations can be done ahead of time, more than once if necessary to arrive at an optimal set of special distances and angles. Once this is done, it is a part of the model of the object and need not be recomputed.

Also, this takes care of a certain amount of error in both the model and the image. As pointed out before, the special distances have a range associated with them. So, if there is any small error the initial seeding is not affected. This makes the system more robust.

IMPLEMENTATION AND RESULTS

Most of the above ideas were implemented in PSL (Portable Standard LISP)[24]. The compiled code was run on a VAX-8600. The actual run times can be vastly improved if implemented on a LISP machine like the Symbolics 3600.

The system works in two phases. In the first phase, all the models are input and the threshold distance, special lengths and angles are determined and stored in the model structure. This is only a one-time cost and is similar to the training phase of local feature focus[18]. The input is at an intermediate level of description. What is input to the program is a list of local features, their locations and the constraints between them. The different types of constraints used are parallel, perpendicular, adjacent, longer-than etc.

Table 4. Results for nonoccluded scenes

Object number	Run time (ms)	Constraint types	Total number of constraints
1	357	5	43
2	476	6	33
3	85	4	15

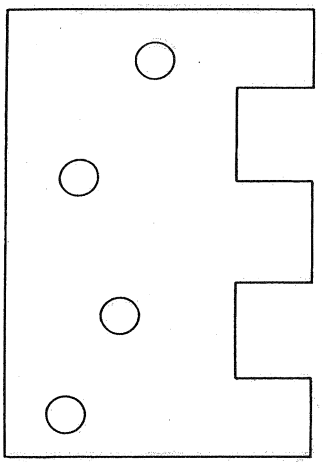

Figure 3. Object number 1

In the second phase, an image description is given along with the set of constraints associated with the local features in it. We also give a model to be searched for in the image. We only check for node and arc consistency. The AC-3 algorithm as defined by Mackworth[3] is used for enforcing consistency. Although we look for a single object in an image, it would be a straightforward extension to look for multiple objects in the same image. We essentially follow the model given above. The image description is similar to that for the models.

At the end of relaxation process, control is passed on to a backtracking procedure which does further consistency checks and lists out all the solutions. The output is a listing of local features in the image and the corresponding features in the object. If the label set of a primitive is empty, it is labelled as unknown.

Nonoccluded object recognition

Table 4 gives the time taken to recognize three parts shown in Figures 3, 4 and 5 in nonoccluded scenes. It also gives the number of different types of constraints and total number of constraints used. It should be

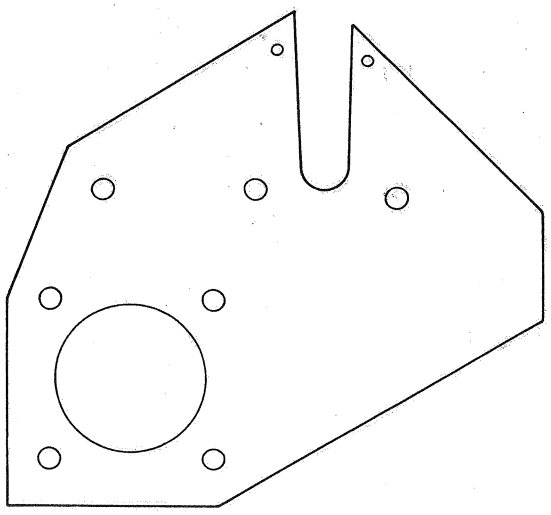

Figure 4. Object number 2

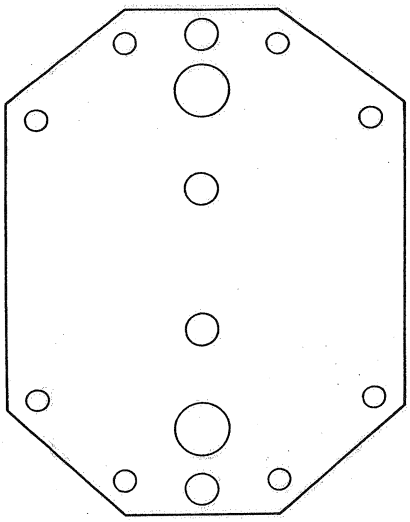

Figure 5. Object number 3

noted that the number of constraints and the constraint types are not optimized. Here only the boundary edges are used for recognition.

Occluded object recognition

Figures 6, 7 and 8 show three scenes where some parts are occluded. These scenes were searched for occurrence of objects 1, 2 and 3 respectively. Table 5 gives the time taken to recognize these parts in the corresponding scene. There are two run times in the table, corresponding to the two schemes used.

CONCLUSIONS

In this paper we have formulated the scene analysis (SA) problem as a consistent labelling problem (CLP). We also gave a solution to the problem within the framework of discrete relaxation methods. However, the approach is based on a different perspective in order to exploit the inherent parallelism in the problem and to account for occlusion. We have also recommended several variations to improve the efficiency of the computation. Also, unlike many other works, we have used more than binary constraints. In fact, we effectively handle unary, binary, ternary and quaternary constraints.

Although the approach sounds attractive, particularly if a multiprocessor is accessible, it by no means solves all the problems. First, we are still dealing with 2D models. Extending the approach to 3D has its obvious challenges. Although we do take into account a certain amount of error, both in the models and during the

Table 5. Results for occluded scenes

Scene number	Run time 1 (ms)	Run time 2 (ms)
1	1360	6307
2	1581	2074
3	3791	26452

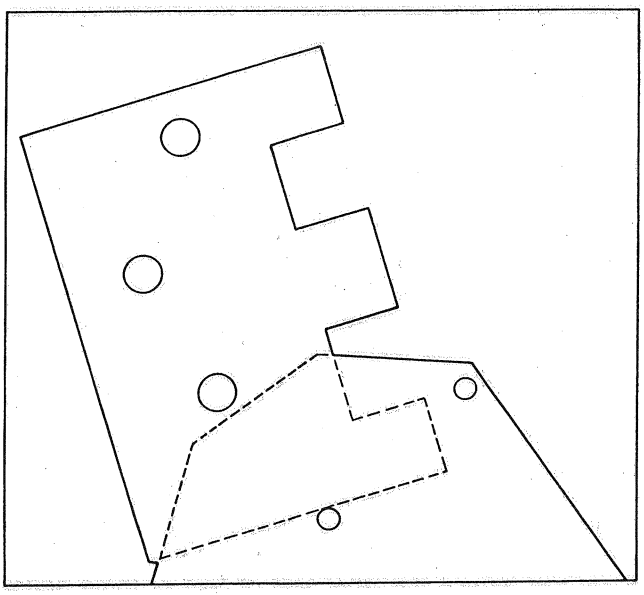

Figure 6. Occluded scene number 1

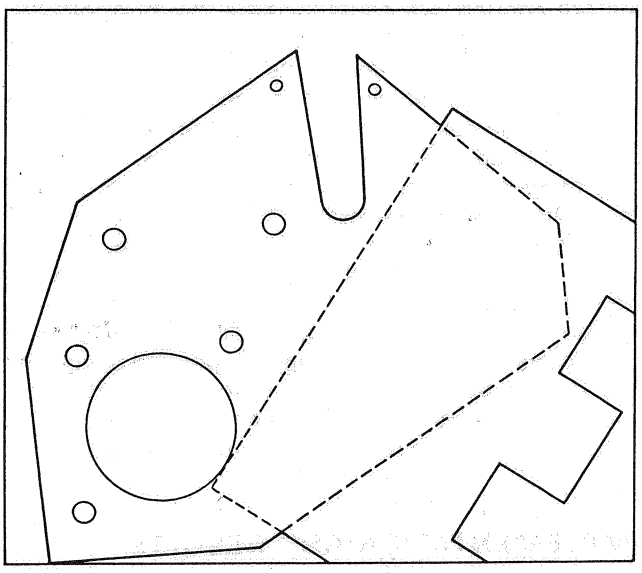

Figure 7. Occluded scene number 2

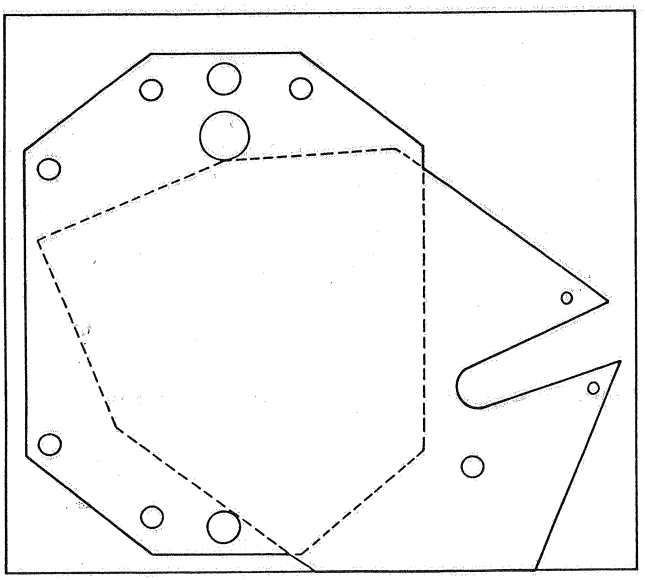

Figure 8. Occluded scene number 3

low-level processing, the error measurements are not based on any sound theory to analytically determine them.

We have shown that global constraints can be useful, and we can effectively use them with local constraints. A problem we have not considered is the optimality of the constraint set. We have used the constraints which were obvious in the models and the image. However, some of them are redundant and there should be a systematic way to eliminate the redundancy.

ACKNOWLEDGEMENTS

This work was supported in part by NSF Grants ECS-8307483, MCS-82-21750, DCR-8506393 and DMC 8502115.

REFERENCES

1 **Haralick, R M and Shapiro, L G** 'The consistent labelling problem: part I' *IEEE Trans. Pattern Anal. Mach. Intell.* Vol 1 No 2 (April 1979) pp 173–184

2 **Gaschnig, J** 'Performance measurement and analysis of certain search algorithms' *PhD thesis* Carnegie-Mellon University, Pittsburgh, PA, USA (May 1979)

3 **Mackworth, A K** 'Consistency in network of relations' *Artif. Intell.* Vol 8 (1977) pp 99–118

4 **Waltz, D** 'Understanding line drawings of scenes with shadows' in **Winston, P H (ed.)** *The psychology of computer vision* McGraw-Hill (1975) pp 19–92

5 **Olivier, F and Price, K** 'Semantic description of aerial images using stochastic labelling' *IEEE Trans. Pattern Anal. Mach. Intell.* (November 1981) pp 633–642

6 **Haralick, R M and Elliot, G** *Increasing tree search efficiency for constraint satisfaction problems* Technical Report, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, VPISU, Blacksburg, VA 24061, USA (March 1979)

7 **Haralick, R M, Davis, L S, Rosenfeld, A and Milgram, D** 'Reduction operations for constraint satisfaction' *Information Sciences* Vol 14 (1978) pp 199–219

8 **Haralick, R M and Shapiro, L G** 'The consistent labelling problem: part II' *IEEE Trans. Pattern Anal. Mach. Intell.* Vol 2 No 3 (May 1980) pp 193–203

9 **Rosenfeld, A, Hummel, R A and Zucker, S W** 'Scene labelling by relaxation operations' *IEEE Trans. Systems, Man, and Cybernetics* Vol 6 No 6 (June 1976) pp 420–433

10 **Montanari, U** 'Networks of constraints: fundamental properties and applications to picture processing' *Information Sciences* Vol 7 (1974) pp 95–132

11 **Mohr, R and Henderson, T C** 'Arc and path consistency revisited' *Technical Report UUCS-85-101* The University of Utah (August 1985)

12 **Knuth, D E** 'Estimating the efficiency of backtrack programs' *Mathematics of Computation* Vol 29 (January 1975) pp 121–136

13 **Davis, L S and Henderson, T C** 'Hierarchical constraint processes for shape analysis' *IEEE Trans. Pattern Anal. Mach. Intell.* Vol 3 No 3 (May 1981) pp 265–277

14 **Henderson, T C and Davis, L S** 'Hierarchical models and analysis of shape' *Pattern Recognition* Vol 14(1–6) (1981) pp 197–204

15 **Ayeche, N and Faugeras, O D** 'A new method for the recognition and positioning of 2 D objects' *ICPR* (August 1984) pp 1274–1280

16 **Bolles, R C and Cain, R A** 'Recognizing and locating partially visible workpieces' *PRIP* (June 1982) pp 498–503

17 **Bhanu, B and Faugeras, O D** 'Shape matching of two-dimensional objects' *IEEE Trans. Pattern Anal. Mach. Intell.* Vol 6 No 2 (March 1984) pp 137–156

18 **Bolles, R C and Cain, R A** 'Recognizing and locating partially visible objects: the local-feature-focus method' *Inter. J. Robotics Research* Vol 1 No 3 (1982) pp 57–82

19 **Aho, A V, Hopcroft, J E and Ullman, J D** *The design and analysis of computer algorithms* Addision-Wesley (1974)

20 **Bolles, R C** 'Robust feature matching through maximal cliques' *Proc. SPIE Tech. Symp. Imaging Appl. Automated Industrial Inspect. Assembly Belingham, WA, USA* (1979)

21 **Papadimitriou, C H and Yannakakis, M** 'The clique problem for planar graphs' *Information Processing Letters* Vol 13(4,5) (1981) pp 131–133

22 **Busacker, R G and Satty, T L** *International series in pure and applied mathematics: finite graphs and networks* McGraw-Hill (1965)

23 **Nitzan, D *et al*** 'Machine intelligence research applied to industrial automation' *Technical Report* SRI International (January 1982)

24 **The Utah Symbolic Computation Group** *The portable standard LISP users' manual* Department of Computer Science, University of Utah, Salt Lake City, UT, USA (1983)

APPENDIX 1: SUBPROCESS STRUCTURE

```
type
  state = (changed, unchanged, working)

process relaxG(i : integer)
begin
  repeat
    Status[i] := working;
    Enforce Arc, Node and Path consistencies in the
    ith graph;
    If there is change in the label-set of any node
    then
      Status[i] := Changed
    Else
      Status[i] := UnChanged
  until terminate
end;
```

APPENDIX 2: CONTROLLING PROCESS

```
var
   m : integer; (* number of graph types *)
   n : integer; (* number of models *)
   k : integer; (* number of features in the image *)
process relaxS(scene, models);
begin
   var i, temp : integer;
       Count   : integer;
       status  : array [1 .. m] of state;
       N       : Set of nodes;
       L_all   : Set of Features;

   N := F^s;   (* Features in image are nodes *)

   L_all := ∪_{i=1}^{n} F_i^m;

   build the image and model graphs;
   for i := 1 to k do Label-set of node N_i := L_all;
   for i := 1 to m do Status[i] := changed;

   Count := 0;
   for i := 1 to m do fork-process(relaxG, i);

   while (Count ≠ m) do
   begin
      temp := 0;
      for i := 1 to m do
         if (Status[i] = unchanged) then
            temp := temp + 1;
      Count := temp;
   end;

   terminate := true;
   find-solutions();
end;
```

APPENDIX 3: TOP LEVEL PROCESS

```
process relaxM;
begin
   while (there is an unmarked feature in the image)
   do
   begin
      f := first unmarked feature;
      O := find-objects(f); (* find objects it can be a
      part of *)
      for (every object O_i in O) do
         fork-process(relaxSS, O_i, f);
   end;
end;
```

APPENDIX 4: PROCESS FOR WORKING ON A SUBSCENE

```
process relaxSS(O : object, f : feature);
begin
   S := features-in-bounding-circle(O, f)
   for each feature f in S do mark(f);
   relaxS(S, O);
   if object O is confirmed then
      mark all its features as explained;
   T := all other features in S;
   if T is not empty then
   begin
      unmark every feature in set T;
      restart process relaxM;
   end;
end;
```

APPENDIX 5: PROCEDURE FOR COMPUTING THE RADIUS OF THE BOUNDING CIRCLE

```
function BCircle (f, F) : real;
begin
   var max, dist : real;

   max := 0.0;
   for all features f_i in F do
   begin
      dist := distance(f, f_i);
      if (dist > max) then max := dist;
   end;

   BCircle := max;
end;
```

APPENDIX 6: COMPUTATION OF SPECIAL DISTANCES

```
procedure ComputeSpecialDistances;

begin
   Compute the threshold-distance;
   for i:= 1 to No-of-Features do
      for j:=i+1 to No-of-Features do
         If distance(feature[i], feature [j] <= threshold-
         distance then
            iv = Make-inter-feature-vector(feature[i],
            feature[j]);
            Append(IVs, iv)
         endif;
   size := GetSize(IVs, error)
   H := Make-histogram(IVs,size)
   for i:=1 to size do
      if H[i] <= Nunique then
            Append(Special-Length-List, details(i));

      endif;
   Store the Special-Length-List in Model;

end;
```