School of Computing

A Theory of Scene Understanding and Object Recognition

Craig Dillon

This thesis is presented as part of the requirements for
the award of the Degree of Doctor of Philosophy
of the
Curtin University of Technology

September, 1996

Acknowledgements

This research described in this dissertation was supported by a Curtin University Postgraduate Scholarship and a Higher Education Contribution Scheme Exemption Award.

I must make special mention in thanking my supervisor and mentor, Professor Terry Caelli, whose enthusiasm, encouragement and humour have made my time as a PhD student exciting and rewarding. I am also indebted to Professor Walter Bischof, Associate Professor Svetha Venkatesh and the members of the "Vislab" (alias "Nerds") for many useful and inspiring discussions.

The technical and administrative staff at both Curtin and Melbourne Universities have provided excellent resources without which this dissertation would not be possible. I must also thank my colleagues at these universities for creating a most enjoyable working environment.

This dissertation is dedicated to Kerri, Beverly and Carl who provided unending support and encouragement.

Abstract

This dissertation presents a new approach to image interpretation which can produce hierarchical descriptions of visually sensed scenes based on an incrementally learnt hierarchical knowledge base. Multiple segmentation and labelling hypotheses are generated with local constraint satisfaction being achieved through a hierarchical form of relaxation labelling. The traditionally unidirectional segmentation-matching process is recast into a dynamic closed-loop system where the current interpretation state is used to drive the lower level image processing functions. The theory presented in this dissertation is applied to a new object recognition and scene understanding system called *Cite* which is described in detail.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Learning about and understanding the environment is critically important to a number of existing and emerging challenges including autonomous navigation, environmental monitoring, surveillance and content specific image queries. Passively viewing the external world is a consistent and informative way automata can obtain local area data. A common passive modality is through visual sources such as CCD cameras. This thesis presents a theory of object recognition and scene understanding which could enable an autonomous robot or other visual processing systems to obtain a detailed description of complex natural and person-made worlds and environments.

The earliest forms of object recognition dealt with the task of identifying single isolated objects in one image. This remains the most common form of object recognition today, and despite the increase in complexity of the methods, the results have not improved greatly. Early scene understanding systems typically dealt with person-perspective or map views of natural scenes containing objects such as a buildings, trees, and roads. Within the object recognition research, considerable effort has been spent on solving the part indexing, part clique and knowledge acquisition problems. By comparison, scene understanding systems tend to be pixel-based and are concerned more with identifying probable labels for small image regions rather than locating and labelling entire objects.

In this dissertation, a new theory of object recognition and scene understanding is

presented. A hierarchical knowledge base is used to provide context sensitivity and knowledge driven lower and middle-level visual processing. Incremental supervised learning is used to continually build and update the knowledge base as the system is used. Relaxation labelling drives a closed-loop constraint propagation and hypothesis generation process which enables global consistencies to be resolved from local ambiguities. Minimum view generation is proposed with respect to optimising classification. Hierarchical knowledge driven segmentation provides the basis for the context sensitive visual analysis.

1.1 Contributions of this Dissertation

The primary contributions of this dissertation are in the development of a general theory which integrates both object recognition and scene understanding. To accomplish this, a hierarchical world knowledge base is required, which is built via supervised incremental learning. An important contribution is in the study of how a more contextually structured knowledge base can be used to improve the descriptive power of the vision system, as well as its efficiency. This has not been studied experimentally before.

The secondary contributions of this dissertation involve the development of specific algorithms for knowledge driven segmentation, hierarchical relaxation labelling and the learning paradigm of explanatory least generalisation. These three new algorithms provide the ground-work upon which a complex vision system can be built. Many existing vision systems are restricted to a simpler knowledge base, and hence do not explore the more complex algorithmic requirements of hierarchical visual interpretation.

1.2 Organisation of this Dissertation

The theory of vision presented in this dissertation is embodied in a computer program called *Cite*. *Cite* contains three inter-connected data structures which represent different aspects of the system; the knowledge base, the scene interpretation and the visual interpretation. The knowledge base structure is described in Chapter 3. Chapter 4 describes the visual interpretation and the scene interpretation structures.

Following these two chapters which deal with representation are the descriptions of the algorithms used in *Cite*. Chapter 5 provides an overview of the operators in *Cite* that create and manipulate the three data structures and their interconnecting hypothesis links. Chapter 6 describes the supervised incremental learning of the knowledge base, Chapter 7 details the hypothesis generation and Chapter 8 describes the relaxation labelling procedures used during recognition.

Following this is Chapter 9 which describes the hierarchical knowledge driven segmentation algorithm and Chapter 10 which details the feature extraction procedures. Partial results are presented throughout the thesis to demonstrate specific algorithms, and these results are listed fully in Chapter 11. The dissertation is summarised in Chapter 12 which includes suggested directions for further work.

Appendix A contains further details of mathematical and algorithmic derivations. Appendix B describes the operation of the *Cite* system and gives a brief overview of the implementation details.

Chapter 2

Related Literature and Proposed Theory

2.1 Introduction

This chapter reviews the published literature in the areas of object recognition systems, scene understanding systems and machine learning technologies. Critical attention is focused on the advantages, disadvantages, abilities amd limitations of the published research, and how this has driven the new ideas presented in this thesis. The chapter concludes with an overview statement of the theory proposed in this dissertation.

2.2 Object Recognition Systems

An object recognition system (ORS) is a methodology, usually embodied in a set of algorithms implemented in a computer, which can take as input colour, intensity and/or range information and produce some form of labelling or description of the objects being viewed. The distinction between an object recognition system and a *scene understanding* system is not clear, although a scene understanding system usually deals with more complex "natural" objects and can sometimes provide a query system for

answering more complex questions than simply what is in the image. Other vision system modalities such as tracking systems and robot navigation systems are not classed as object recognition systems. They may be able to detect the presence of an object (for the purpose of tracking it or avoiding it), but unless they also attempt to label the object with some meaningful description, they will not generally be classified as an ORS.

There are a number of fundamental issues addressed by all object recognition systems. Primarily there is the question of whether the system contains pre-programmed models of the objects or whether it will learn these models from sensed data. Secondly there is the question of whether the objects will be represented using viewer-centred or object-centred features, and whether these are implicit or explicit. Thirdly there is the matching strategy used to determine which object in the database most accurately matches the object in the image.

Other issues revolve around the features used, whether the system operates on range images, intensity images, colour images or some combination of these. There are also the questions of low level filtering and segmentation, whether the system works from regional parts or edge parts or both, and whether the system attempts any type of verification by rendering or remodelling the image in some way.

Each of these properties of object recognition systems will be discussed in this section. Table 2.1 contains a summary of the systems reviewed in this section in terms of these properties. Following this are sections exploring each property and the contributing research in detail.

2.2.1 Knowledge Acquisition

In order to recognise objects or scene elements, any object recognition system must contain a database of knowledge (a *knowledge base*). This knowledge can take a wide variety of forms, and can be acquired in different ways.

There are two main categories of knowledge acquisition; it can be learnt or it can be

System	Year	Knowledge Acquisition	Representation	Recognition Procedure
Garvey [Garvey, 1976]	1976	learnt	implicit	histogram
ACRONYM [Brooks, 1981]	1981	programmed	object-centred explicit	matching search
Local Feature Focus [Bolles and Cain, 1982]	1982	CAD analysis	explicit object-centred	local feature focus
Osima and Shirai [Oshmia and Shirai, 1983]	1983	learnt	object-centred explicit	region
3DPO [Bolles and Horaud, 1986]	1986	learnt CAD	explicit features	pruned hypothesis
Faugeras and Herbert [Faugeras and Herbert, 1986]	1986	learnt (range)	explicit viewer-centred	hypothesis tree search
Grimson and Lozano-Pérez [Grimson and Lozano-Perez, 1986]	1986	programmed	explicit	tree search
SCERPO [Lowe, 1987]	1987	Programmed 3D wire frame	explicit	geometric consistency, ranking
Evidence Based Recognition [Jain and Hoffman, 1988]	1988	learnt (range)	bounded evidence rules	parallel certainty inference
Fan et al [Ting-Jun Fan and Navatia, 1989]	1989	learnt (range)	explicit (graph)	pruned graph matching
Hansen and Henderson [Hansen and Henderson, 1989]	1989	CAD analysis	object-centered explicit	strategy trees
Murray et al [Murray, Castelow and Buxton, 1989]	1989	CAD wireframe	explicit	geometric matching
Wong et al [Wong, Lu and Rioux, 1989]	1989	CAD analysis	Attributed hypergraphs	Graph monomorphism
3D Objects from Image Contours [Kriegman and Ponce, 1990]	1990	programmed	Explicit surface of rotation	algebraic Elimination
BONSAI [Flynn and Jain, 1991a]	1991	learnt from CAD models	implicit, interpretation trees	tree search, verification
Evidence Based System [Caelli and Drier, 1994]	1994	learnt (range)	bounded rules, relational	parallel inference

Table 2.1: Summary of Object Recognition Systems

programmed. Learning can be done either from examples or from a non-sensed representation of the objects. Learning from examples yields the greatest level of automation, learning from other representations is less automated, and acquiring knowledge by direct programming yields no automation.

Computer aided design (CAD) representations are the most common in knowledge acquisition from a non-sensed representation. In systems such as BONSAI [Flynn and Jain, 1991b], analysis of CAD models of objects is used to create an interpretation tree. At run time, sensed data (in this case, range data) is examined and label hypotheses are generated from this precomputed structure. Interpretation trees have also been constructed to represent three dimensional polyhedral objects [Grimson and Lozano-Perez, 1986]. In this approach, the interpretation tree is constructed from CAD models

by examining the range of distances, angles, directions and triple product signs between pairs of surface points and their local surface normals. Extensions to this work [Murray and Cook, 1988] examine other edge-based constraints extracted from the model and data geometries.

One interesting approach is to precompile part graphs based on the process of graph path hypothesis generation as used in the local feature focus (LFF) method [Bolles and Cain, 1982]. In this method, explicit knowledge representation is optimised for the matching process. This is achieved by ranking significant parts according to rotational symmetry and structural equivalence. The explicit nature of this representation reduces generalisation, however this approach does represent some matching optimisation.

A knowledge base can be acquired by examining a training set of input images to determine salient features which are then used during recognition. The evidence based approach [Jain and Hoffman, 1988] induces sets of evidence rules, of which each element is a weighted bounded attributed rule. For example, an object perimeter in the range (10.0, 20.0) may give strong support ($w_A = 1.0$) for object A, no support ($w_B = 0.0$) for object B and weak refutation ($w_C = -0.5$) for object C. The rulebase is generated by removing inconsistent edges from a minimal spanning tree built in a labelled feature space. In this approach an inconsistent edge is defined by a number of heuristics operating on the topology of the minimal spanning tree.

One central problem in 3D object recognition is that of rotational variance of objects. Most objects, whether they be man-made or natural, have high rotational variance. One frequently finds that views of different objects have greater similarity than different views of the same object. A common approach to solving this problem is to first partition the object view space into what is termed a *view sphere* in which each view is essentially treated separately. An alternative approach is to treat each view as a seperate example of the object and construct the knowledge base assuming that multiple rules will be required to adequately cover all views of the object, such as in the evidence based approach [Caelli and Drier, 1994].

The view-sphere representation can be constructed from CAD models [Flynn and Jain,

1991b], or from different views of the object [Seibert and Waxman, 1992]. In both cases the number and Euler position of each view can be specified heuristically or by analysis of the object in question. These methods have some disadvantages. In both cases a complete object description must be available for analysis, and it is easy to show that the resulting aspect graphs are not necessarily the minimal view representation for a given object *with respect to the knowledge base*. This thesis solves these problems by defining the minimal aspect graph as that which permits the recognition of each object against the entire knowledge base.

An interesting variation on the multiple view theme is to use a motion sequence of views to provide temporal ordering of expected views [Seibert and Waxman, 1992]. This temporal sequencing creates a further constraint which can be applied during recognition of moving objects. A number of different schemes exist for determining the number and position of the views in a view-sphere. Regular tessellation of the sphere of up to 320 views have been proposed [Flynn and Jain, 1991b]. Random tessellations of up to 100 views have also been proposed and evaluated [Raja and Jain, 1992].

Explicit representation schemes (see Section 2.2.2) match explicitly to codifications of sensed images. In this sense there is only weak learning because there is little or no generalisation. For example, explicit surface patch representations of industrial parts [Faugeras and Herbert, 1986] cannot be generalised to determine salient features, thus not accommodating variations in objects belonging to the same class. This is entirely suitable to industrial part inspection where a valid error model may be available, but much less suitable to objects that cannot have such explicit descriptions (such as many natural and man made objects). In general, explicit representations permit only weak generalisation and the knowledge acquisition is a trivial process of data storage.

In the Smoothed Local Symmetry representation [Connell and Brady, 1985], ANALOGY is used to generalise examples and to create reduced representations. Inductive generalisation is used, but only on positive examples of the object at each node in the visual hierarchy. To prevent gross over-generalisation, disjunctions of rules are formed and later consolidated into single rules where appropriate. This process is controlled by a set of heuristics which determine the distance a new object is from existing rules and

whether this will result in over-generalisation.

Alternative schemes using neural networks as the knowledge base have also been attempted. In the CRESCEPTRON framework [Weng, Ahuja and Huang, 1993], edge features are used as the input to a multi-layer self-organising network. Translation and scale invariance is achieved by naively scanning the image with a foveation window scaled by 20% at each pass. The system is not 2D rotationally invariant, and no consideration is given to 3D rotational invariance. As expected and demonstrated, such approaches are valuable in 2D pattern recognition, but are of less importance to 3D object recognition because of the lack of part indexing, multiple view representations and geometric constraint satisfaction.

2.2.2 Object Representation

Generally speaking, knowledge representation can be *explicit* or *implicit.* In an explicit representation object recognition system, the knowledge base contains a direct or complete representation of each object. For example, if one stores CAD models or bit-map images of objects then this is an explicit representation.

Implicit representations are usually characterised by partial or minimal knowledge coded in an operationally efficient way. For example, a decision tree with simple decision rules in each node and object labels at each leaf node is an implicit representation of the objects in the database. In general, an explicit representation will contain a richer description of the objects and will not be organised well for matching, while an implicit representation contains less data and is better organised for matching.

BONSAI [Flynn and Jain, 1991a] represents objects implicitly in the form of an interpretation tree. Each level within this tree represents an additional part labelling, and the depth of the tree is determined by label and pose uniqueness. Each object is distributed widely across the tree, which is optimised for run-time efficiency. The principal disadvantage for the application of such methods in this research is the requirement of full CAD models for the construction of the knowledge base. One objective in this

thesis is to build the knowledge base only using sensed data. The interpretation tree representation has also been applied to bin-of-parts problems [Ikeuchi, 1987].

Another implicit representation is that used in the evidence based approach [Jain and Hoffman, 1988] where objects are represented across a large set of evidential rules which, when activated, combine to provide a final classification hypothesis.

Explicit representations tend to be used in research aiming for exact pixel-level matching, such as in industrial inspection or in the recognition of objects from their contours or surface representations. For example, objects may be represented by surfaces of rotation [Kriegman and Ponce, 1990] and algebraic matching is done on the detected contours, or in the form of CAD models described by declarative and functional codification [Goodman, Haralick and Shapiro, 1989]. While this work is less relevant to this thesis where the main concern is in the classification from minimal representations, one sub-goal is the provision of seamless integration of explicit representations into the one scheme.

Another common representation is that of surface patches segmented from range images [Faugeras and Herbert, 1986]. To form this representation, the 3D surface normal is estimated at each pixel in a range image of the object. Regions are then formed and merged against a global error measure based on least squares fits to low order polynomial models. Differential geometry [Besl and Jain, 1985] and zero-crossing analysis of principal curvature [Fan, Medioni and Navatia, 1987] may also be used to segment range images into surface patches (see Section 2.2.5).

ACRONYM [Brooks, 1981] uses an explicit representation in the form of a compositional hierarchy of ribbons (generalised cones) and ellipses in a database of objects. From this a prediction graph is generated which contains invariant and quasi-invariant properties that need to be matched in the image. The database of objects is hand-coded and the system's ability to generalise is expressed as algebraic relationships between properties of parts. In this sense, ACRONYM can provide quite a powerful matching across wide classes of manufactured objects with multiple parts. However, natural objects are not considered in this approach.

A generalised form of Gray coding has been used to describe 2D object boundaries [Connell and Brady, 1985]. This work is interesting because the scalability of the boundary representation was used to form simple taxonomies of objects (in this case common hand tools). The geometric nature of these boundary representations tends to restrict the visual hierarchies to scale based decompositions. Some modifications to the boundary representation have been explored which add symbolic descriptions of parts at different levels in the hierarchy (such as *end*, *corner*, *bump* or *smooth*).

Wireframe and line drawing representations are also useful in recognising many man-made objects. Line drawings may be interpretted as three dimensional surfaces [Barrow and Tenenbaum, 1981] by a process of line classification and surface interpolation. This may be assisted by alternative approaches in which lines are classified by their singularities and junctions [Nalwa, 1988].

2.2.3 Matching Strategies

Pattern recognition is primarily concerned with the matching and categorisation of data according to its first order (unary) properties. Object recognition is made more complex by the requirement that individual parts and their relationships (binary properties) must be matched. The matching strategies used in object recognition are varied and diverse, covering both explicit and implicit knowledge representations.

When explicit object representations are used, the matching strategy is critical as there is generally no precompiling of decision strategies into the knowledge base. In contour matching of surfaces of revolution [Kriegman and Ponce, 1990], algebraic elimination can be used to reduce projection and image contour equations to provide a distance measure between an object and image contour $\mathcal{C}$. The best matched object has to be found by testing all objects against $\mathcal{C}$, or by indexing parametric functions of closest fit models (in [Kriegman and Ponce, 1990] this was the ratio of inner to outer radii in a torus fit to $\mathcal{C}$).

Some object recognition schemes use a parts-and-their-relationships representation. Ex-

plicit forms of this representation require a complex structural matching process. The attributed graph representation [Ting-Jun Fan and Navatia, 1989] is matched using a bipartite unary matching, with relational information being factored in to evaluate the quality of each candidate matching. In this approach, an initial pruning is performed based on simple measures such as the number of nodes, the number of planar nodes and the area of the largest surface patch.

Studies have been carried out in *inexact* graph matching [Shapiro and Haralick, 1981] with the clear result that a pruned tree search using backtracking and/or forward look-ahead is more efficient than computing exhaustive sub-graph isomorphism. The general understanding that determining full sub-graph isomorphism is NP-complete has attracted considerable research effort to the problem. Unfortunately, in practical object recognition systems the number of parts per view of object is typically very small. The amount of computation saved by these more elegant algorithms when applied to real objects is generally negligible in comparison to the linear database search between each sensed object and each object in the database.

Explicit surface patch representations can be matched to the test object by a pruned tree search [Faugeras and Herbert, 1986]. This is achieved by generating surface patch correspondence hypotheses based on unary constraints, and then pruning the tree enumeration by binary constraints and heuristics applied to each feature type.

Interpretation trees provide an efficient implicit representation which is optimised for matching. BONSAI [Flynn and Jain, 1991a] and others [Grimson and Lozano-Perez, 1986] use interpretation trees constructed from CAD models of the objects to be recognised. The interpretation generation process is a simple and efficient one of descending the tree, computing features at each level within the tree and pruning sub-trees when they are deemed to be invalid interpretations. Global validity for an interpretation can be established by using partial hypotheses to determine the scale, rotation and translation which link the model to the sensor [Murray and Cook, 1988]. The features which are used to prune such interpretation trees are described in Section 2.2.5.

Similar to tree search, the local feature focus method [Bolles and Cain, 1982] matches to candidate objects by moving from one feature (the focus feature) to the next in a sequential manner. Matching strength is based on unary attribute compatibility with the next feature being chosen from a pre-compiled local cluster of features. Conditional rule generation (CRG) [Bischof and Caelli, 1994] descends a rule hierarchy until unique labellings are found. In this approach, each rule level represents alternatively a unary and a binary feature space resulting in the accumulation of label evidence over a sequence of parts and their relations. The region based matching of Oshima and Shirai [Oshmia and Shirai, 1983] is similar to CRG in that unary-binary chains are analysed for global compatibility.

A directly geometric approach to matching 2D edge images subject to 3D rotation is explored in SCERPO [Lowe, 1987]. In this approach, the knowledge base is an explicit 3D wire-frame representation of the object being searched for. Perceptual groupings of sensed image edges are made such that these groupings are most likely to be invariant under 3D rotation. These groupings are made on the basis of the proximity of line endpoints, the degree of line parallelism, and the degree of line collinearity. Object hypotheses are then made subject to a probabilistic ranking of these perceptual groupings which is also pruned by a pre-computed visibility property (a *visible hemisphere*) of each group. SCERPO has been demonstrated effectively on classic bin-of-parts problems in industrial inspection. It is not possible to generalise the notion of perceptual groupings to cover hierarchical objects or objects that are described by properties more complex than edges without also needing to come to terms with issues of part indexing and matching generalisation. The explicit programmed nature of the knowledge base in SCERPO does not facilitate generalisation, yet does permit precise geometric matching to well defined objects. This is a common tradeoff in object recognition systems.

Evidence may be accumulated for each hypothesised object using *recognition networks* [Bolle, Califano and Kjeldsen, 1992] which sum multiple hypotheses and use geometric and image model constraints to refine these. This approach has not yet been demonstrated, and is confined to model based 3D object recognition.

When multiple objects are present in the scene, there is a non-trivial process of breaking

this into most probable groupings of objects, termed the *parts clique* problem. One heuristic approach to this problem is to split the multiple object graph along lines of weak matching [Ting-Jun Fan and Navatia, 1989], and then complete the single object matching on the remaining graph components.

Relaxation labelling has been proposed as a method for solving the parts clique problem [Kim and Kak, 1991]. In this approach, initial estimates are obtained using bipartite matching, and then relational compatibilities are propagated using discrete relaxation labelling.

2.2.4 Verification Processes

After generating a set of possible label and orientation hypotheses, BONSAI [Flynn and Jain, 1991a] verifies each by rendering a range map and comparing this at a pixel level to the sensed data. While this is successful in certain applications, for more natural objects of less definition such as "tree" or "grass", this would not work. This method would also work poorly on intensity data unless very accurate models of material and lighting were deducible, which is itself an extremely complex problem (for example, [Kay and Caelli, 1994]). However, BONSAI does represent a successful application of feedback in computer vision. It is worth noting that although an implicit representation is used for hypothesis of location and orientation, an explicit CAD model is used during the verification process.

SCERPO [Lowe, 1987] verifies object position and rotation by projecting a visible wire-frame model (derived from a 3D line model) onto the sensed edge image. Newton iteration is used to shift, scale and rotate the projected lines to best match the image lines. Although this will correct for small errors in initial hypotheses, it is not used to generate new hypotheses. As such, this single stage verification process is still only open-loop and does not represent true closed-loop feedback.

2.2.5 Segmentation and Feature Extraction

Any image, whether range, colour or intensity, contains a large amount of data that needs to be partially summarised before object recognition or scene understanding can occur. A common approach is to break the image into regions of common property, or into edges and other salient features such as corners or holes. For each of these image *tokens*, properties can be computed describing the token (called *unary* features) and the relationship between this token and others (called *binary* features). Higher order features have been proposed [Flynn and Jain, 1991b] and implemented [Flynn and Jain, 1991a].

BONSAI uses range data only, and computes the four unary part attributes of area, surface type (planar, cylindrical and spherical), and surface intrinsic parameter equality. It also uses four binary attributes which are parallel plane distance, common view visibility, pairwise relative orientation, and a global model-centred orientation. Segmentation into parts is achieved by pruned connected components analysis of clustered surface normals (using program CLUSTER [Michalski and Stepp, 1983a]).

Other interpretation tree approaches [Grimson and Lozano-Perez, 1986] use features such as surface distance, intersecting angle, surface orientation and the sign of surface normal triple products to prune the interpretation process. These are typical features applied when 3D range data is available. Shape-only features such as angles and relative directions have also been used [Murray and Cook, 1988] to provide a degree of scale invariance in matching 3D polyhedral models.

Differential geometry is a popular method for segmentation of range images. When dense range images area available, mean and Gaussian curvature can be computed from local difference operators [Besl and Jain, 1985]. Surface classification into eight fundamental types can be achieved by thresholding each of the mean and Gaussian curvatures into positive, zero and negative ranges. Table 2.2 lists each of these basic surface types. Note that the Gaussian curvature cannot be positive when the mean curvature is zero.

Mean Curvature	Gaussian Curvature	Surface Type
H < 0	K > 0	Peak
H > 0	K > 0	Pit
H < 0	K = 0	Ridge
H > 0	K = 0	Valley
H = 0	K = 0	Planar
H = 0	K < 0	Minimal
H < 0	K < 0	Saddle Ridge
H > 0	K < 0	Saddle Valley

Table 2.2: Surface Type by Mean and Gaussian Curvature

The segmentation of range images using mean and Gaussian curvature has the advantage that these measures are 3D rotationally invariant, but the disadvantage that they are quite sensitive to noise. The numerical estimates of mean and Gaussian curvature can also be used as unary surface patch features [Caelli and Drier, 1994] [Barth, Caelli and Zetzsche, 1993], although they are sensitive to scale.

In the evidence based recognition approach [Jain and Hoffman, 1988], range images are segmented by clustering on spatial coordinates (x, y, z) and surface normal components (n_x, n_y, n_z) estimated from a 5x5 window. Morphological properties of perimeter and the number of background connected patches are used. Individual surface patches are described by their surface type, size, span and boundary linearity. Boundary relations (binary properties) are described by the boundary type (jump, adjacent or remote), the angle between patch normals, the minimum and maximum distance between patches, the boundary angle and the jump gap distance (where appropriate).

Explicit representational schemes [Kriegman and Ponce, 1990] extract detailed object contours using standard edge detection methods. Such recognition schemes can only be applied to smooth boundary, high contrast objects with low surface texture. Other systems that use simple object extraction methods [Seibert and Waxman, 1992] typically do basic extraction and log polar transformations on single object images. Contour based descriptions of objects can also be determined by identifying association relations between contours from full volumetric data [Wang and Aggarwal, 1986].

For industrial parts classification and location [Bolles and Cain, 1982], corners, edges

and holes may be used to assist in recognition, especially where back-lit (silhouette) 2D intensity images are available.

Algebraic descriptions of surfaces and edges can also be used as unary features in vision systems. Quasi-topological features such as convexity, loop and connectivity [Nishida and Mori, 1992] can describe images curves.

Image segmentation is a non-robust process, and a number of approaches have been applied to the process of *resegmenting* the image according to the ongoing interpretation. Heuristics have been used to alter thresholds and control parameters in the resegmentation process [Wang and Srihari, 1988] [Nazif and Levine, 1984]. Knowledge based approaches have also been explored using either domain specific knowledge in the resegmentation of aerial images (the SPAM system [McKeown, Harvey and McDermott, 1985]), or by changing the segmentation operator and control parameters based on expectation [Hwang, Davis and Matsuyama, 1986].

2.2.6 Range Versus Intensity in Object Recognition Systems

Object recognition systems typically use intensity, colour or range information as the primary input. Intensity and colour images are represented as two dimensional arrays of pixels, where each pixel represents the surface illumination or colour at a point projected back to the camera. Range images are usually represented as two dimensional arrays of depth values where each depth value represents the *distance* from the camera to a particular surface point.

Range information can be computed directly using active means such as time-of-flight laser range finders, active lighting shape-from-shading, or projective strip-light scanning. The resulting range maps are usually quite dense with valid range estimates at most pixel locations. Range information can also be estimated passively using multiple cameras in a stereo configuration, shape from focus, passive-lighting shape from shading and shape from local texture gradients. Stereo vision is the most popular, but has been extended to more than two cameras [Dhond and Aggarwal, 1991] and to other

novel methods such as shape from virtual aperture focus [Dillon and Caelli, 1993] and the plenoptic camera [Adelson and Wang, 1992].

The choice of whether to use range or intensity/colour information as the input into an ORS depends on the desired application of the system. Natural outdoor scenes and aerial image analysis tends to provide little useful depth variation, although airborne stereo has been studied and laser range finders have been used successfully in autonomous roving vehicles. Under some circumstances, dense range images are more robustly segmentable than colour images. However, most matching algorithms can be adapted to use either range or colour information. The exceptions to this are the methods which require range data to generate hypotheses about the three dimensional properties of surface patches for the purpose of image level object fitting. Intensity and colour images tend to provide a much greater range of computable features.

Computing depth information from stereo is a popular modality despite the fact that it gives reasonably poor results due to the sparse nature of the computed depth map. The high level of interest in this approach is probably due to the fact that most animals have two eyes, although it is clear that stereo vision is not required for animal object recognition. If this were not the case, the visual world would disappear into a meaningless jumble if we had only one eye open, which clearly does not occur. Stereo vision is, however, useful in robotic object avoidance and high accuracy stereoscopic applications such as photogrammetry.

2.3 Scene Understanding Systems

In contrast to the object recognition systems described in Section 2.2, scene understanding systems try to label multiple complex objects in images which are usually of natural outdoor scenes. The methods used in scene understanding systems tend to be less precise and more evidential in approach.

2.3.1 SCHEMA

The SCHEMA system [Draper, Collins, Brolio, Hanson and Riseman, 1989] is a major attempt at a general purpose knowledge based scene interpretation system. The fundamental aspect of SCHEMA that sets it apart from most other scene interpretation systems is that both knowledge and computation are partitioned at a coarse-grained semantic level. The knowledge base contains a set of schemas, each of which is designed to recognise one particular class of object. This differs from most machine vision approaches which usually focus on one representation and one matching strategy.

Each schema has access to a gobal *blackboard* which contains the current interpretation of the scene, and can post and receive messages to this global space asynchronously. An instance of the appropriate schema is generated for each hypothesised instance of the object in the scene. Each schema instance has access to what are termed *knowledge sources*. The knowledge sources, which operate at a number of different levels, provide analysis of the image and resultant image structures. For example, there are lower level feature extraction and segmentation knowledge sources, and higher level knowledge sources covering initial hypothesis generation, token clustering and knowledge driven resegmentation.

The knowledge base in SCHEMA is hand coded by a knowledge engineer in declarative langugage which covers the *endorsement space*, *confidence function* and *control strategies* for each schema. The endorsement space contains structures which can accumulate evidence for an instance of the object represented by the schema, and the confidence function describes how to combine these various endorsements. The control strategy

section of each schema determines how the support space is searched by providing flow control over the sequence of knowledge sources accessed by the schema. Although demonstrated to be successful on natural images with a small number (15) of objects in the knowledge base, the requirement for each schema to be *programmed* requires the presence and attention of an expert.

The features used in SCHEMA cover colour, texture, shape, size and location of regions and other basic scene elements. The colour features include the mean and standard deviation of the red, green, blue, excess red, excess green, excess blue, and the three colour components of the HSV and YIQ colour spaces. Excess red is calculated as $2R-G-B$, and the YIQ colour spaces are calculated as linear combinations red, green and blue combinations.

2.3.2 SIGMA

The SIGMA system [Matsuyama and Hwang, 1990] is a complete image understanding system designed for aerial image analysis. It contains three modules for low level vision, model selection and geometric reasoning, as well as a query module through which the user interacts. These modules are interconnected such that top-down and bottom-up vision processes are closely intergrated. The interpretation within SIGMA is in the form of a *part-of* hierarchy with image features at the leaf nodes and spatial relations describing the higher level structure.

SIGMA's knowledge base is object oriented in the sense that instances of an object are created (instansiated) dynamically from base object classes. Each object has three main knowledge components; unary properties of the object, relationships with other objects, and control information to guide the analysis process. The control structures are triplets of the form (condition, hypothesis, action).

After an initial segmentation, seed hypotheses are generated and stored in the *iconic / symbolic database* which represents the instantaneous scene interpretation. These initial hypotheses result in bottom-up hypothesis generation of higher level object in-

stances. These, in turn, result in the top-down hypothesis generation of missing parts. The geometric reasoning expert iterates the interpretation cycle through hypothesis generation, evidence accumulation and the resolution of the most reliable composite hypothesis. This cycle continues until no new hypotheses are generated and the resulting global interpretation is stable.

SIGMA contains a sophisticated low level vision module which performs goal directed image segmentation. A *goal* in this system is expressed as a set of constraints and properties that the solution must satisfy, some of which specify the environment in which the solution is to be found. One strength of this system is the ability to search the image at a very low level for expected image features.

SIGMA has been demonstrated on aerial images of new housing developments and robustness to shadows and poor image contrast has been shown. As with SCHEMA, the knowledge base for SIGMA is hand-coded by an expert, and the system as a whole has not been demonstrated to recognise more than a few different object classes.

The texture measures in SIGMA include the per-unit strength of horizontal, vertical and diagonal lines, and the energy and entropy of the intensity histogram. Lines are also grouped according to length and contrast, and measures of theire density included as texture measures.

Regions are described using compactness (the ratio of the square of perimeter to area), minimum bounding rectangle (MBR) orientation, MBR fill ratio, region height to width ratio and the ratio of the region perimeter to the MBR perimeter. Region size is described in terms of absolute pixel count, perimeter length, region height and width, and MBR height and width, all in units of screen pixels. SIGMA also describes each region's location in terms of the absolute screen position of the upper left corner, lower right corner and region centroid.

2.4 Machine Learning

Machine learning and computer vision have, in general, remained separate disciplines within computer science. From a computer vision perspective, this has resulted in less emphasis on learning relational structures in the machine learning literature, and an ignorance in the computer vision literature of the power of many machine learning algorithms.

For this review, a suitable taxonomy of machine learning systems is to classify them according to their application domain, the underlying learning strategies used and the representation of knowledge [Carbonell, Michalski and Mitchell, 1983].

In terms of the *application domain* we are primarily interested in classification systems. Other learning application domains such as computer programming, planning and problem-solving will not be considered unless the knowledge representation is also applicable to classification.

With respect to the *underlying learning strategy*, the theories presented in this dissertation primarily involve supervised learning from positive and negative examples. This may be incremental or non-incremental, although the final algorithms developed will all be incremental. Unsupervised learning can still be valuable in situations where a data model is known, such as in image segmentation. Contrary to some beliefs, unsupervised systems such as COBWEB [Fisher, 1987] and CLUSTER/2 [Michalski and Stepp, 1983b] do contain a data model (usually in the form of metric on the attribute space).

The issue of *knowledge representation* is critical in developing any machine learning system. Of relevance to this investigation are decision trees, graph and network representations, frames and schemas, taxonomies and hybrids of these. Other knowledge representations which are less suited to the classification problem include formal logic expressions, procedural codifications, sets of production rules and other formal grammars.

2.4.1 Single Point Unary Classification

The single-point unary classification problem can be stated as follows. Consider a vector space of dimensionality n, and a set of m points in that vector space each of which is labelled with an integer in the range 1 to r. The objective is to determine a function F which will map all of the points onto their labels. Once this function has been determined, new unlabelled points can be classified. If F is well designed, the new data points will be correctly classified and the system will have "learnt" to classify new data. There is a necessary requirement that the initial labelled data cover all labels ($m \geq r$) and it is typical to have many examples of each label ($m \gg r$).

There have been a number of approaches to generating the classification function F. One of the most common is through decision trees. Each non-leaf node of a decision tree contains a decision rule which operates on one attribute (or feature). These rules can be designed to operate on discrete or continuous data, and can branch to two child nodes (a binary decision tree) or a larger number of child nodes. The leaf nodes in the decision tree contain the final classification label.

ID-3 [Quinlan, 1986] determines the attribute which is to be operated on in each node by measuring the information theoretic minimum entropy partitioning of the data. At each node, the attribute and decision threshold for that attribute which minimise the partition entropy of the training data in that node is chosen. The decision tree is built recursively in this manner until the entropy at each leaf node is zero (corresponding to training data at that leaf node which all have the same classification label).

FOIL [Quinlan, 1990] is also a data-driven induction method which specialises conjunctive expressions to determine a set of non-overlapping covers for objects of different classes. The PROTOS system [Bareiss, Porter and Weir, 1988] integrates inductive learning and deductive problem solving in the task of heuristic classification in medical applications.

2.4.2 Relational Learning

The previous section discussed single point, or *unary* classification and learning. This methodology assumes that the input sample is described by a single vector, where each dimension represents some measure of the input instance, and the order of these measures is fixed. Object recognition requires an extension to this approach because an object is typically analysed in terms of a number of parts, each of which has specific properties, as well as specific relationships to the other parts in the object.

An object is typically represented as a graph in which the vertices are the parts and the edges are the relationships between the parts. There is no natural ordering of the parts that will be stable to 3D rotation, noise and other distortions, so it is not possible to form a single *stable* vector which describes the full object and all its relationships. If this were possible, unary classification could be used on this full vector. However, because there is no natural ordering of the parts, a large number of permutations exist (order $n!$ in the number of parts) and this would be impractical to be learnt using any unary classification system.

Relational matching schemes include various degrees of solution to the graph isomorphism problem, interpretation trees and evidential approaches. Relational *learning* typically involves the process of constructing the appropriate data structures to complete the relational matching, based either on a training set of images, or on internal object representations such as CAD models. Most relation matching schemes use unary attribute learning, and rely on the run-time matching strategy to provide the desired level of labelling uniqueness.

The Evidence Based System (EBS) [Caelli and Drier, 1994] learns both unary and binary attributes into non-indexed rules. These rules are then mapped using a neural network which is trained using back propagation. This approach does not represent full relational learning, although co-occurrence is learnt by the neural network classifier. Conditional Rule Generation (CRG) [Bischof and Caelli, 1994] performs relational learning of paths through the object graphs. These paths, representing chains of unary and binary information, are stored in the form of a decision tree with the partition

feature being determined by partition entropy.

2.4.3 Hierarchical Knowledge and Learning

Within the artificial intelligence community, hierarchical knowledge representation is common place. However, research into automatically building hierarchical knowledge is less common, and it is virtually non-existent in machine vision. Despite this, there are a number of research outcomes relevant to this dissertation in the area of hierarchical knowledge and hierarchical learning.

Minimum entropy hierarchical clustering algorithms have been proposed [Wallace and Kanade, 1989] which form clusters over unclassified data points using a Gaussian entropy model. While this technique provides interesting results, it is not directly suitable for *supervised learning* machine vision applications such as *Cite.* Clustering does, however, play an important role in the low level process of image segmentation in most object recognition systems. Incremental learning of hierarchical concepts has also been demonstrated in the CLASSIT algorithm [Gennari, Langley and Fisher, 1989] and applied to the unsupervised learning of cardiology data.

2.5 Proposed Theory

This dissertation describes *Cite* which specifically aims to extend and improve the current research in object recognition, scene understanding systems, and machine learning as applied to computer vision systems. *Cite* extends the conventional flat knowledge base to a fully hierarchical one with no depth limitation. Multiple hypotheses are generated for each scene element, and these are resolved using a hierarchical extension to relaxation labelling. Traditional feed-forward segmentation is augmented with knowledge driven resegmentation which closes the control loop on low level vision processes. *Cite* also extends the typical learn-phase then run-phase operational environment with robust incremental learning algorithms which build and improve the knowledge base after each scene has been fully analysed.

An architectural diagram of *Cite* is shown in Figure 2.1. The numbers beside each function block represent the approximate order of operation for each operator. An initial segmentation (1) of the image causes the unary feature calculator to compute features for each of the low level regions (2). These features are then matched with the knowledge base (3) to provide initial labelling hypotheses which are represented in the indexing structure called the *scene interpretation*. Clique resolving and hierarchical binary matching then occur on these initial hypotheses (5) using binary features calculated from the hierarchical segmentation (4). The higher level scene hypotheses are added into the scene interpretation structure, and hierarchical relaxation labelling begins to resolve the multiple ambiguous labels for each object (6).

As the labels begin to resolve, nodes are individually resegmented (7) using parameters stored in the knowledge base. These resegmentations replace the initial segmentations in the visual interpretation structure, resulting in a repeat of the unary and binary feature extraction and matching (stages (2) through (6)). This cycle continues a number of times until the interpretation becomes stable. If *Cite*'s final interpretation is incorrect, the user may chose to incrementally learn the correct object labelling (8) by selecting the incorrectly labelled nodes and the desired knowledge base node. The updated knowledge base is then available as the next scene is viewed.

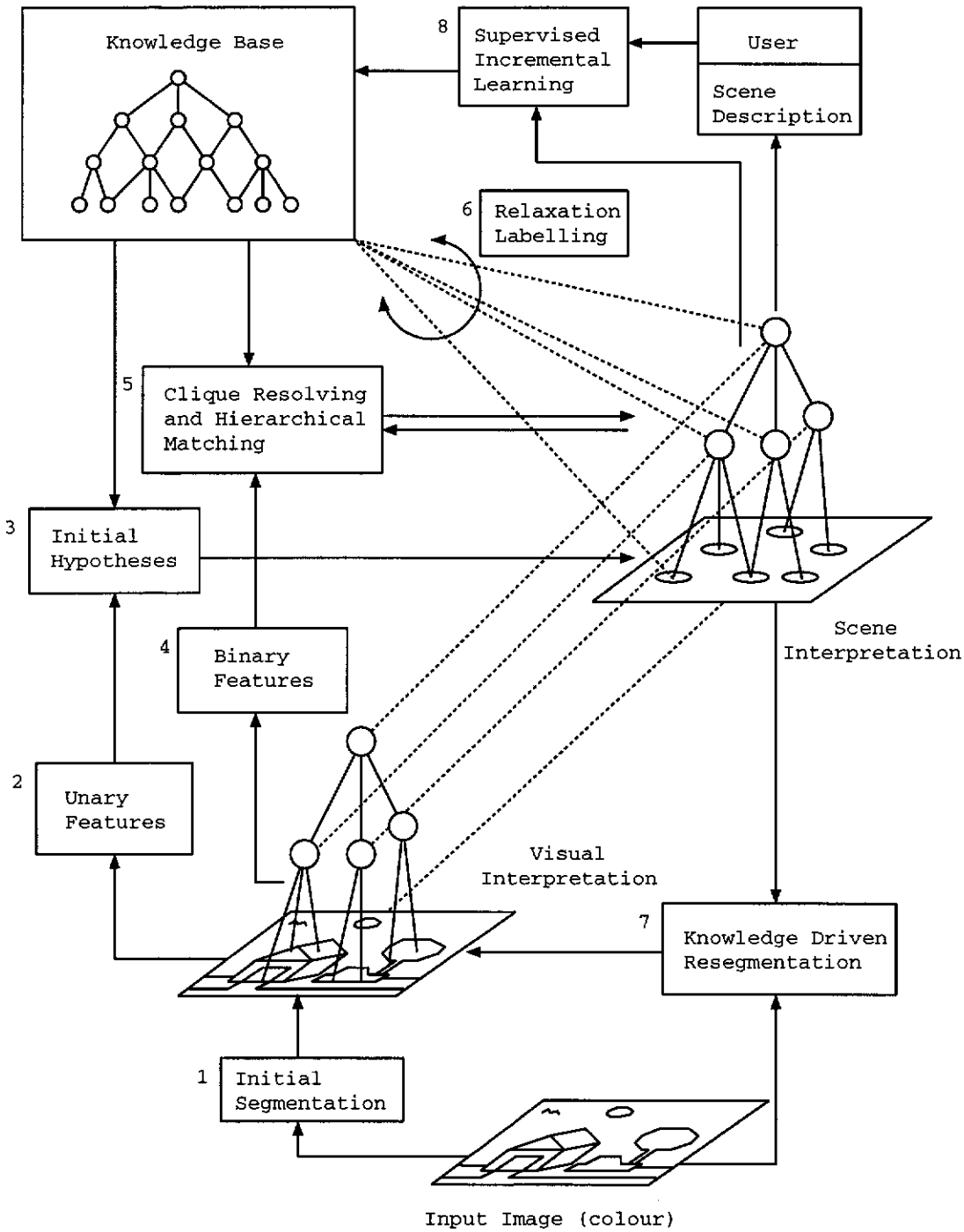


Figure 2.1: **Overview of *Cite* Architecture**

The relaxation labelling, knowledge driven resegmentation and hierarchical clique resolving and matching provide very tight closed-loop feedback within *Cite*. The hierarchical knowledge base provides a rich scene description which can include contextual,

taxanomic and deep decomposition information. The use of incremental supervised learning provides *Cite* with the ability to increase the descriptive power and accuracy of its analyses, as well as to add new world knowledge as it becomes available, rather than requiring the full set of scene objects to be present during an initial learning phase.

Chapter 3

World Knowledge

One of the most important data representations in scene understanding and object recognition systems is the representation of world knowledge. *Cite* represents world knowledge as a semi restricted graph in which each node represents an object or visual concept which is either a **part-of**, a **view-of** or a **type-of** its parent or parents. Figure 3.1 is a simple example of such a knowledge base. Within each node is stored information about the optimal segmentation, feature extraction and matching algorithms that are used to recognise this object in an image.

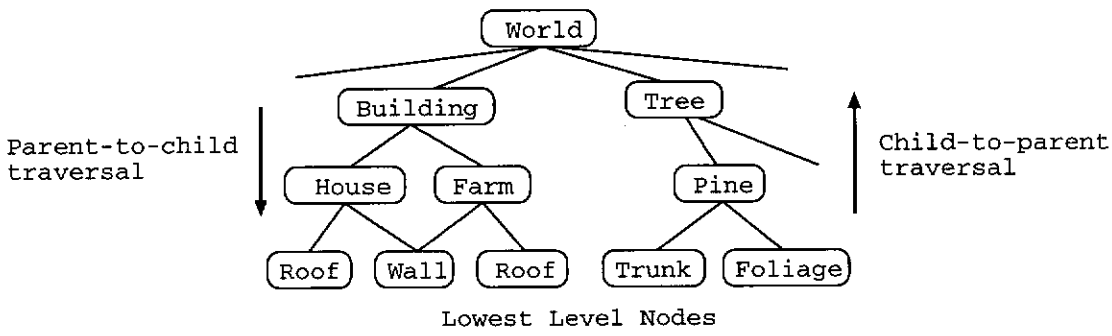


Figure 3.1: **Knowledge Base Graph Example**

Most previous object recognition systems represent world knowledge as a simple linear list of objects, each of which may be described by specified properties, or as a set of parts each of which can be described. *Cite* can easily represent such a data structure,

but because the objective is to use higher level contextual information to describe the scene, the hierarchical knowledge base in *Cite* represents a significant knowledge extension in comparison to conventional object recognition systems.

This chapter describes the data structures used to represent knowledge in *Cite*. The supervised incremental learning processes which build the knowledge base are described in Chapter 6. Chapter 4 describes the data structures used to represent each image, and the scene interpretation data structures which connect the visual analysis of each image to the knowledge base.

3.1 Knowledge Base Structure

The knowledge base in *Cite* is stored as a semi-restricted graph where each node in the graph represents an object or scene element. The objects themselves may be parts of objects or parts of parts of objects, and so on. Each node can have a number of child nodes and a number of parent nodes. The child nodes represent either parts-of, views-of or types-of the parent node.

The graph is restricted in that no cycle in the graph can contain only parent-to-child edges or only child-to-parent edges. There must be at least one child-to-parent and one parent-to-child relationship in every cycle. An important and useful consequence of this is that the graph can be drawn in two dimensions such that for every node its parents are above it and its children are below it.

Figure 3.2 illustrates the parent and child relationships for a node in the knowledge base. Any downward traversal of the graph is treating edges as parent-to-child edges, and upward traversals are child-to-parent edges. All knowledge bases in this dissertation are depicted in this manner. An interesting point is that nodes higher up in the knowledge base represent more general knowledge, and nodes lower down in the knowledge base represent more specific knowledge.

The knowledge base is unbounded with the exception that it must have exactly one

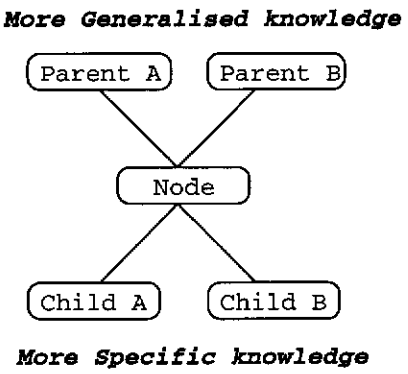


Figure 3.2: **Knowledge Base Parent-Child Relationships**

node with no parents (called "World"). There must also exist a downward traversal path containing only parent-to-child edges connecting the top level "world" node to every other node. Expressed in another way, every node other than the top level node must have at least one parent. A typical knowledge base tends to expand out looking similar to a tree, with the addition of a small number of nodes with multiple parents. Nodes with no children are called leaf nodes and generally represent the smallest atomic scene elements.

3.2 Knowledge Base Node Types

The knowledge base in *Cite* decomposes world knowledge in three ways. A scene element or object may be broken into constituent parts, into different views of the object under 3D rotation, or into different specific types of the object. Consequently there are three types of parent-child relationships; part-of, view-of and type-of. Nodes in the knowledge base always contain children with the same parent-child relationship type. A parent whose children have a part-of parent-child relationship is called a part-of node for simplicity, and similarly for view-of and type-of nodes. Sections 3.2.1, 3.2.2 and 3.2.3 discuss these three knowledge base node types in detail.

3.2.1 Constructed Objects - The Part-Of Node

In the part-of node, the parent is constructed or composed from instances of the children. For example, "car" would be the parent node, and "wheel", "door", "windscreen" and "side-panel" could be the children. The "wheel" node in turn could have two children "tyre" and "rim". A forest could be described as being composed of trees, ground, bushes, and maybe a river.

There are two variants of the part-of node. The "constructed" part-of node describes objects which are composed from a strict number of parts where the relationships between those parts is complex but generally rigid. The "consists" part-of node describes scene elements where the parts are collected together less rigidly to form the collective object. A car would be described using a "constructed" part-of node, while a forest would be described using a "consists" part-of node.

Deep knowledge hierarchies are supported by *Cite* so that it is possible to decompose objects over a number of levels. Part-of nodes within such a hierarchy can be a mixture of "constructed" and "consists" nodes. This permits the construction of less well-defined objects such as a "street-scene" which contains well defined objects such as cars, people, and a road.

Each part-of node contains a unary and a binary matching strategy, usually in the form of rules which are evaluated on candidate image regions or groups of image regions. The unary matching strategy is used to recognise the object as a whole, whereas the binary matching strategy is used to recognise the configuration of the constituent parts.

A special situation arises with leaf nodes in the knowledge base. These are always designated "part-of" nodes and contain a valid unary matching strategy, but no binary matching strategy as there are no children. In this sense it is important to realise that an object may match using data stored at its own node, but relationships with other objects are matched using data stored at the parent node.

3.2.2 Taxonomies - The Type-Of Node

Cite represents taxonomies using the type-of node. For complex hierarchical world knowledge, taxonomies represent valuable reductions in data complexity. A "forest" may contain a number of "tree" objects where each "tree" could be of type "gum", "poplar" or "oak". In the same knowledge base, "my house" might be represented as a "house" object next to a "gum". Specifying a simple tree taxonomy in this situation gives the specificity required for "my house" and the generality required for "forest".

Deep taxonomies can be built using many levels of type-of node and each level offers a different degree of semantic generalisation. Because the knowledge base is built using supervised learning, the taxonomies embedded in the knowledge base are context sensitive. Figure 3.3 illustrates two possible "tree" taxonomies - one that could have been constructed by a botanist, the other by a forester. The variations in these contexts result in differing interactions between the unary and binary learning and the hierarchical relaxation labelling.

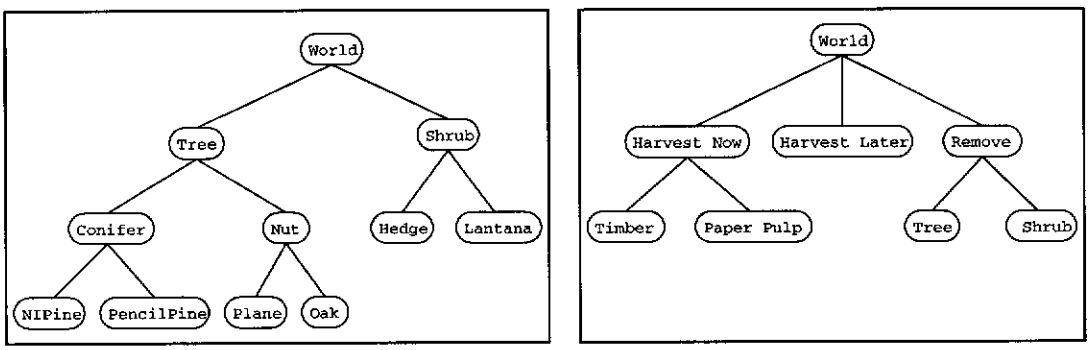


Figure 3.3: **Two Alternative Tree Taxonomies**

3.2.3 View Representations - The View-Of Node

Most objects look very different from different viewing directions. There is often more similarity between two views of different objects than there is between two different views of the same object. *Cite* represents different views of the same object as children

of a common parent. This is achieved through the view-of node.

Some object representations are viewer-centred, meaning that the object is composed from a union of the projected views that the viewer is able to see. Other objects are represented using object-centred descriptions, meaning that only one "view" of the object is required. For example, surface representations are viewer-centred, whereas constructive solid geometry (CSG) is an object-centred representation.

Many object recognition systems make the assumption that their world is best represented by either viewer-centred or object-centred representations, but never a mixture of the two. Often this imposes strict limits on the range of knowledge that can be represented. By permitting any choice of features to represent an object, and by permitting single or multiple view representations, *Cite* avoids this limitation.

The view-of representation is not tied explicitly to a geometric viewsphere interpretation, and consequently *Cite* can represent multiple views of objects using only as many views as are required to distinguish the object from other objects in the knowledge base. In most viewsphere representations a predetermined tessellation of the viewsphere is used to calculate the number of views required, or analysis of CAD models or complete 2D image sets ([Seibert and Waxman, 1992] for example) determines the full set of *characteristic* views. These methods all treat the object in isolation to the other objects in the knowledge base, and hence excessive data can be stored.

3.3 Special Constructions

When dealing with higher level scene elements there are situations where the formal structure of the knowledge base as described in the previous section is too restrictive. In some higher level scene interpretations certain scene elements may be able to exist on their own or as part of another construction.

For example, in interpreting aerial views of an airport, a "loading" may be defined as as the presence of a "plane" and a "terminal" with certain relational (binary) properties.

Similarly, an "emergency" may be defined as a "plane" and a "firetruck" in close proximity. Figure 3.4 illustrates the emergency scene and Figure 3.6 illustrates the loading scene.

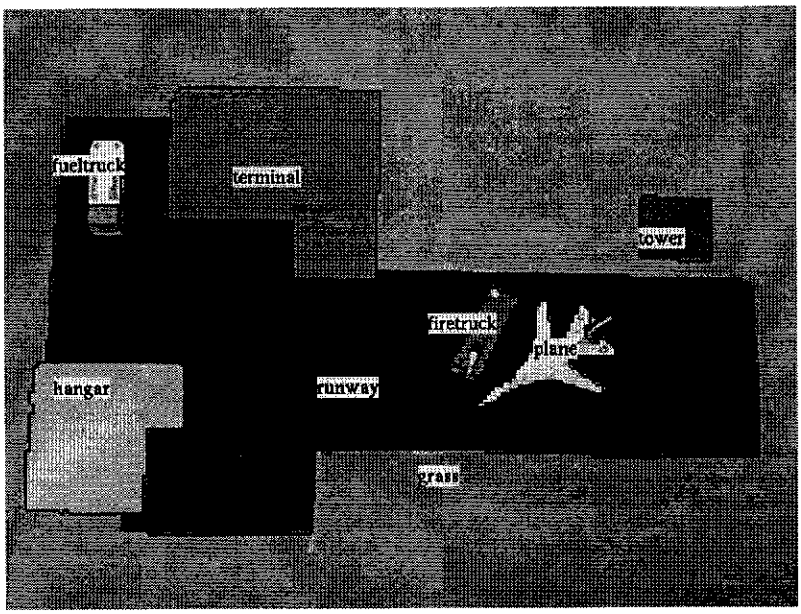


Figure 3.4: **Labelled Emergency Scene Image**

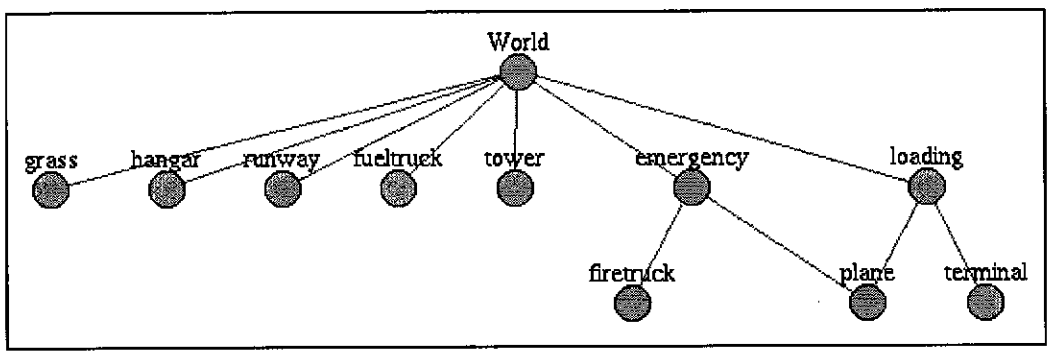


Figure 3.5: **Knowledge Base Supporting Airport "Emergency" and "Load"**

The knowledge base which can support all the concepts depicted in these two airport scenes is shown in Figure 3.5. A complication arises because according to this knowledge base the "terminal" object can *only ever* be part of a "service" operation. In other words, when the system detects a "terminal" there must be a "service" object present, in which case there should also be a "plane" present with the appropriate binary relations.

This clearly fails when, for example, the system views the emergency scene and the "terminal" as simply part of the "world" and not related to anything else.

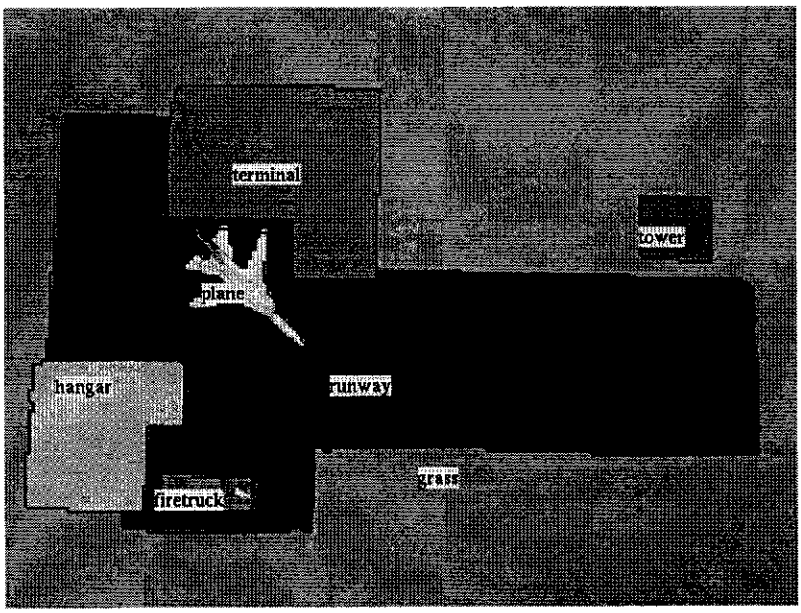


Figure 3.6: **Labelled Loading Scene Image**

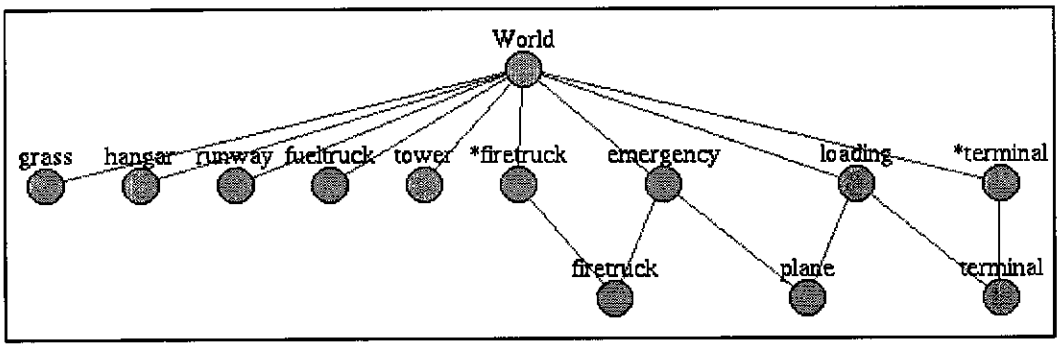


Figure 3.7: **Correct Knowledge Base Supporting Airport "Emergency" and "Load"**

For consistency with the relaxation labelling process it is undesirable to have direct links that permit a node in the knowledge base to have both a parent and a parent's parent as its own direct parents. Links are permitted to span multiple generations, but only as long as these links join nodes to their grand-parent nodes. The solution to this problem is to insert a node in the link between the child and the grand-parent node, as shown in Figure 3.8. This represents the notion of an object possibly not being a part

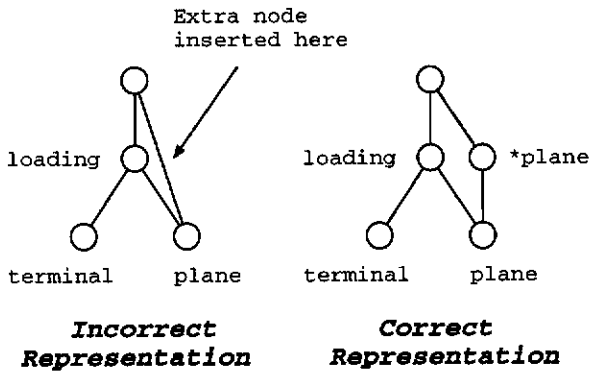


Figure 3.8: **Incorrect(left) and Correct(right) Versions Of Representing Not-Always-Part-Of**

of any higher level construction at this level within the knowledge base.

Such marker nodes are represented as part-of nodes with one child. Because this construction will never be created otherwise, it is easy to detect when generating descriptions of the scene. As such, they add nothing to the complexity of the recognition process, rather they reduce the need for the relaxation labelling and hypothesis generation processes to cope with tight multiple generation links. Nodes used for this purpose are labelled with the name of their child node preceded by an asterisk, as shown in Figure 3.7.

It should be noted that these special case part-of nodes are not treated any differently by any of the internal processes in *Cite* and are only occasionally required in the correct representation of higher level scene elements.

3.4 KB Node Common Properties

Each knowledge base (KB) node contains one list of parents and one list of children. Each element in these lists is another KB node. The parent list is an unweighted list, while the child list is weighted. This is done so that the KB-KB hypothesis weight is stored in only one place. The probability of KB node a being the parent of KB node b is the same as the probability of KB node b being the child of KB node a.

3.5 KB Node Specific Properties

Each of the three KB node types contain specific properties related to the use of the particular KB node type. This section describes these specific properties in detail.

3.5.1 Part-Of Specific Properties

The part-of node contains three additional components beyond the parent and child lists. The first is a flag indicating whether the type-of node is a "consists" node or a "constructed" node. This flag is used by a number of processes in *Cite* to determine whether all children are expected or required.

The other two additional components are the unary and binary matching strategies used by that node. These matching strategies do not necessarily contain descriptions of the objects, rather they contain parameterised processes which can match the object to a given set of data. Both the unary and binary matching strategies are built by supervised incremental learning. This process is discussed further in Chapter 6.

3.5.2 Type-Of Specific Properties

There are no additional specific properties in the type-of node, other than those present in all three knowledge base node types.

3.5.3 View-Of Specific Properties

The view-of knowledge base node could contain Euler angle or other pointing vector notation such that the orientation of the object can be determined once it has been optimally matched. This is currently not done as it is a simple addition which is not relevant to the overall task of decomposing the input images into their constituent scene elements.

Chapter 4

Interpretation Structures

Cite contains a hierarchical segmentation in the form of a semi-restricted graph called the *visual interpretation.* Multiple labelling hypotheses can be generated for each node in the visual interpretation and these are stored in an intermediate structure called the *scene interpretation.* This chapter describes both these structures and concludes with a formal definition of the three data types used in *Cite.*

4.1 Visual Interpretation

Most object recognition systems work from single level segmentations. That is, the image is broken into non-overlapping regions in which each pixel belongs to one region and one region only. It is generally assumed that by direct (open-loop) segmentation these regions will all represent objects or parts of objects, and the objects can be found from appropriate groupings. In the case of recognising isolated objects this latter grouping is not required as all parts (apart from the background) are assumed to belong to the object in question.

Cite differs in that it contains a true hierarchical segmentation in which regions are grouped to form larger regions which are grouped to form even larger regions and so on. There can be multiple visual interpretation graphs for the scene; one for each

image.

Unlike the knowledge base, there is only one type of node in the visual interpretation graph. This is called the VI node and may contain multiple parents and multiple children. Multiple children represent the grouping of smaller regions with common properties or labels into a larger region with a single label. Multiple parents represent the notion of ambiguous set membership, which is important in solving what is called the *clique problem.*

The clique problem is one of the main reasons why object recognition systems that can recognise only single objects cannot be easily generalised to recognise multiple objects. By way of example, consider a crowded street scene where there may be many parts that match as car bodies and car wheels. For any given car body or car wheel part there may be a number of possible ways to group this with other parts to form a complete "car" subject to the expected relationships between the parts. When examined locally, many of these groupings may appear consistent and hence can only be resolved by looking at a more global level. Unfortunately, whenever global consistency is required in computer vision, the complexity of an exhaustive solution rises exponentially. The clique problem can be stated most succinctly as being the process of extracting sub-groups of parts (cliques) to form objects so as to maximise adherence to local constraints within the confines of maximising global consistency.

Cite uses a process of parallel hypothesis generation to generate multiple likely solutions (local constraints) and then relaxation labelling to propagate these hypotheses to maximise global consistency. This process is discussed in detail in Chapter 7, suffice to say at this point that this approach relies on the ability for the segmentation structure to support multiple ambiguous parent memberships for parts.

The remainder of this section describes the data structures used to represent the hierarchical *visual interpretation.* Section 4.2 describes the *scene interpretation* structure which provides indexing links between the visual interpretation and the knowledge base.

4.1.1 Visual Interpretation Structure

There is one node type in the visual interpretation (VI) graph, called a VI node. There is a separate VI graph for each image of the current scene.

Each VI graph is restricted in that no cycle in the graph can contain only parent-to-child edges or only child-to-parent edges. There must be at least one child-to-parent and one parent-to-child relationship in every cycle. Similar to the knowledge base graph, the VI graph can be drawn in two dimensions such that for every node its parents are above it and its children are below it.

The visual interpretation graphs are unbounded with the exception that there is one node with no parents, representing the entire image. Leaf nodes with no children represent the lowest level (finest detail) of image segmentation. Unary (part) and binary (relational) features are calculated and stored at each level in the segmentation hierarchy.

4.1.2 The Visual Interpretation Node in Detail

4.1.2.1 Graph Connections

Each VI node contains a list of parents and a list of children. Each element in these lists is another VI node. The parent list is an unweighted list, and the child list is weighted. This is done so that the VI-VI hypothesis weight is only stored in one place, although for the purposes of analysing the VI structure the hypothesis weight is accessible from either the parent or the child node. Thus, the probability of VI node a being the parent of VI node b is the same as the probability of VI node b being the child of VI node a, as depicted in Figure 4.1.

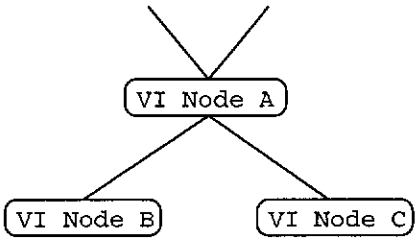


Figure 4.1: **Undirected Hypothesis Weights Connecting VI Nodes**

4.1.2.2 Scene Interpretation Connections

The scene interpretation (SI) structure is an indexing structure which connects the visual interpretation to the knowledge base, and represents the current scene belief state. The scene interpretation structure is discussed in full detail in Section 4.2.

Each VI node contains a weighted list of hypothesis links to SI nodes and each hypothesis link is interpreted as being the visual support for the given scene interpretation node. Multiple SI hypotheses may be connected to a VI node during top-down hypothesis of the existence of missing or occluded parts. Ambiguous clique memberships may also be represented by multiple SI hypotheses on the given VI node.

4.1.2.3 Region Description

Each VI node also contains a list of pixels that the node represents in the original input image. There is no ordering or adjacency limitation placed on these pixels, which means that a VI node can be used to represent multiple non-connected regions. This is particularly useful for representing occluded objects and complex high level scene elements whose constituent parts are not necessarily connected in the image.

The only requirement placed on a VI node's pixel list is that the list be a subset of each parent's pixel list. This implies that at some level in a VI graph a pixel may appear in multiple children of a VI node, representing ambiguous classifications for that pixel. This is *entirely* the objective of *Cite* in supporting multiple contradictory segmentations with the view to resolving them using higher level knowledge. The

VI graphs constructed in this manner efficiently represent multiple segmentations at different levels of detail.

Consider VI nodes c_1 to c_n which are children of VI node p_1, where the child nodes have pixel sets $\{(x,y)\}_{c_1}$ through to $\{(x,y)\}_{c_n}$ and the parent's pixel set is $\{(x,y)\}_{p_1}$. Formally we have that $\{(x,y)\}_{c_i} \subset \{(x,y)\}_{p_1} \forall i \in \{1..n\}$, for all nodes in the VI graph. In the case where all of the child nodes are leaf nodes, we can say that $\bigcup_i \{(x,y)\}_{c_i} = p_j \forall j \in \{1..n\}$. In addition, for leaf child nodes of any given parent the child nodes do not overlap (that is, $\{(x,y)\}_{c_i} \cap \{(x,y)\}_{c_j} = \emptyset \forall i,j,i \neq j$).

In addition to the region description itself, the VI node is also the data structure where image features are stored when they have been calculated. Image features may describe a region (*unary* features), or the relationships between regions (*binary* features). Unary features are stored in the VI node and binary features are stored in the common parent of the multiple VI nodes between which they are calculated. Both unary and binary features can be calculated on a "need-to-know" basis, and the feature operators are discussed in detail in Chapter 10.

4.1.3 The Set of Views and Visual Interpretations

Cite has facility for viewing multiple images of the same scene to establish a more complete interpretation. Each image is represented as an element in the *view-set* which contains both the input image and the visual interpretation structure for each image. There is, however, only one *scene interpretation* structure for each scene being viewed. The visual interpretation nodes all provide SI hypothesis links connecting to this.

Adding multiple views adds the additional complexity of the *object correspondence problem.* If an image contains three trees for example, and another view from a different direction contains those three trees, it is a non-trivial problem to determine which tree corresponds to which in the two images. This is analogous to the depth-from-stereo correspondence problem with the additional complication that the views can be from vastly different positions making some properties such as the baseline and crossover constraint unusable.

4.2 Scene Interpretation

Most object recognition systems work only from single images of single or small numbers of different objects. The internal representation tends to be a list of parts, each with one label which represents the most common object part. In more sophisticated systems, there may be multiple labels for each part or a first level of grouping parts together to form objects, each of which has a label. In these systems, the knowledge base is shallow and the data representation for hypotheses is a straight forward connection from each part to a node in the knowledge base.

Unfortunately for more complex scene analysis this direct connection of regions to labels is not sufficient. For example, we may know that a region exists but it may belong to two possible objects. In this instance, the region requires two part labellings. Without a further indexing scheme it will not be clear to each of the two parent object labels which of the child labels necessarily supports its own hypothesis. This situation is shown in Figure 4.2. When computing the support for the node labelled VP-KP1 it is not clear how to interpret the multiple VC-KC1 and VC-KC2 hypothesis labellings. Of course, one could search the parent-child relationships to look for ancestry, but in the generalised graph where the hierarchy may be considerably deeper this form of search becomes very costly.

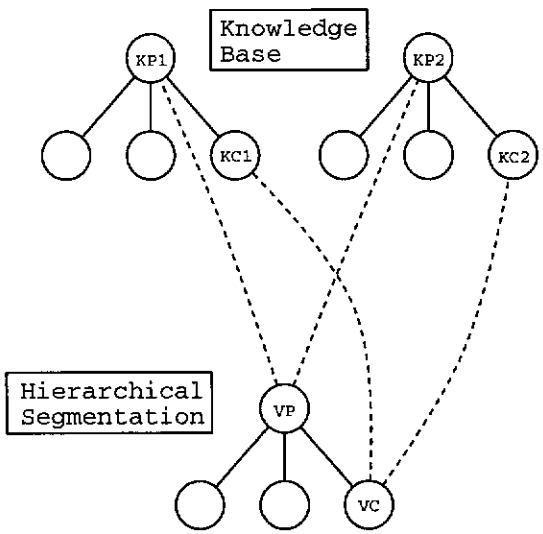


Figure 4.2: **Illustration of Inadequacy of Direct Labelling**

Another problem arising from the direct labelling depicted in Figure 4.2 is that the instantaneous interpretation of the scene is a piece-wise linear addition of multiple interpretations. That is, the scene would be described as containing an object of type KP1 or KP2, with a child of type KC1 or KC2. A more appropriate interpretation would be that the scene contains either a KP1 with a KC1 child, or a KP2 with a KC2 child. Again, this interpretation could in most cases be deduced by examining the labelling paths to determine the most likely outcome. However, this will not work in general and will be computationally expensive across deep hierarchies.

The problem occurring in Figure 4.2 is that there is no relational information between the various hypotheses. The relational information required is in the form of a compatibility set which groups hypotheses into mutually consistent groups. For a hierarchical scene description these compatibility sets must also be hierarchical. The most appropriate structure is a graph of similar construction to the knowledge base and visual interpretation, which is called the *scene interpretation.*

The direct labelling example in Figure 4.2 is shown with the intermediate scene interpretation (SI) structure in Figure 4.3. The instantaneous scene description, including all the appropriate ambiguities, can now be read directly from the SI graph. The scene now contains two alternative interpretations, embodied in nodes labelled S1 and S2. These scene interpretation nodes establish the compatabilities between hypotheses such that, for example, the KP1-VP hypothesis cannot now be associated with the KC2-VC hypothesis. When one considers that *Cite* operates over deep hierarchies with a knowledge base where a node may belong to a number of parents, the scene may contain multiple instances of the same object and where multiple views of the scene need to be integrated, the scene interpretation graph becomes a fundamental requirement.

In many ways the scene interpretation graph represents a full analysis of the scene. However, it contains no data other than the hypothesis links to the KB and VI nodes. All of the segmentation and feature information (the data) is stored in the visual interpretation. All the world knowledge and matching strategies are stored in the knowledge base. In this sense, *Cite*'s three data structures represent an elegant and formal separation of a vision system into the knowledge, data and interpretation constituent parts with a generalised hypothesis scheme connecting the three structures.

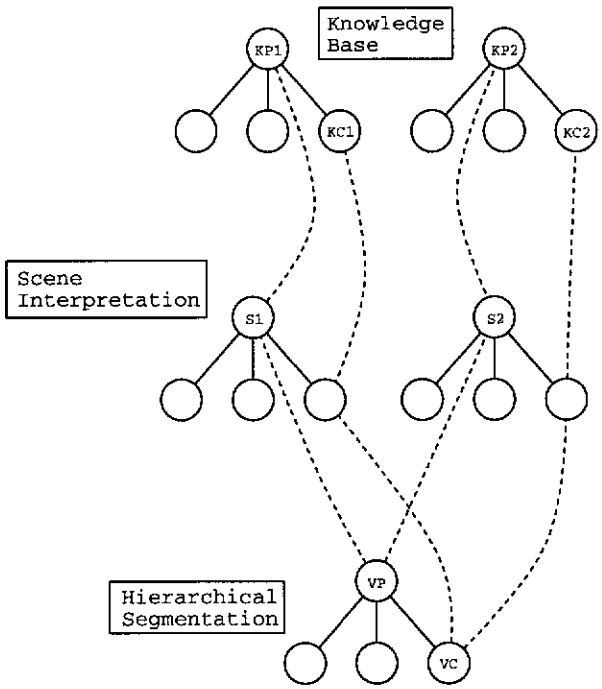


Figure 4.3: **Hierarchical Compatibility Sets via the Insertion of the Scene Interpretation Structure**

4.2.1 Scene Interpretation Structure

There is only one node type in the scene interpretation (SI) graph, called an SI node. Like the knowledge base, the SI graph is restricted in that no cycle in the graph can contain only parent-to-child edges or only child-to-parent edges. There must be at least one child-to-parent and one parent-to-child relationship in every cycle.

The scene interpretation graph is unbounded with the exception that there is one node with no parents as this represents the entire scene. Leaf nodes with no children represent the lowest level (finest detail) of scene analysis.

4.2.2 Scene Interpretation Node in Detail

4.2.2.1 Graph Connections

Each SI node contains a list of parents and a list of children. Each element in these lists is another SI node. The parent list is an unweighted list and the child list is weighted. This is done so that the SI-SI hypothesis weight is stored in only one place. The probability of SI node a being the parent of SI node b is the same as the probability of SI node b being the child of SI node a, in exactly the same way as described for the visual interpretation structure (see Section 4.1.2.1).

4.2.2.2 Visual Interpretation Connections

Each SI node contains a weighted list of VI nodes that are interpreted as being the visual support for that particular scene element. Multiple VI nodes may be connected to a SI node during top-down hypothesis of the existence of missing or occluded parts.

The weighting for the SI to VI hypotheses is identical to the VI to SI hypotheses (that is, the VI-SI hypothesis is undirected). The actual weight for each VI-SI hypothesis is stored in the SI node.

4.2.2.3 Knowledge Base Connections

Each SI node contains a weighted list of knowledge base nodes that represent possible object labellings for that SI node. Each weight in the KB node list can be seen as a probability, and as a result of the normalisation process these will add to 1.0.

Unlike other inter-node connections in *Cite*, the SI-KB connections are not duplicated as KB-SI connections. This is because the knowledge base is the last point of reference and there are no operators that traverse the knowledge base needing to refer back to the scene. This separation of scene interpretation and knowledge base is another motivation for moving the inter-object indexing away from the knowledge base and into

the distinct data structure of the scene interpretation graph.

Multiple SI-KB labellings represent an ambiguous classification and these are resolved through a process of relaxation labelling. This process is described fully in Chapter 8. The SI-KB hypothesis weight is stored in the SI node, and in conjuction with the VI-SI weights, these connections represent the main "interpretation state" of *Cite* at any given moment.

4.3 A Formal Definition of the VI, SI and KB Structures

Definition 1: *A C-node is a graph vertex containing a set of edges to parent C-nodes and a set of edges to child C-nodes. In each C-node these two sets contain no duplicates and are disjoint.*

Definition 2: *A C-graph is a semi-restricted graph of C-nodes which has one C-node with an empty parent set, and all cycles contain at least one parent-to-child edge and at least one child-to-parent edge.*

Definition 3: *A Part-Of node is a C-node additionally containing a text label, a unary matching strategy and a binary matching strategy.*

Definition 4: *A Type-Of node and a View-Of node are C-nodes additionally containing a text label.*

Definition 5: *The knowledge base (KB) is a C-graph with un-attributed edges where each C-node is a Part-Of, Type-Of or View-Of node.*

Definition 6: *A visual interpretation (VI) node is a C-node additionally containing a set of (x,y) pixel locations, a set of unary features, a set of binary features, a segmentation history list, and a set of attributed hypothesis links to the scene interpretation.*

Definition 7: *A VI graph is a C-graph where each edge is attributed with a single weight in the range [0,1] and each vertex is a VI node.*

Definition 8: *A scene interpretation (SI) node is a C-node additionally containing a set of attributed hypothesis links to KB nodes and a set of attributed hypothesis links to VI nodes.*

Definition 9: *A SI graph is a C-graph where each vertex is a SI node.*

Definition 10: *A unary matching strategy is an algorithm containing the required data and processes to compute the degree of similarity to a VI node, and to adaptively update these data and processes to incorporate positive and negative examples in the form of*

VI nodes.

Definition 11: *A binary matching strategy is an algorithm containing the required data and processes to compute the degree of similarity to a set of VI nodes with a common parent, and to adaptively update these data and processes to incorporate positive and negative examples in the form of a set of VI nodes with a common parent.*

Cite contains one knowledge base graph as defined in Definition 5. For the current scene, *Cite* contains one scene interpretation graph as defined in Definition 9, which is renewed as each scene is viewed. For each image of the scene there is one visual interpretation as described in Definition 7.

Chapter 5

Operational Overview

The two previous chapters described the knowledge base and scene interpretation data structures used in *Cite*. These following chapters describe the operators and algorithms which construct and manipulate these data structures. This chapter provides an overview of these processes.

Many vision systems have a simple flow-through architecture. Typically this involves an image segmentation stage, followed by a feature extraction stage, followed by a matching stage. In each stage the data flows in one direction only and there are no other control signals. Some systems, such as BONSAI [Flynn and Jain, 1991a] do have a coarse-grained feedback mechanism that makes corrections to the object hypothesis by image back-projection. *Cite* attempts to further refine these feedback processes, and the resultant complexity requires a more structured description.

By way of introduction, Figure 5.1 shows an example of the three main data structures in *Cite* and the operators which construct and manipulate these data structures. In this chapter a simple example is followed through in detail, followed by a brief description of each operator. Figure 5.1 serves primarily as an example road-map for the operation of *Cite*, and Figure 2.1 (at the end of Chapter 2) should be consulted for a description of the temporal data flow sequence.

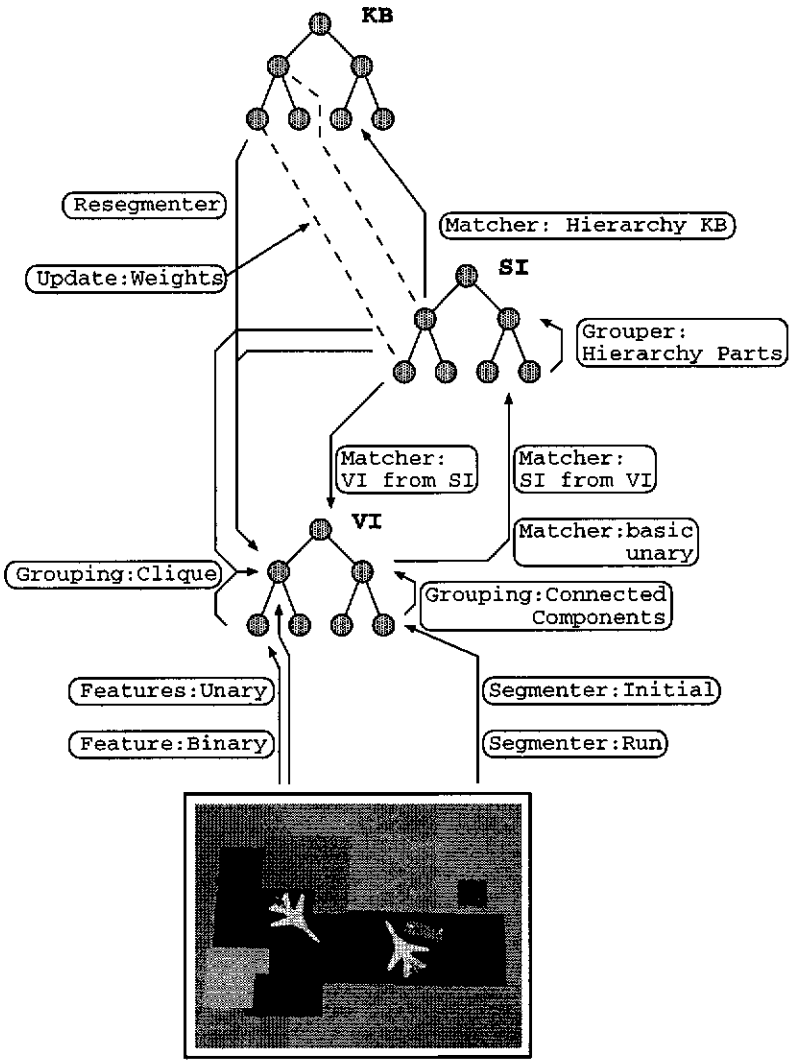


Figure 5.1: **Overview of Data Structures and Operators within** *Cite*

5.1 A Simple Example

For the purposes of illustrating the function and data flow surrounding the main operators in *Cite*, a simple example of recognising a Norfolk Island Pine using the Botanist knowledge base is considered. The sample image and knowledge base (KB) are shown in Figure 5.2.

When *Cite* first loads an image a default top level VI node is created. The segmentation initialisation process detects that this VI node has not yet been segmented and begins this process. As no recognition has occurred yet, *Cite* uses the top level KB node to

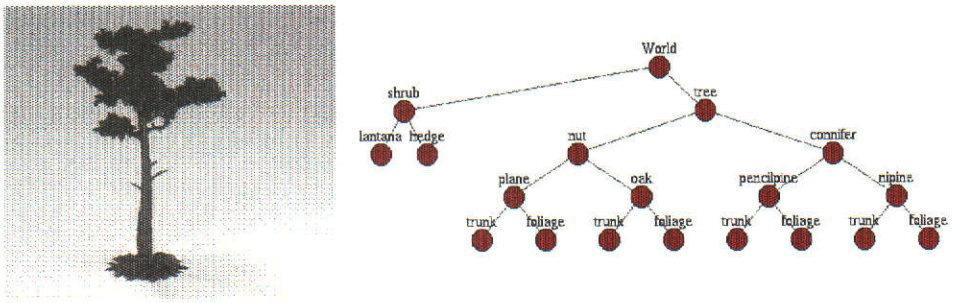


Figure 5.2: **Norfolk Island Pine Image and Botanist Knowledge Base**

determine the initial segmenter and its parameterisation. This information is stored inside the KB node, and can be viewed by the user in a small pop-up window as is shown in Figure 5.3.

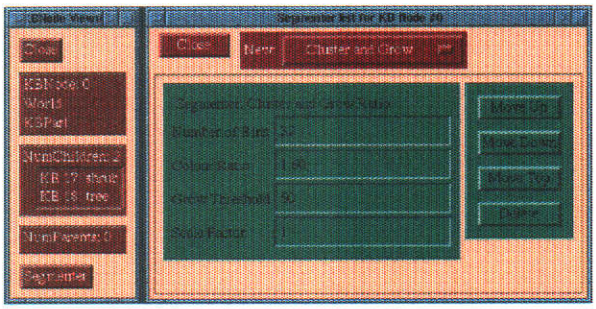


Figure 5.3: **Top Level KB Node Segmenter used for Initial Segmentation**

The segmentation operator then executes the segmentation process. This operator can execute multiple independent segmentations on different parts of the image at the same or overlapping times. The last thing the segmentation operator does before terminating is to generate the VI nodes corresponding to each region. Figure 5.4 shows this initial segmentation and the VI graph at this point. Note that this initial segmentation has resulted in an over-segmentation of the image. The objective is to end up with just two parts, the trunk and the foliage, as described in the knowledge base.

After the initial segmentation occurs, the unary feature extraction operator detects unattributed VI nodes and begins calculating the unary features for these nodes. At the same time, the connected components grouper constructs a parent VI node representing the collection of VI leaf nodes. For this example, the connected components operator

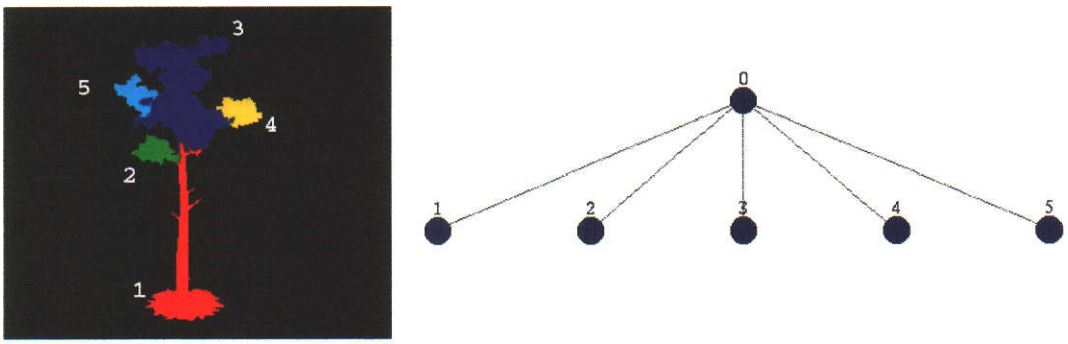


Figure 5.4: **Initial Segmentation and Corresponding VI Graph**

is used instead of the clique resolving operator for simplicity. The grouped VI nodes are then processed by the SI-from-VI matching operator. This begins the process of generating the scene interpretation graph. Once the scene interpretation has been started, the basic unary matching occurs. This matches the leaf VI nodes with the knowledge base and generates SI hypotheses linking the SI nodes to the knowledge base. When this is complete the hierarchical knowledge based matching operator detects missing levels in the scene interpretation and fills these in. The VI-from-SI matching operator then propagates these down to the VI graph.

Figure 5.5 illustrates the three graph structures at this point in the analysis process. Several SI nodes have been selected by the user (indicated by being coloured) which also displays their KB and VI hypotheses. The weights on each link indicate the initial matching strengths before the relaxation labelling process has begun.

In conjunction with the SI and VI node construction operators, the knowledge driven resegmentation operator scans the SI graph for nodes that may require resegmentation. This process is discussed in more detail in Chapter 9, suffice to say at this point that although *Cite* believes the object to be an oak tree it clearly has too many children and can be resegmented using the segmentation parameterisation stored in the "Oak" KB node. This resegmentation results in the dismantling of most of the VI and SI nodes, leaving just the parent object node which it knows is a "Tree".

Figure 5.6 shows the new data graphs after the system has stabilised. The hypothesis for the object is now that it is a "nipine" (Norfolk Island pine), which is correct. Note

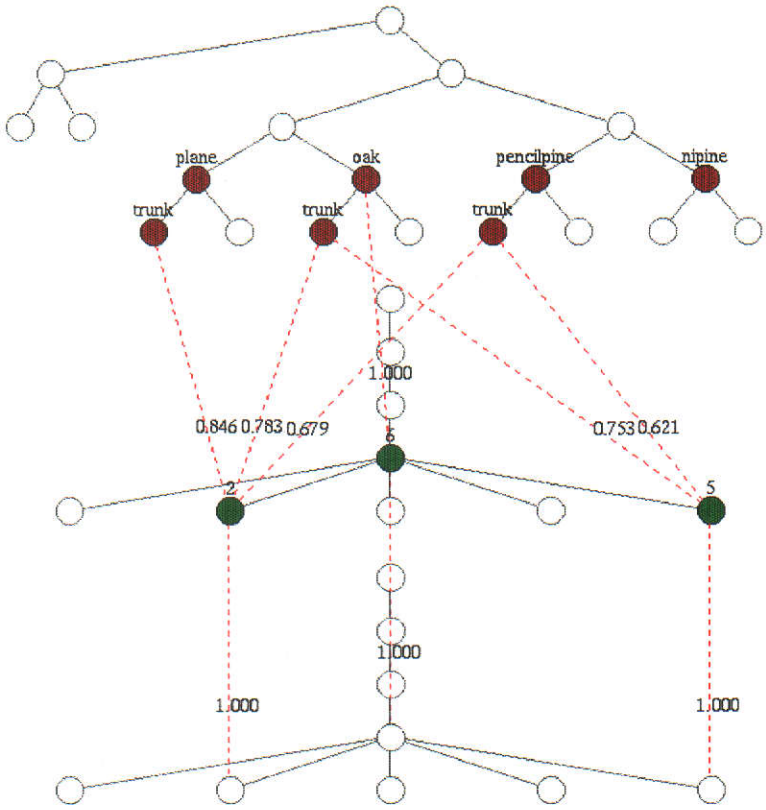


Figure 5.5: **KB, SI and VI Graphs After Initial Segmentation and Processing**

that one SI leaf node has multiple hypotheses linking it with the knowledge base. This will be resolved trivially by the hierarchical relaxation labelling process, which has not been discussed in this example. The labelled segmentation is shown in Figure 5.7. *Cite* can also produce a text description of the scene, which is detailed in Figure 5.8.

As a rough account of the performance of the system, this simple scene interpretation took 66 seconds to compute[1], most of which was spent completing the two segmentations. The knowledge base used in this example was constructed from 19 images of four types of tree and two types of shrub. The process of building the knowledge base is described in Chapter 6.

This example has shown the basic methods by which *Cite* operates. The most important aspects are the three interconnected data structures representing the knowledge base, scene interpretation and visual interpretation, and the range of operators which create

[1]This is when executed on a single 150 MHz R4400 processor

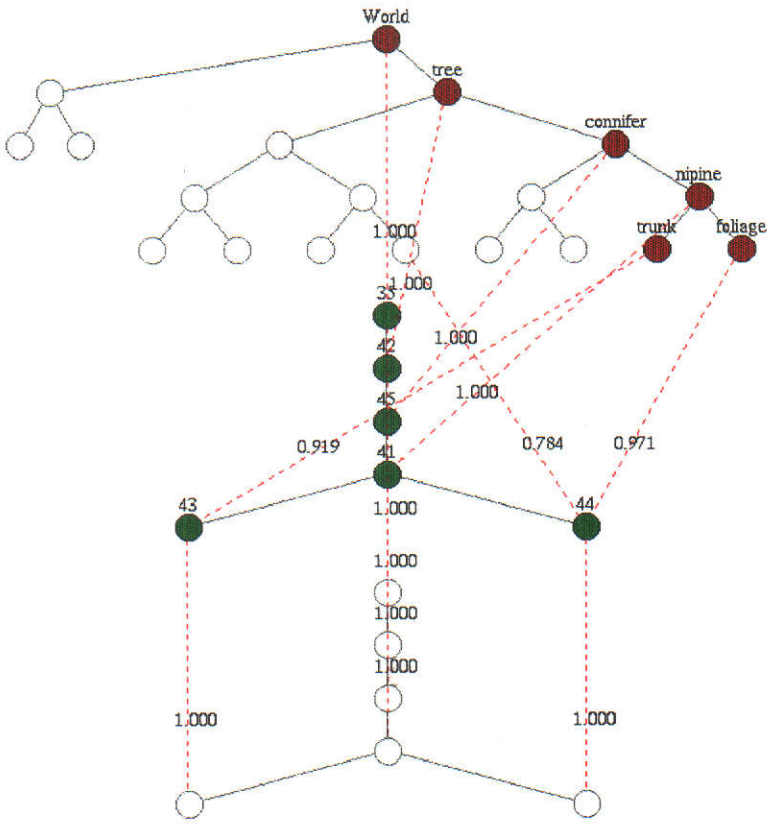


Figure 5.6: **Example Data Graphs after the Second Segmentation**

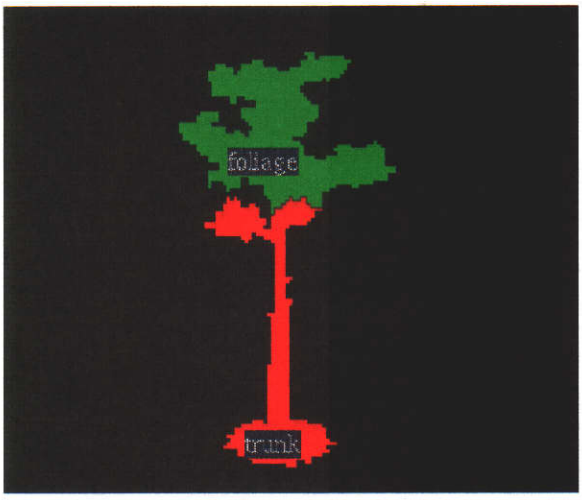


Figure 5.7: **Final Labelled Segmentation of the Example Image**

and modify these data structures. The operators behave independently and represent a range of functionality from low level segmentation and feature extraction through to hierarchical relaxation labelling, clique resolving and knowledge directed segmentation.

```
World[0] (1.000) Consisting of:
└── tree[11] (1.000) Of Type:
    └── conifer[12] (1.000) Of Type:
        └── nipine[6] (1.000) Constructed from:
            ├── trunk[9] (0.919)
            └── foliage[10] (0.971)
```

Figure 5.8: **Text Description of Example Scene**

The remainder of this chapter describes how the operators are run, and categorises and summarises each of the operator types and their function. It will be valuable for the reader to refer back to Figure 5.1 to locate each operator and examine its position within the data structures in *Cite*.

5.2 Operator Scheduling

Cite's operators are all loaded into a process stack when the system is initialised. The process scheduler cycles through this process stack and executes each operator. The process scheduler is not interrupt or timer driven but relies on each operator to return control to the scheduler after a short time interval. There is no inter-process communication other than via the status of the interpretation structures.

Each operator is responsible for determining whether there is any work for it to do. If so, it must return control to the process scheduler within a reasonable period of time (less than one second) to ensure that the user interface does not slow down. A commercially oriented implementation of *Cite* in a multi-processing environment could easily fork each operator as a separate process. The current cyclic scheduler was used for simplicity alone.

In the evolution of *Cite* other architectures and process scheduling algorithms were considered. The final system is a data structure oriented system where the data structures alone hold all process control information in an implicit form. This architecture is similar to a data flow architecture in that the process control is carried out directly according to the state of the data.

The choice of a cyclic scheduler over a control flow scheduler was made so that each operator behaves independently. This provides the system with an "agented" flavour, however this description would be inaccurate as the processes do not duplicate or remove themselves and do not contain any permanent data. Another motivation for this approach is that the data structures in *Cite* represent the current scene interpretation, and the interpretation functions decompose easily into a number of separate and independent processes.

5.3 Segmentation Operators

The segmentation operators are responsible for grouping pixels in the image into regions of common texture or colour, for the grouping of these regions into possible higher level objects, and for the knowledge driven resegmentation. The segmentation algorithms are discussed in more detail in Chapter 9, but each will be described briefly here.

5.3.1 Segmentation Execution Operator

The PROCESSSEGMENTRUN operator is the main segmentation execution operator. It scans the visual interpretation graph for nodes with incomplete segmentations loaded into their *segmentation stack*. In this instance, a segmentation comprises the choice of segmentation algorithm and the set of parameters for that instance of the segmentation algorithm.

When an incomplete segmentation is detected, this process executes one cycle of the segmentation before returning control to the process scheduler. When the segmentation process on the given VI node is complete, the PROCESSSEGMENTRUN operator removes the segmentation from the segmentation stack and places it on the *segmentation history stack* in the VI node. The segmentation history is maintained so that the knowledge driven segmentation process knows what segmentations have been tried in the past on each VI node.

5.3.2 Segmentation Initialisation Operator

The PROCESSSEGMENTINITIAL operator scans the scene interpretation graph for nodes with no child visual support. That is, if the visual support node or nodes for the SI node under consideration have no children then *Cite* knows that segmentation is required on the childless VI node. When this condition is detected *Cite* determines the best segmenter algorithm and its parameterisation to be run on the VI node. This is loaded into the *segmentation stack* in the VI node and will then be executed by the PROCESSSEGMENTRUN operator in due course. The algorithm for determining the best segmenter and its parameterisation is described in Section 9.1.1.

This operator is used to generate the initial segmentation for each image and the initial segmentation for top-down VI hypothesis generation. The PROCESSRESEGMENT operator is similar, but operates only after the initial segmentation and labelling has occurred.

5.3.3 Resegmentation Operator

The PROCESSRESEGMENT operator is similar to the PROCESSSEGMENTINITIAL operator in that it scans the scene interpretation graph looking for nodes that require segmentation. However, the resegmentation operator only operates on SI nodes whose visual interpretation nodes have had at least one segmentation already, and where multiple knowledge base hypotheses exist or, in the case of part-of nodes, the number of children does not match the number in the knowledge base.

This operator will initiate a resegmentation of the VI node in question if labelling subsequent to the earlier segmentations gives rise to a better hypothesis of what the scene element is, and there exists a more appropriate segmenter that can be run based on this subsequent labelling. This is *knowledge driven resegmentation* and is described in further detail in Chapter 9. The net result of this operator is the dismantling of the local VI sub-tree, and the loading of a new segmenter into the segmentation stack of the parent VI node.

5.3.4 Connected Components Grouping Operator

The PROCESSSEGMENTCC operator is used to group regions by connected components analysis to form objects. This is a standard bottom-up part grouping algorithm in common usage in machine vision systems. Although fast, it only works on isolated objects (objects that do not touch other objects). In *Cite* this operator is only used in early learning when the more sophisticated clique resolving operator has been deactivated.

5.3.5 Clique Resolving Operator

The PROCESSCLIQUEGROUPING operator is a more sophisticated part grouping algorithm that creates groups of parts that may belong to the same object. This algorithm typically generates a number of different groupings when there are multiple touching or overlapping objects. This operator over-rides the simple connected components operator which is only used when there are single or isolated objects in the image.

5.4 Feature Extraction Operators

The feature extraction process is currently carried out by two processes; PROCESSBASICUNARY and PROCESSBASICBINARY. These calculate the default unary and binary features and store the results in the visual interpretation structure. One important part of these operators is that they must recompute features after resegmentation takes place. This is achieved by iteration-stamping the features in the visual interpretation structure so that *Cite* knows how old the features are.

5.5 Hypothesis Generation Operators

Hypotheses in *Cite* include all components of the visual and scene interpretation structures. In some senses, the segmentation processes are hypothesis generation processes because they generate low level "hypotheses" about which pixels belong together. How-

ever, because the accepted terminology segregates these into the class of "segmentation" operators, they are described separately. The remaining hypothesis generation processes are discussed here.

5.5.1 Basic Unary Matching Operator

The PROCESSUNARYMATCH operator takes low level nodes in the visual interpretation and tries to match them with the knowledge base. This is achieved by executing the **EvaluatePoint** function of the learning/matching strategy stored in the knowledge base with the unary features of the part. The hypotheses generated link the associated scene interpretation node with the knowledge base nodes that best match the unary region properties, using a measure defined separately by each unary matching process.

5.5.2 Generating SI from VI

The PROCESSSIFROMVI operator examines the hierarchical structure of the visual interpretation graph to make sure that the hierarchical components are reflected in the scene interpretation graph. The scene interpretation graph is not necessarily identical to the visual interpretation graph but must reflect its topology. This is a bottom-up process.

5.5.3 Generating VI from SI

The top-down version of the PROCESSSIFROMVI operator is the PROCESSVIFROMSI operator which propagates scene interpretation hierarchical structure down to the visual interpretation structure. The scene interpretation structure typically comes from the knowledge base and hence is a knowledge driven top-down process.

5.5.4 Hierarchical Group Matching

The PROCESSHIERARCHYMATCHGROUPS operator takes possible groupings of parts and examines the knowledge base for objects that the parts could belong to. This process combines unary and binary information to generate the initial object hypotheses. This operator only executes when the hypothesised parent knowledge base node is a Part-Of node indicating a construction.

5.5.5 Knowledge Base Hierarchy Matching

The PROCESSHIERARCHYMATCHKB operator examines the knowledge base hierarchy and ensures that the scene interpretation structure reflects the Type-Of and View-Of knowledge. This operator effectively works hand in hand with the PROCESSHIERACHY-MATCHGROUPS operator to provide knowledge driven (top-down) hierarchical scene decomposition. The hierarchy generated in the scene interpretation is then propagated down to the visual interpretation structure by the PROCESSVIFROMSI operator.

5.6 Relaxation Labelling Operators

Cite can generate multiple labelling hypotheses, multiple grouping hypotheses and multiple parent relationships for nodes in the scene interpretation and visual interpretation. Each of these is weighted, and the weights can be interpreted as probabilities. A hierarchical form of relaxation labelling is used to update these probabilities and propagate constraints across the various graph structures.

The relaxation labelling process is a two stage process; the first stage generates new estimates for each hypothesis, and the second stage normalises these hypotheses with respect to the scene and visual interpretation structures. Because of the interaction of the hypothesis update algorithms, all hypotheses must be re-estimated before any are normalised. The estimation processes leave the hypotheses in an un-usable state, so for this reason the normalisation stage must occur immediately after the estimation stage.

Consequently, all the relaxation labelling algorithms are combined into one process called ProcessUpdateWeights which does both the estimation and then the normalisation on all hypotheses in *Cite*. This one process is quite involved algorithmically, but is fast to execute, so there is no problem integrating the various algorithms together. Each of the hypothesis update algorithms is described in detail in Chapter 8.

5.7 Ancillary Operators

In addition to the main operators described in the previous sections, *Cite* contains a number of maintenance, status and consistency checking operators. These make *Cite* more usable as well as automatically detecting problems in the interpretation structures. This latter facility is most valuable because the interpretation structures are constantly being created, dismantled and moved around.

5.7.1 Initialisation Operators

The two operators ProcessInitScene and ProcessInitVINode generate the top level SI and VI nodes respectively when a new scene or view is loaded. These processes effectively seed the more complex operators described above. ProcessInitVINode creates a new top level VI node if it detects an image without a corresponding VI node. ProcessInitScene creates a top level SI node if there is a top level VI node with no SI hypothesis and the top level SI node does not exist.

5.7.2 VI Node Checking

Two operators, ProcessVINodeCheck and ProcessVINodeCheck2, scan the visual interpretation structure and make sure that the various requirements of this structure are maintained at all times. These operators check the following VI node properties:

- Each pixel in a VI node is also a pixel of each parent VI node.
- No pixel is repeated in a VI node pixel list.

5.7.3 CPU Performance Operator

The PROCESSPERFORMANCECHECK operator computes a simple estimate of the CPU performance in comparison to a 150 MHz R4400 processor (as this is what *Cite* is normally run on). *Cite* maintains a list of execution times and rates for each operator which is useful in comparing the relative computational cost of each process. The global performance measure is useful in comparing these execution times when the program is run on other systems.

5.7.4 Profile Update Operator

The PROCESSPROFILEREFRESH operator periodically updates the operator profile window. This window contains a list of all operators, their total execution time, and a number of time weighted averages of their execution time and percentage total execution time. This status window is described in more detail in Appendix B.2.5.

Chapter 6

Learning World Knowledge

The database in any information processing system can be built in three main ways. It is possible to sit down and "hard-code" algorithms optimised for the detection of certain signals based on the experts knowledge of these signals. Secondly, knowledge can be constructed from idealised internal representations such as computer aided design (CAD) object models. Finally, knowledge can be obtained by learning from examples that are sensed in the same manner as in normal run-time operation.

Hard coding algorithms and building knowledge from idealised models is known as the *model-based* approach. One example of this is Computer Aided Design (CAD) based object recognition which is quite common in industrial inspection applications where the object models are already available. Hard coding of recognition algorithms tends to be quite difficult in object recognition because of the large amounts of data involved and the inflexibility of the resulting system, and for these reasons is not a common approach.

Learning from examples is a difficult approach yet one that is becoming increasingly popular mainly because it moves the knowledge acquisition burden from knowledge engineering to machine learning. Within the learning from examples approach, there are two broad categories; supervised and unsupervised learning. In supervised learning, the classification system is told what each object is as it is being learnt, whereas in

unsupervised learning this additional information is not known.

Cite uses the supervised learning method because the limitations imposed by the other two methods are too great. Direct hard-coding is time consuming and, in the case of computer vision, subject to a general lack of expert knowledge. One disadvantage with model based approaches is that the system requires access to complete descriptions of objects so that it can then prune the information to a minimal form for recognition purposes. Only for simple constructed objects is such information available. Less rigidly defined objects such as "tree" or "house" and higher level concepts such as "forest" or "street scene" have no known explicit model representation.

The choice of supervised over unsupervised learning is clear in the case of *Cite*, and of scene understanding and object recognition systems in general. Unsupervised learning (similar to *clustering* and *data mining*) will not necessarily learn the desired categories for objects as determined to be useful by the user. In addition, *Cite* uses *incremental* supervised learning in which the training and run-time phases are unified so that the system may continually learn as it is being used.

This chapter describes in detail the learning mechanisms in *Cite*. Section 6.1 discusses the differences between supervised and unsupervised learning in more detail, and Section 6.2 describes in general terms the refinements required of supervised learning algorithms to make them incremental. The remainder of the chapter describes the learning processes used in *Cite* and concludes with a discussion of how the unary and binary matching algorithms interact with the relaxation labelling process.

6.1 Supervised Versus Unsupervised Learning

There are two general categories of inductive machine learning; supervised and unsupervised. In supervised learning, the system is told what each example is and must work out what sets it apart from other objects of different classes, and what common characteristics it has with objects of the same class. In unsupervised learning, the system creates class categories from a large set of examples based on a model of the data

coded into the algorithm.

It is often claimed that unsupervised methods discover natural categories in training examples *with no prior knowledge*. Unfortunately this is not the case as all unsupervised learning methods contain some form of metric which measures how similar or dissimilar two data items are. Most unsupervised learning methods that operate on real valued data use some form of clustering or agglomeration to group objects of similar characteristics together. Unsupervised neural network training schemes such as adaptive resonance theory (ART and ART-II) [Carpenter and Grossberg, 1988] and self organising maps such as Kohonen maps [Kohonen, 1990] contain metrics for determining input vector differences, or differences between input vectors and prototypic output states. In all of these unsupervised learning schemes this metric represents *prior knowledge* as it is a model of the space of input vector values. There is no "discovery of natural classes", rather, these methods cluster or map feature vectors onto output classifications (or mappings) to preserve or optimise some *pre-determined* model of the input vector space.

By way of example, AUTOCLASS [Cheeseman, Kelly, Self, Stutz, Taylor and Freeman, 1988] is an unsupervised learning scheme based on a Bayesian statistics model. In the conclusion, the authors state that the approach is "free of *ad hoc* quantities, and in particular free of any measures which alter the data to suit the needs of the program". Contrary to this claim, AUTOCLASS uses an axis-aligned normal distribution model of the input feature space, which is a commonly chosen model. This is a model of the data which may need to be altered in different circumstances; for example bounded rules assume no correlation between features whereas Gaussian distributions do.

Unsupervised learning, clustering, correlation and other data analysis methods can yield very interesting and important results. As mentioned in [Cheeseman et al., 1988] the resulting clusters can provide a useful starting point for further expert analysis of the data. The main point of this discussion is to unambiguously assert that the differences between unsupervised and supervised learning are significant, and the claims of generality occasionally made in some unsupervised learning research are not always entirely appropriate.

6.2 Incremental Supervised Learning

Most supervised learning concentrates on the task of building a classification mapping from a set of representative training data. A number of common issues arise in relation to the amount of training data to use, and how representative of the full problem space the data is. Most researchers report the best classification rates obtained from both the training data and the unseen test data. This methodology has a distinct training phase and a distinct runtime phase where classification rules and parameters are frozen after the training phase has completed.

A variant on the supervised learning method merges the training and run-time phases together so that the system can continue to learn as well as being used to classify seen and unseen data. This approach is called incremental supervised learning and requires a number of extensions to simpler non-incremental approaches.

The conversion of a non-incremental learning algorithm to an incremental form is a complex and seemingly not well understood problem. In general, there are three categories of such conversions; catastrophic, useless and ideal. The category can be determined by looking at the way the memory consumption of the algorithm increases and the way the computational complexity of the algorithm increases as a function of the number of points being learnt.

If $M(n)$ is the memory usage of the incremental form, then the ratio of $R_M(n) = \frac{M(n)}{n}$ gives an indication as to how the memory usage increases with the number of learnt points, n. If R_M is equal to one, then the algorithm probably falls into the useless category as it would be just as memory efficient to store all points ever seen. If R_M is greater than one, the algorithm is catastrophic as it is using more memory than would be required to store all the points ever seen. Clearly it is ideal for the algorithm's memory use to be sub-linear in the number of points seen. This requirement is usually achieved by non-incremental machine learning algorithms, but can be overlooked in the conversion to the incremental form (for example, ID-3 and its incremental forms ID-4 [Schlimmer and Fisher, 1986] and ID-5R [Utgoff, 1989]).

If $C_{NI}(n+1)$ is the number of computation steps required to learn $(n+1)$ points from scratch in a non-incremental algorithm, and $C_I(n \rightarrow n+1)$ is the computation required for the incremental algorithm to go from the n^{th} to the $(n+1)^{th}$ point, the ratio $R_C(n) = \frac{C_I(n \rightarrow n+1)}{C_{NI}(n+1)}$ should be less than one also for an ideal algorithm. If R_C is equal to one, then it is no different to simply learning all n points from scratch, and if R_C is greater than one the algorithm is clearly catastrophic in its computational complexity.

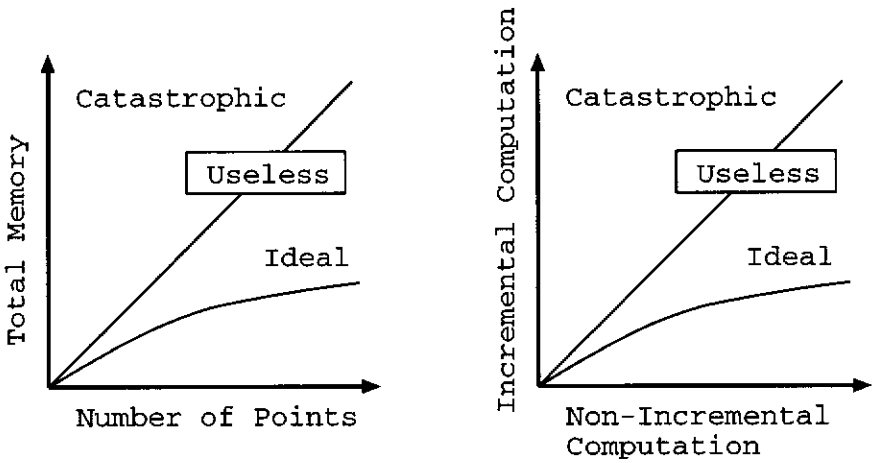


Figure 6.1: **A Categorisation Method for Incremental Learning Algorithms**

Figure 6.1 illustrates these two ratios and the categories described. Some incremental learning algorithms proposed in the literature are linear in memory or computation or both, and hence have questionable usefulness. The Explanatory Least Generalisation algorithm described in Section 6.5.3 as applied to incrementally generating decision trees is novel in that it achieves sub-linear growth in both memory and computation requirements.

Further, it appears that an incremental learning algorithm should converge to a solution equivalent (if not identical) to the non-incremental form as the number of training samples approaches infinity and given a stationary class distribution. Some of the incremental algorithms published achieve this, but invariably do so by repeatedly rebuilding their rule structures from scratch with linear computational cost in the number of training samples.

6.3 The Part-Indexing Problem

A common claim underpinning a large proportion of the supervised learning literature is that the learning problem can be stated *without loss of generality* as being the problem of finding a mapping between the input space and a classification space such that the training data is classified correctly and the system behaves with sufficient generality on unseen data. This assumption is reasonable with regard to many applications of machine learning where each feature point represents a single isolated object. However, in object recognition one typically has several parts of a given object and the generality of this approach fails when part labels (identity or indexing) must be considered.

To illustrate this problem, consider a face recognition application where two eyes, a nose and a mouth can be segmented out and described by a set of *unary* features covering the colour, size and shape of each part. The system can also compute *binary* features such as the distance and relative orientation between pairs of these parts. For four parts, there are four unary description vectors and up to twelve[1] binary description vectors. A system could conceivably contain a unary classification mapping and a binary classification mapping, but a problem exists when trying to put all of this evidence together to detect the presence of the "face" object.

Adopting a simple approach of accumulating evidence for the "face" object from the unary and binary components will produce usable results but it will not, for example, be able to distinguish the two images shown in Figure 6.2. This is because the set of unary and binary features for the parts in these two objects is identical.

An unambiguous interpretation requires not only unary and binary features to be matched, but the correct indexing of these parts. For example, a system may know that there exist two parts 5cm apart, but it must know which parts are 5cm apart in order to arrive at a correct interpretation. The worst case ambiguity arises when matching objects with identical parts and one can typically, for n parts, have $n!$ objects which could map perfectly to it without full part indexing.

[1]The maximum number of asymmetric binary relations between n parts being $n(n-1)$

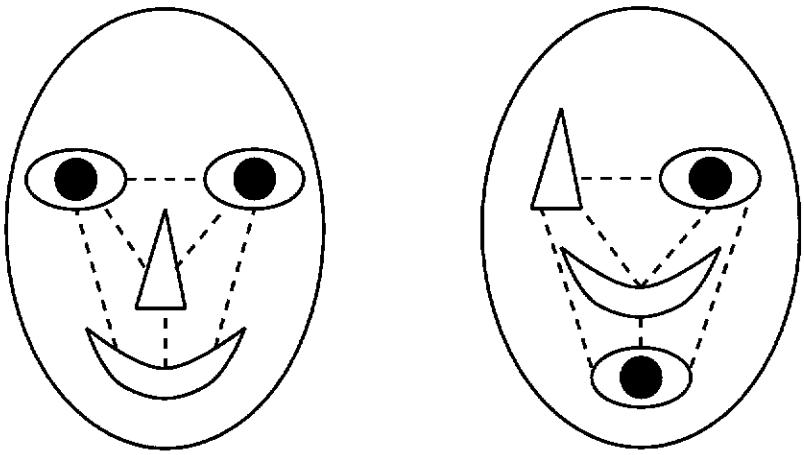


Figure 6.2: **Two Objects Indistinguishable Without Correct Indexing Between Unary and Binary Features**

A number of solutions to the indexing problem have been proposed. A simple solution is to store a graph representation of the object such that at recognition time the best possible cross-indexing of parts can be determined (usually a costly process). More sophisticated approaches have been proposed to reduce the graph sub-isomorphism complexity (such as the CLARET algorithm [Pearce, Caelli and Bischof, 1994]) or for recognising objects more as a probabilistic mapping on parts (such as CRG and its variants [Bischof and Caelli, 1994]).

In *Cite*, the binary matching algorithms may be different at each applicable node in the knowledge base and may include exhaustive graph matching and other techniques which exhibit differing degrees of solution of the part indexing problem. Relaxation labelling is always used to resolve wider scope label incompatibilities represented by the hypotheses in the scene interpretation. This can influence the way the binary matching results are propagated through the scene interpretation graph, but does not play a role in how the binary matching results are generated. This is solely done by the binary matching algorithms stored at each Part-Of parent node in the knowledge base. These binary matching algorithms are discussed in detail later in this chapter.

6.4 Overview of the Learning Strategy in *Cite*

As discussed above, single point classification strategies require some level of solution of the part indexing problem to be useful in object recognition. This problem has received some attention in the literature, as has the learning of these more sophisticated recognition strategies. For *Cite*, however, supervised learning must be extended once more to also include the learning of hierarchical knowledge. Hierarchical learning has received attention in AI literature, but because this tends to be hierarchical learning in symbolic concept formation, it is less applicable to scene understanding systems.

The hierarchical learning in *Cite* is made relatively easy by the fact that the hierarchy is defined contextually according to the manner in which the system is used. That is, the descriptions of the objects that the operator gives *Cite* during an incremental learning step contain sufficient information to build the hierarchy in a reasonably straight forward manner. Hierarchical learning in *Cite* is of less interest than the incremental learning which occurs at each node within the knowledge base, and of the way the hierarchy is used during classification.

This section describes the various processes which are used to build the knowledge base in *Cite*. Following this is a description of the supervised incremental learning algorithms used at each node within the KB for both unary and binary classification.

6.4.1 Learning Options

If *Cite* correctly classifies a new instance of an object already in the knowledge base, then, in general, it can be claimed that the knowledge base is sufficiently general to recognise the new object instance. In some cases, such as when the learning algorithm being used is based on a set of probabilistic rules, it would be appropriate to run the **LearnTrue** function on this correctly labelled object to reinforce its rules. In most other learning methods there is nothing to be gained from learning already correctly labelled objects.

When *Cite* incorrectly labels an object that is already in the knowledge base there are three possible reasons for this, each of which requires a different action:

- The incorrectly labelled object may be a completely new view of the object, and hence a new view must be added to the knowledge base.
- The matching algorithm may not be utilising the correct or available features (whether unary or binary) so these may need to be added to or deleted from the existing knowledge base.
- The object may be a different example of an existing view of an object, but the matching algorithm may have incorrect rules or parameters that require incremental retraining.

Of these three options, incremental rule modification is the least computationally and memory expensive, while adding a new view is the most expensive in these two regards. In the current implementation of *Cite* the features calculated are fixed and all are considered during learning although they may not all be considered during recognition. Deciding between learning a new view and learning a new example of a current view is determined by the user when learning is required.

6.4.2 Building a Context Sensitive Knowledge Base

In a conventional flat knowledge base built with supervised learning the labels assigned to each object can reflect context sensitivity to some extent. For example, if the user of a system only cares to classify trees as all being the same type, the description for a tree will be very general. If the user of such a system labelled trees according to their species name, then each tree description would have to be far more specific.

Cite is designed to accommodate such differing levels of context sensitivity. However, a far more valuable context sensitivity occurs in the construction of the knowledge base hierarchy according to the user's classification. Having deep knowledge hierarchies gives rise to reduced feature and rule requirements, thus improving performance with fewer

and less complex rules.

To demonstrate this, classification taxonomies of trees and bushes are built under three different contexts; that of a botanist, a forester, and a landscape gardener. The botanist will classify trees and bushes according to species, the forester will classify trees according to their straightness and size and ignore bushes, and the landscape gardener will classify the trees and bushes according to their size, foliage shape and wind coverage.

In each of the three cases, the same data is used but classified in different ways. In addition, the effectiveness of the hierarchy in reducing the knowledge base information is demonstrated by building three flat knowledge bases which represent the same taxonomies built without hierarchies.

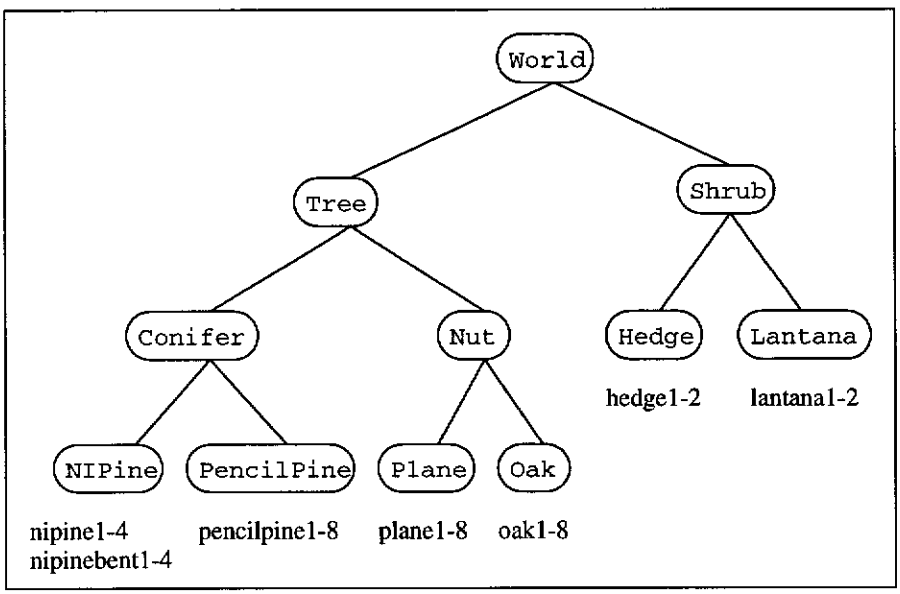


Figure 6.3: **Knowledge Base Depicting Botanist's Taxonomy**

The three taxonomies are depicted in Figure 6.3, Figure 6.5 and Figure 6.4. The full learning and recognition results for these context sensitive knowledge bases is described in Section 11.4.

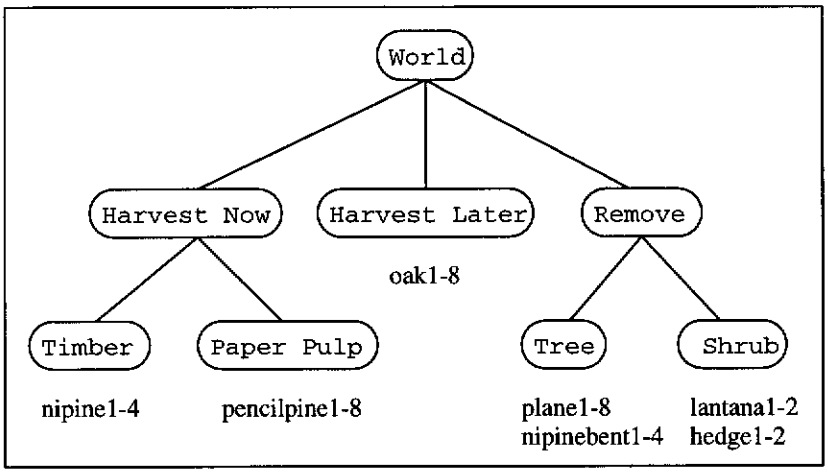


Figure 6.4: **Knowledge Base Depicting Forester's Taxonomy**

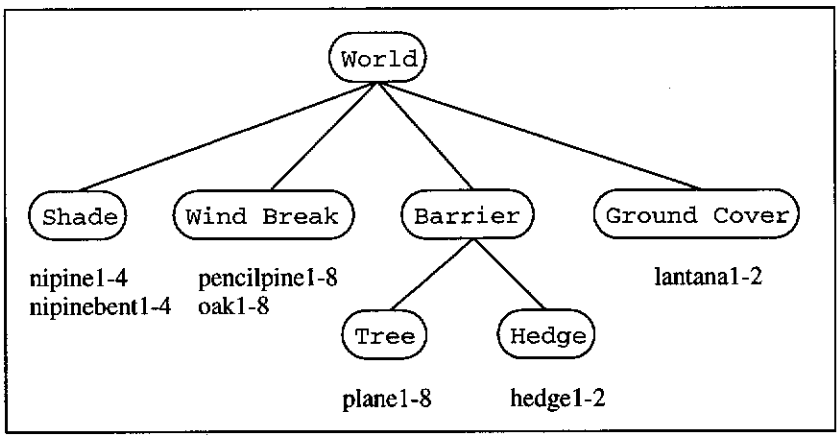


Figure 6.5: **Knowledge Base Depicting Gardener's Taxonomy**

6.5 Unary Learning Algorithms

In each node of the knowledge base, a unary matching strategy and all its required data structures can be stored. The unary matching strategy may be different at each node, and there need not be one at each node, although there does need to be at least one at every leaf node. Each matching strategy must be able to perform three operations:

1. to determine the degree of match or mismatch to a candidate part,

2. to learn a new positive example,

3. and to learn a new negative example.

In addition it is desirable that the learning strategy adhere to the two incremental learning algorithm requirements of bounded memory usage and bounded computational complexity when incrementally learning.

Cite currently contains two unary matching strategies; a bounded rule based method and a prototype nearest-neighbour based method. In their current form, neither of these algorithms conform to the bounded memory and complexity requirements of ideal incremental learning algorithms. However, this dissertation does contain a significant new method for converting non-incremental to incremental algorithms which does conform to the bounded memory and complexity requirements. This method, known as the explanatory least generalisation (ELG) method is described in detail in Section 6.5.3 where it is applied to the ID-3 minimum entropy decision tree algorithm [Quinlan, 1986]. The ELG method could be applied to the two unary learning strategies used in *Cite*, but with the modest size of the knowledge base there would be no practical performance improvement.

6.5.1 Learning Unary Bounded Rules

A common rule representation in machine learning and pattern recognition is that of the conjunction of bounded attribute rules. For example, one may say that if an object's redness is between 0.8 and 1.0, its circularity is between 0.7 and 0.9 and its size is between 0.15 and 0.2, then it is an apple. Such rules effectively form hyper-rectangular volumes in feature space, with each providing evidence for the object that is represented by the rule. Objects may have multiple overlapping rules, and some approaches have rules representing evidence for the object and other rules representing evidence against it.

Figure 6.6 illustrates a simple two dimensional feature space with multiple classes and a number of possible rules operating on this space. Such feature spaces typically have ten to twenty features. Note that if overlapping rules give evidence for different classes

of objects, then some further resolution is required.

Generalisation is achieved by having rule conjunctions covering a volume in feature space greater than the set of individual training examples. One problem with generalisation obtained using this method is that the training examples need to cover the extreme range of feature bounds unless a heuristic is used to make the bounding volumes larger than the convex volume surrounding the training samples.

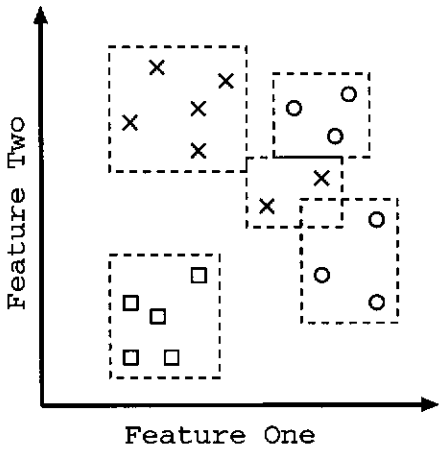


Figure 6.6: **Examples of Unary Bounded Rules in a Two Dimensional Feature Space**

In *Cite* the bounded unary rule matching strategy is coded into a matching and learning class appropriately called the UNARYBOUNDED algorithm. The UnaryBounded rules are incrementally built from example positive and negative points. Bounded rules are evaluated as a conjunction of feature upper and lower bounds. That is, test point **u** activates rule i if, for all features $j \in \{1...n\}$, $r_{i,j}^{\text{low}} \leq u_j \leq r_{i,j}^{\text{up}}$ where $r_{i,j}^{\text{low}}$ is the lower bound of rule i in feature j and $r_{i,j}^{\text{up}}$ is the upper bound of rule i in feature j.

In the UnaryBounded learning algorithm, rules are constructed according to the algorithm described in 8 as listed in Appendix A.1.2. This algorithm is pictorially described in Figure 6.7. To learn a new positive example, both positive and negative rules are considered. Each positive rule is extended to include the new point (1), and then any of these extended rules which overlap negative rules are not considered (2). If there are no extended rules remaining, a new small volume rule is constructed centered around the

new point (3). If there are multiple extended rules which do not overlap any negative rules, the rule of smallest volume increase (4) is kept and placed in the final positive rule set (5).

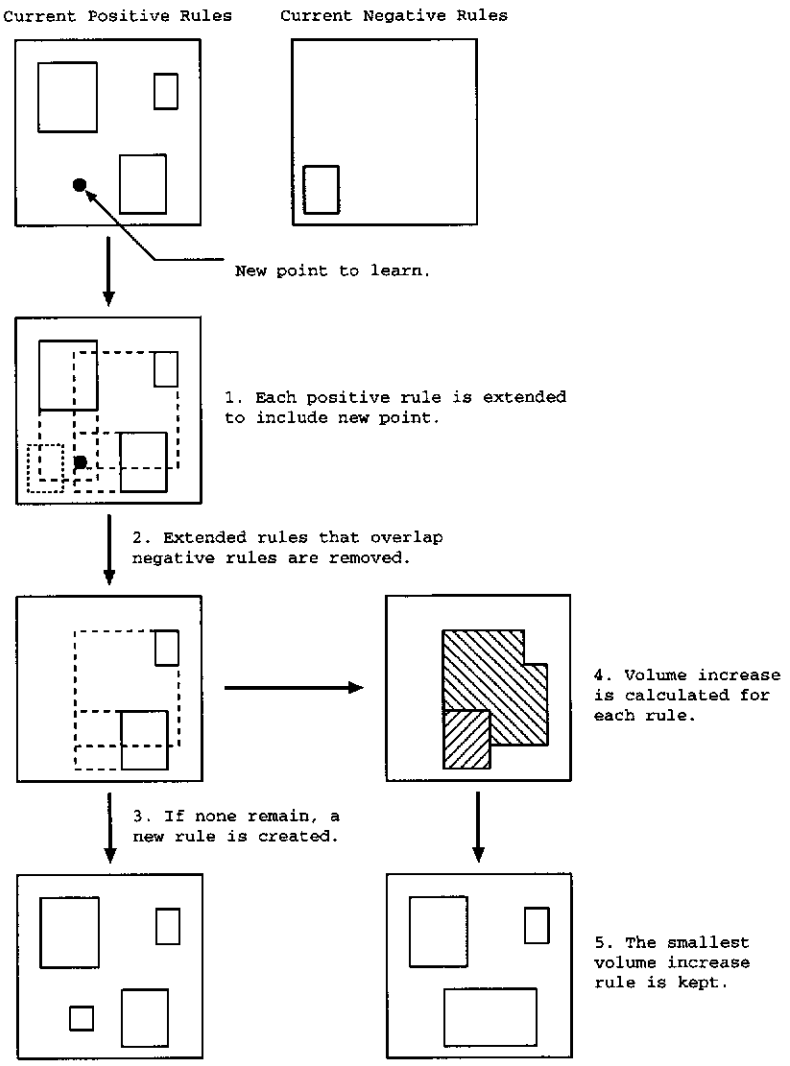


Figure 6.7: **UnaryBounded Learning Algorithm Flow Chart**

In the case of learning a new negative example, the same algorithm is applied but negative rules are extended and not permitted to overlap existing positive rules. The net result is that positive rules may overlap each other, as may negative rules, but positive and negative may not overlap each other. This algorithm keeps only existing rules from one iteration to the next, but is not as stable as the ELG algorithm applied to decision trees. However, it does provide a usable algorithm to compare performance at matching and generalisation.

6.5.2 Learning Unary Prototype Rules

An alternative to the UNARYBOUNDED algorithm is to construct prototypes of each object in the feature space, and then classify according to the test point's distance from each prototype. Depending on the distance metric used, the classification boundaries are a form of Voronoi tessellation, as shown in Figure 6.8.

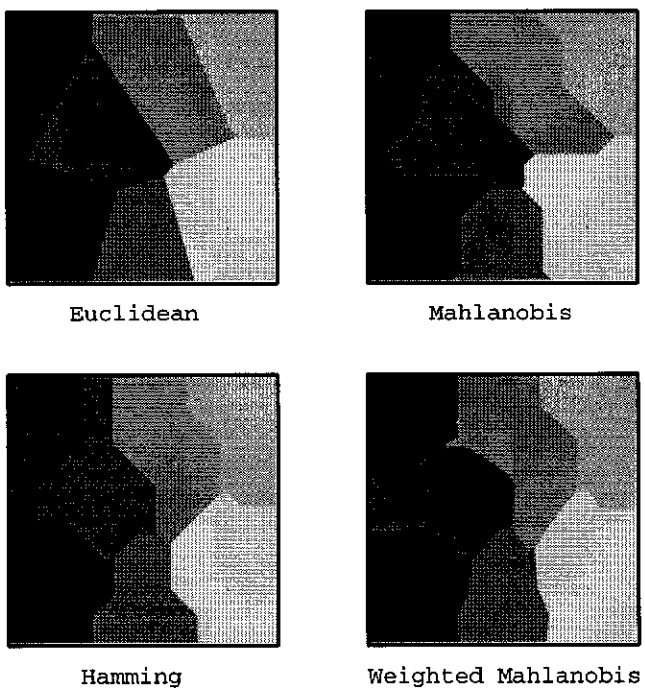


Figure 6.8: **Decision Boundaries For Various Popular Metrics**

In *Cite* this matching strategy is called the UNARYCLOSEST algorithm. This is a simple system, and the prototype points may be built by a number of methods. In a non-incremental form the prototypes can be built using clustering techniques. The incremental form proposed uses a straight forward time series averaging algorithm, although many other approaches could be used.

The incremental learning algorithm for the UNARYCLOSEST matching strategy moves the prototype points towards the training samples when correctly matched, or creates new prototype points when not correctly matched (this is similar to Kohonen Maps [Kohonen, 1990]. The Euclidean metric is used to determine proximity to the prototype

points, which end up following a trajectory towards the mean position of the local training samples, as shown in Figure 6.9. In this figure, the numbers indicate the order in which the points are learnt and the circles represent the new updated position of the prototype point.

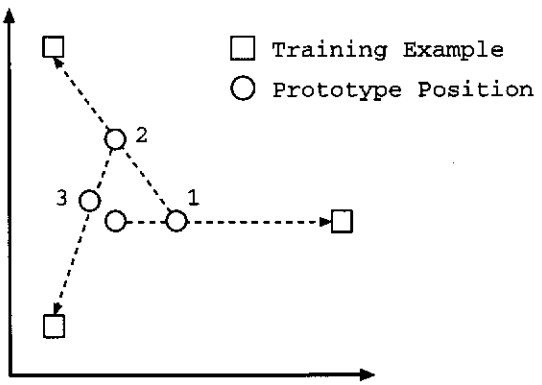


Figure 6.9: **Prototype Trajectory for One Prototype During Incremental Learning**

6.5.3 Learning by Explanatory Least Generalisation

Categorical machine learning is concerned with the classification of object instances by concepts learned from examples of these objects. The objective is to form concepts (rules) that operate on object attributes such that the system can be used to classify further unknown objects. Incremental learning is concerned with learning these rules from a serial presentation of object instances, as opposed to having a large *training set* of instances available for learning.

As discussed earlier, incremental learning algorithms should be sub-linear in memory and complexity with respect to the number of training examples, and should converge to at least an equivalent solution as the non-incremental form of the algorithm given that the presentation order to the incremental algorithm is first-order stationary. Published incremental learning theories in machine learning, such as Schlimmer and Fisher's ID-4 [Schlimmer and Fisher, 1986] and Utgoff's ID-5R [Utgoff, 1989], do not completely satisfy the above requirements. ID-4 keeps insufficient information to converge to the same solution provided by the non-incremental variant, ID-3 [Quinlan, 1986]. The ID-

5R algorithm essentially keeps the full set of instances distributed across the concept hierarchy.

The following section is concerned with a general approach to the development of incremental supervised learning methods that conform to the conditions described above, and a specific example is demonstrated in the development of a memory-bounded and convergent incremental learning system for decision trees on real-valued attributes. This general approach is achieved by storing a least generalisation of the data which gives a minimum explanation of the current concept hierarchy. This is termed the explanatory least generalisation (ELG) incremental learning method.

6.5.3.1 ELG - A Theory of Incremental Learning

At each iteration, an incremental learning system is provided with a new instance of data that may or may not be classified correctly according to the current learned concepts (Figure 6.10). If the new data is not classified correctly then the current concepts must be modified to accommodate the new data. Local concept restructuring will not always result in the optimal solution[Utgoff, 1989], so global restructuring is necessary. However, global restructuring on available instances is not only computationally intensive, but also requires storage of these instances, which does not satisfy the incremental learning requirement of bounded memory.

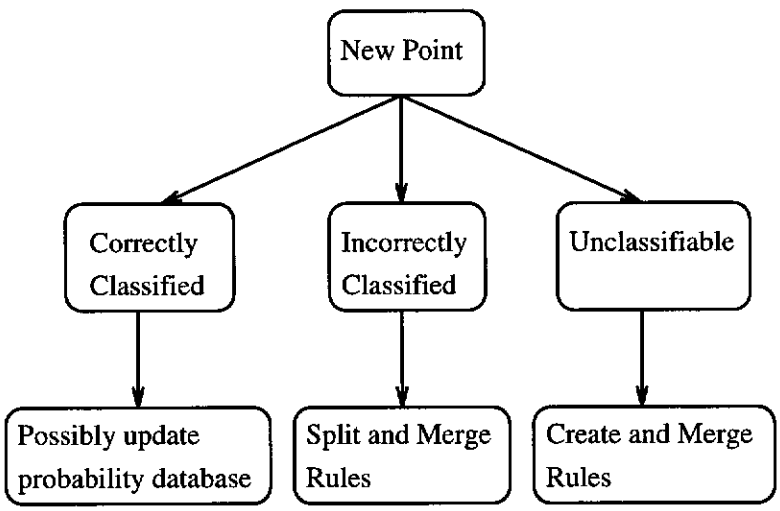


Figure 6.10: **Basic Incremental Learning Flow Chart**

The solution lies in a general approach of storing both the concept hierarchy and a *sufficient least generalisation of the instances observed to explain the current concept hierarchy.* This least generalisation is termed an *explanatory least generalisation* (ELG) of the instances with respect to the particular concept hierarchy used. The ELG can be constructed by maintaining a least generalisation of instances that map into each rule. At each iteration, the ELG is used in restructuring the set of classification rules (the knowledge base), and then these rules are used to split and merge the ELG. The result is that at the end of each iteration the ELG is valid for the current set of categorical rules. In order to formalise this theory, the following sections demonstrate the ELG approach applied to binary decision trees.

6.5.3.2 The Incremental Decision Tree Algorithm

Pattern recognition typically involves real-valued attributes (also called features), which are often visualised as a Cartesian space. Object instances are represented as points in this feature space, one-sided rules are represented as partitions, and conjunctive rules are represented as hyper-rectangles. Both partitions and hyper-rectangles are generalisations in the feature space of dimensionality equal to the number of features.

Formally the attributes for each instance are denoted as f_{i_j} for attribute j of instance i. Attributes are typically measures of size, colour, shape, or relational attributes. The binary decision tree is a hierarchy of feature partitions constructed from single feature splits, as shown in Figure 6.11. Each node in the decision tree is either a categorisation (a leaf node) or a decision node. At each decision node, one and only one feature value is one-sided tested, with the left branch being true, and the right-branch being false. This gives rise to tiled hyper-rectangular feature space partitions.

Non-incremental binary decision tree algorithms examine a large set of training data in order to arrive at the optimal set of feature partitions. The ELG method constructs and maintains the ELG alongside the binary decision tree and uses this, rather than the training data, to construct the binary decision tree.

The ELG that is maintained at each leaf node in the binary decision tree is a feature

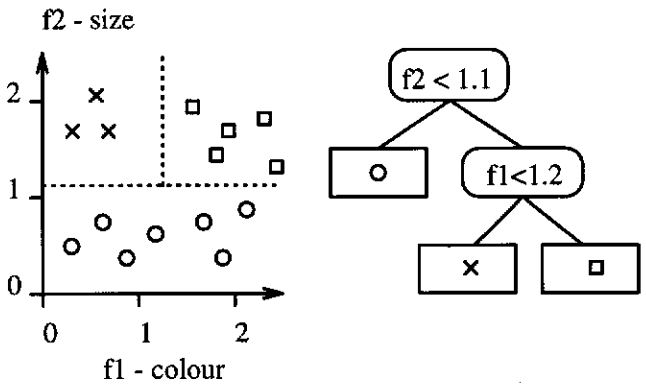


Figure 6.11: **Example of a Feature Space and the Computed Binary Decision Tree**

space hyper-rectangle that contains, in addition to the feature bounds of instances generalised by the hyper-rectangle, the time the hyper-rectangle was created and the number of instances generalised by the hyper-rectangle. This latter information is used in merging, splitting and removing hyper-rectangles.

At each application of a new, correctly labelled feature instance, the algorithm creates a single-point hyper-rectangle of no volume. This hyper-rectangle, along with the previous explanatory least generalisation hyper-rectangles are then processed to form a binary decision tree (Figure 6.12). Splitting is done to minimise the partition entropy, which is a common metric used in real-valued decision trees. Partition entropy is calculated as $E = P_0 + P_1$ where P_0 is the entropy of the ELG partitions below the nominated decision boundary and P_1 is the entropy above the nominated decision boundary. Entropy is calculated as $P = -\sum_i^N p_i ln(p_i)$ where N is the number of different categories, and p_i is the fraction of instances of class i. The number of instances that fall into each ELG partition needs to be kept in order to calculate the partition entropies. Apart from this information, however, the exact feature vector $\tilde{f}_i$ is ignored after this point. The important distinction between this and previous methods is that the data atoms are the ELGs, which are hyper-rectangles, as opposed to the full training set, which is a large set of feature points. This results in the possibility of splitting an ELG hyper-rectangle, in which case point instances are distributed uniformly amongst the split regions.

The new set of hyper-rectangles is then merged according to the feature space partition-

ing dictated by the decision tree thus created. This resultant set of hyper-rectangles is the correct ELG for the next iteration, and the new decision tree can be used for correct classification at that point in time.

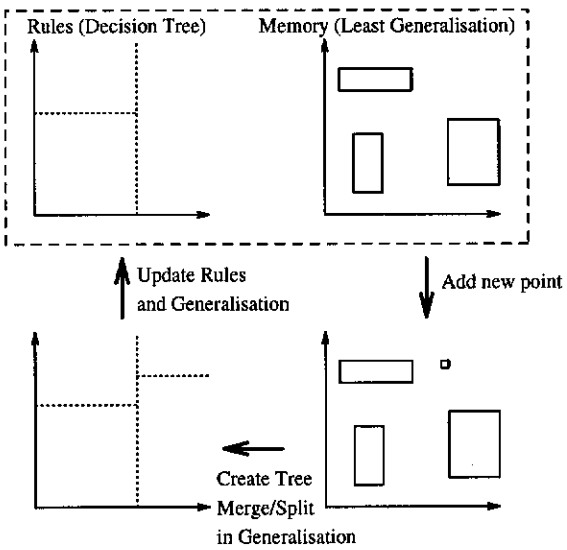


Figure 6.12: **Incremental Learning Rule and Least Generalisation Model**

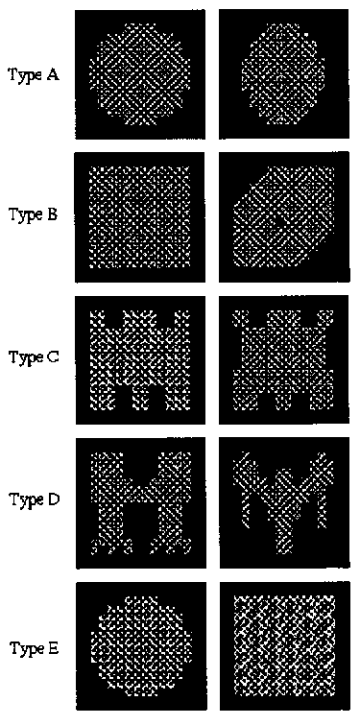


Figure 6.13: **Examples of the Five Pattern Categories**

```
PROCEDURE ADDINSTANCE:
CREATE A ONE-POINT LABELLED PARTITION.
RECURSIVELY CALL GENERATE.
MERGE.
PURGE.
PROCEDURE GENERATE:
if NO VALID POINTS then
  RETURN
end if
for all F ∈ FEATURELIST do
  SORT ELG BOUNDS ACCORDING TO THIS FEATURE.
  CONSTRUCT A LIST OF PARTITION SPLIT POSSIBILITIES.
  for all SPLIT ∈ SPLITLIST do
    MEASURE THE PARTITION ENTROPY.
    BEST SPLIT IS LOWEST ENTROPY.
  end for
end for
SPLIT RELEVANT ELGS ON BEST SPLIT.
PROCEDURE MERGE:
for all SPLIT ELG PARTITIONS do
  if PARTITION LABELS ARE IDENTICAL then
    MERGE THE ELG PARTITIONS.
  end if
end for
PROCEDURE PURGE:
for all ELG PARTITIONS do
  if ACTIVITY LESS THAN THRESHOLD then
    REMOVE PARTITION
  end if
end for
```

Algorithm 1: Algorithm for Adding New Instance to Knowledge Base

The algorithm for adding a single new instance to the knowledge base is described in Algorithm 1. The procedure **Generate** is similar to ID-3 and other binary decision tree systems. However, the method for generating potential feature splits is far more complex here because the "raw data" is a set of non-zero extents on each feature axis (pairs of upper and lower bounds from each ELG partition). Similarly, once the feature and the position of the best split is determined, it is necessary to split the ELG partitions. This can become complex as a number of special cases arise if the new data point happens to land in a current ELG partition.

Typically, complete restructuring is rare and only occurs when a new instance sets up a long chain of concept violations. The complexity of this reconstruction is bounded approximately by the number of leaf nodes in the decision tree. This compares favourably

with the heavily computational task of reconstructing the decision tree on a large number of individual training instances (as is the case in non-incremental learning). As the complexity of the ELG matches that of the decision tree and dictates the amount of data used in each reconstruction process, the amount of memory and computer time used is directly related to the complexity of the classification problem, not the amount of data used.

6.5.3.3 Example Results

The example learning task is that involving noisy samples of 5 objects depicted in Figure 6.13. For illustrative purposes, only two features were chosen; compactness and image variance. For the compactness measure we used $ln(\frac{P^2}{A})$ where P is the object perimeter and A is the object area. Image variance was calculated as the normalised intensity variance over the object. Noisy samples of the objects were chosen such that the samples covered the regions in feature space as shown in Figure 6.14.

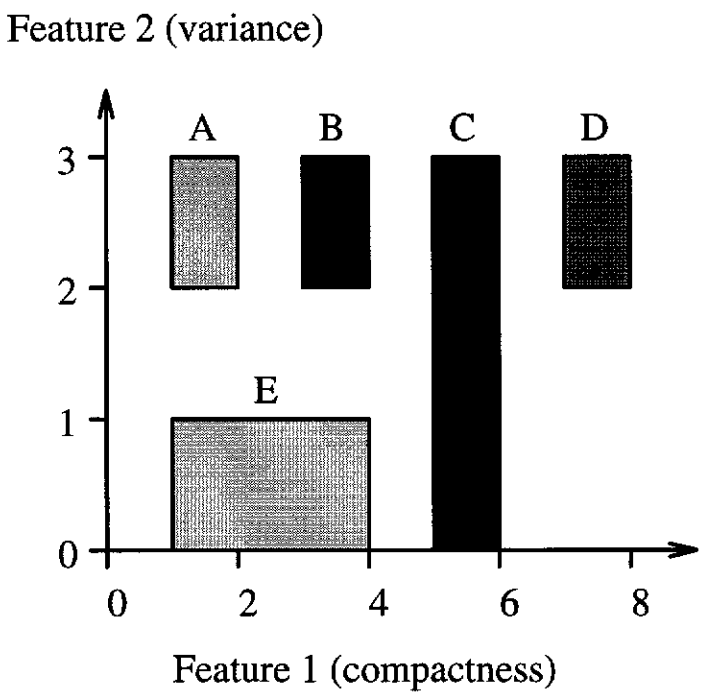


Figure 6.14: **Feature Space Distribution of the Five Pattern Categories**

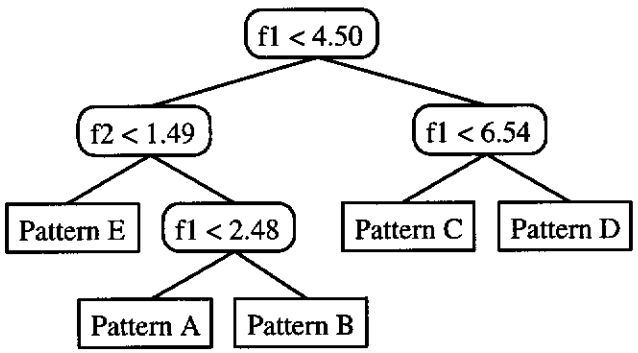


Figure 6.15: **Decision Tree Generated after 100 Iterations**

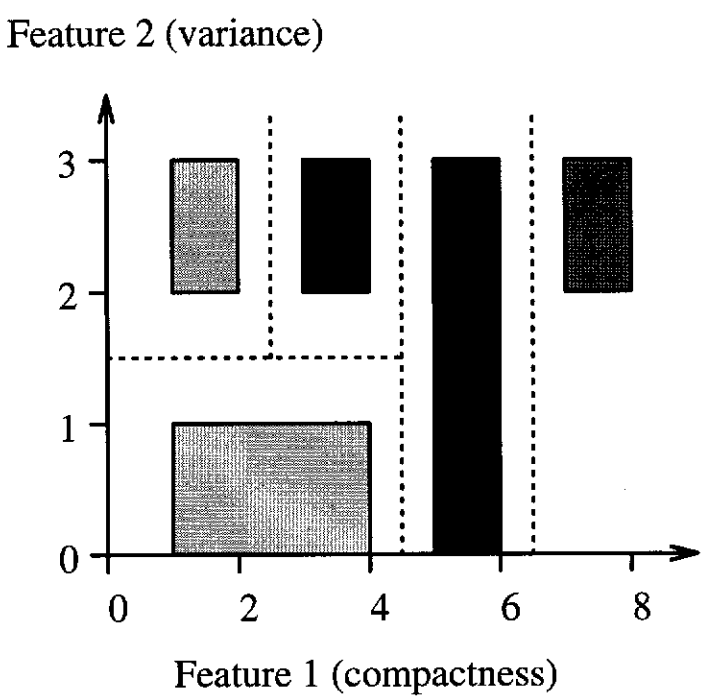


Figure 6.16: **Decision Tree Represented as Feature Space Partitions**

These samples were presented in a random order to the ELG algorithm which started from a null knowledge base. At example 28, the correct number of rules and partitions is established. However, one of these is incorrect as it is removed at example 50 by the procedure **Purge**. Correct rules are established at example 58. From example 58 onwards the ELG partition bounds are gradually improved through merging on new data, and no more splitting is required. Table 6.1 illustrates the convergence of the ELG partitions to the correct feature distributions, with snapshots at 100 and 200 examples.

Pattern	Min f_1	Max f_1	Min f_2	Max f_2
A (Actual)	1.0	2.0	2.0	3.0
A (n=100)	1.161	1.842	2.123	2.938
A (n=200)	1.155	1.944	2.013	2.971
B (Actual)	3.0	4.0	2.0	3.0
B (n=100)	3.041	3.967	2.014	2.924
B (n=200)	3.030	3.995	2.000	2.991
C (Actual)	5.0	6.0	0.0	3.0
C (n=100)	5.041	5.997	0.004	2.972
C (n=200)	5.008	5.997	0.004	2.972
D (Actual)	7.0	8.0	2.0	3.0
D (n=100)	7.084	7.749	2.075	2.924
D (n=100)	7.083	7.879	2.008	2.936
E (Actual)	1.0	4.0	0.0	1.0
E (n=100)	1.153	3.841	0.060	0.989
E (n=200)	1.053	3.841	0.015	0.989

Table 6.1: **Convergence of the ELG Partitions**

The binary decision tree used in the classification phases converges to the non-incremental decision tree at the same rate because the binary decision tree is determined from the ELG partitions. The final decision tree is described as a tree in Figure 6.15 and as a set of feature space partitions in Figure 6.16.

6.6 Binary Learning Algorithms

Binary matching in *Cite* occurs when the operator HIERARCHYMATCHKB detects an unmatched parent with multiple children each of which has at least one initial label hypothesis. Binary matching can also occur as a result of the clique resolving algorithm generating possible groupings of parts. The binary matching algorithm is run similarly to the unary matching algorithm except that the test point is a binary feature table rather than a single unary feature point.

Cite contains two binary matching algorithms, BINARYBIPARTITE and BINARYCLOSEST. The choice of binary matching algorithm used can be different for each node in the knowledge base.

6.6.1 Learning Binary Closest Matching

The BINARYCLOSEST matching algorithm is a simple high-speed binary matching algorithm that relies on the correct ordering of the unary parts from the unary matching process. The degree of match is computed as the Euclidean proximity to the closest prototype point, which is very similar to the UNARYCLOSEST algorithm.

If we let $\tilde{f}_{i,j}$ be the binary feature vector describing the relationship between parts i and j, and $\tilde{M}^k_{i,j}$ be the binary feature vector describing the relationship between model parts i and j in prototype k, then the degree of match is defined as the following:

$$d = \min_{k \in 1..N} \sum_{i,j} \| \tilde{f}_{i,j} - \tilde{M}^k_{i,j} \| \tag{6.1}$$

$$\text{match} = 1 - \frac{d}{N} \tag{6.2}$$

Updating of the prototype models M is done in the same manner as in the UNARYCLOSEST algorithm. When learning a new positive example, the closest prototype is time-series updated if it is within a pre-set threshold distance of the closest prototype point,

otherwise a new prototype point is added to the model set.

The BINARYCLOSEST matching algorithm does not fully address the part indexing problem. For many objects the parts are correctly indexed by combining the unary matching outcome with relaxation labelling, then the binary matching determines which is the best "group" description of the parts. For this, the BINARYCLOSEST algorithm works perfectly well, especially in conjunction with the clique resolving process. However, for objects that have multiple similar parts with different spatial relations to other parts in the object, the part indexing problem needs to be taken more seriously. To cover this range of objects the BINARYBIPARTITE algorithm, described in the next section, is also included in *Cite.*

6.6.2 Learning Binary Bipartite Matching

Complete solutions to the graph isomorphism problem are generally computationally exhaustive [2] or memory intensive with large pre-compilation complexity [Messmer and Bunke, 1995]. Algorithms providing different degrees of solution exist, ranging from the naive complete solution (which is O(n!) in the number of parts) to simple methods that effectively ignore the part indexing problem and accumulate evidence based on occurrence of features (such as in EBS [Caelli and Drier, 1994] or BINARYCLOSEST above). For different applications, different degrees of solution are required to provide satisfactory answers and there is usually a trade-off between speed and uniqueness.

Bipartite matching is an intermediate solution to the graph matching problem which, when implemented directly, incorporates unary information but not binary or *relational* information other than the injective mapping between the data and model graphs. Figure 6.17 illustrates this direct implementation of the bipartite matching scheme. Such a scheme has been used in object recognition, usually in conjunction with other methods to incorporate relational information[Kak and Kim, 1991]. Bipartite matching has been shown to be polynomial in the number of parts [Hopcraft and Karp, 1973].

[2] The best algorithm for the worst case scenario is known to be $O(2^{\frac{n}{3}})$ in the number of parts [Tarjan and Trojanowski, 1977]

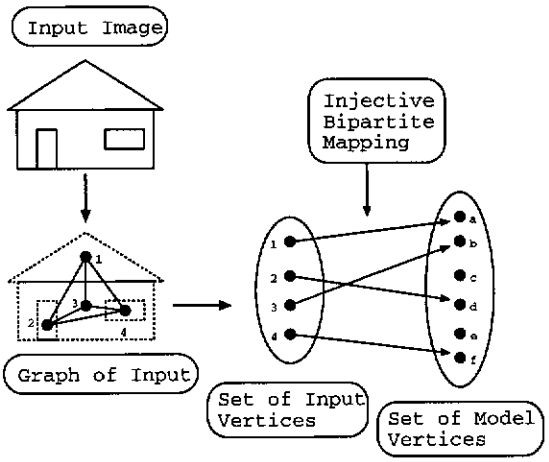


Figure 6.17: **Direct Implementation of Bipartite Matching**

Bipartite matching can be easily extended to include some relational information by matching pairs of data nodes to pairs of model nodes. This is achieved by creating the $n(n-1)$ pairs of data nodes and $m(m-1)$ pairs of model nodes and then applying bipartite matching to these. Figure 6.18 illustrates this process using the same input graph as in Figure 6.17. Note that in extended bipartite matching each matching node now represents two unary nodes (vertices) and the binary relations between them (the edge).

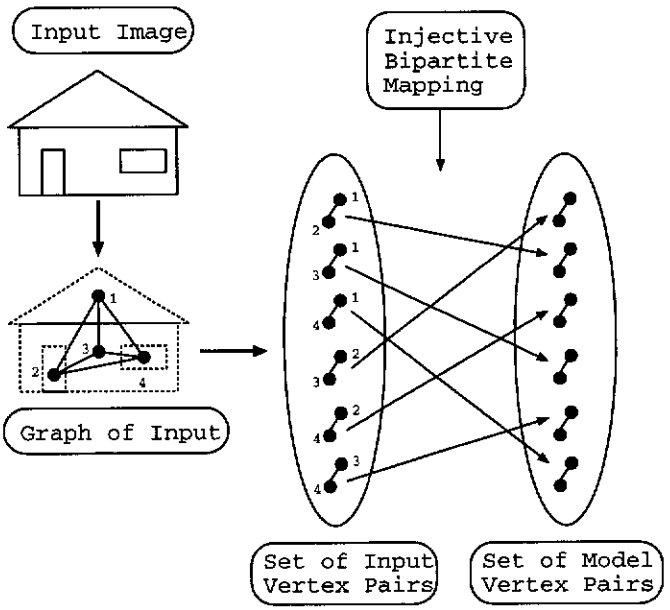


Figure 6.18: **Bipartite Matching Extended to Include Partial Relational Information**

Graph bipartite matching of this form using partial relational information does not guarantee a unique solution and in some cases can provide "non-physical" solutions. For example, the node corresponding to the data pair (1,2) may map to the model pair (5,6), and the data pair (1,3) may map to the model pair (7,8). The mapping D(1,2) $\rightarrow$ M(5,6) implies that data vertex (1) maps to one of model vertices (5) or (6) and that data vertex (2) maps to the other. However, the second mapping D(1,3) $\rightarrow$ M(7,8) implies a contradictory possible mapping. The reason for this is that indices are not used in the matching process. If they were, the algorithm would be fully solving the graph matching problem, which is computationally exponential.

The *measure* of match for bipartite graph matching can be obtained using a number of methods. The implementation in *Cite* sums the total error based on a Euclidean distance metric and the best match is declared as that which has the smallest error. A discrete form of this is presented in [Kak and Kim, 1991] in which matches occur based on proximity subject to a global noise threshold. In general, the *actual* solution to the bipartite matching is not important; the algorithm is used simply to determine whether a solution exists or not. In *Cite* the hypothesis is made that the solution exists (based on *unary* matching) and the *degree* of best match as determined by the bipartite matching is used to initialise the relaxation labelling process. This interaction is discussed in detail in the next section.

The number of nodes to be matched in extended bipartite matching is $n(n-1)$, however this still yields a polynomial solution for the most efficient algorithm. The value of polynomial over exponential solutions to graph matching problems is largely academic when applied to general machine vision systems. The number of parts that need to be matched is typically small and the main problem is robust segmentation of the parts, not the few milliseconds required to match the graphs. In more sophisticated applications of graph matching the complexity of the algorithm suddenly becomes quite important as the number of parts exceeds about twenty.

6.7 Interaction of Unary and Binary Matching with Relaxation Labelling

Cite's recognition process may be considered to be operating at a number of levels. At the "local" level, recognition is distributed between four operator classes which cover unary and binary matching, group hypothesis generation and relaxation labelling. These are described as the local recognition operators because they concentrate on the local structure of the scene interpretation (SI) graph. At a wider scale, other operators such as the knowledge driven resegmentation and hierarchical matching also contribute importantly to the recognition outcome. However, their interactions are far more quantised than the local processes.

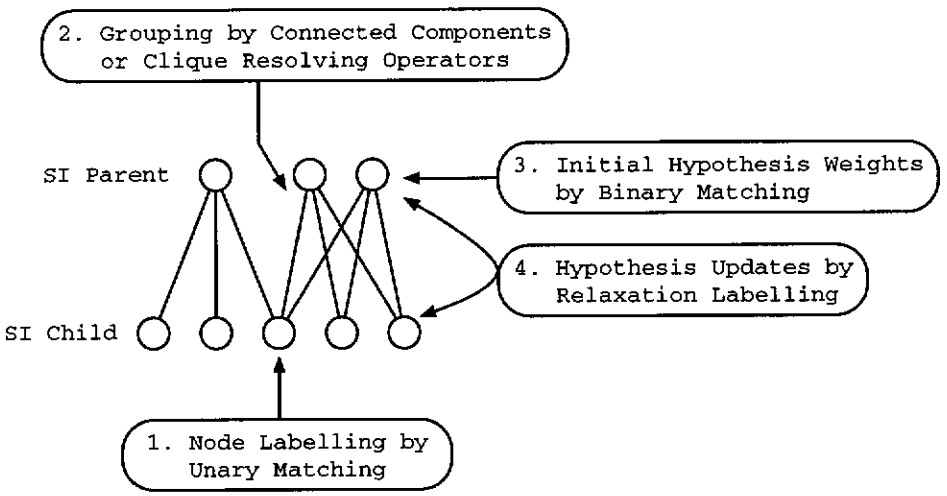


Figure 6.19: **Interaction of Unary and Binary Matching, Relaxation Labelling and Hypothesis Generation**

Figure 6.19 illustrates the local recognition processes on a small section of a scene interpretation graph. The numbers in each of the boxes gives an approximate temporal order of events. Before the parent SI nodes exist, the child nodes are generated indirectly from the segmentation process. These are then matched to the knowledge base using the unary matching strategy (1) present at each applicable KB node. From this, the connected components or clique resolving operator groups the child nodes into multiple possible parent groupings (2). This again is an indirect process because grouping

actually occurs in the hierarchical segmentation (VI graph) and is propagated to the SI by another operator. When these possible groupings have been established, the binary matching operator determines the match to each parent using the matching strategy at each applicable node in the knowledge base (3). These degrees of matching combined with unary matching and support from related sections of the SI graph then serve to initialise the hierarchical relaxation labelling process (4).

Although the initial order of these operations for a single object will be as described, for multiple objects the process becomes more temporally complicated. Certain objects will be classified faster than others, and the resegmentation process will restart the entire local recognition process for different sub-parts of the scene interpretation graph at different times. This on-going change means that the relaxation labelling process must be able to cope with the creation and removal of unary and binary matching strengths at each point in the scene interpretation graph at different times.

Chapter 7

Hypothesis Generation

Cite generates classification labels and grouping hypotheses in the visual interpretation and scene interpretation structures based on segmentation results, unary and binary feature matching and examination of knowledge base data structures. Hypotheses exist at many different levels within *Cite* and are resolved by a process of hierarchical relaxation labelling and constraint propagation. By extending traditional relaxation labelling to operate over a hierarchy, top-down and bottom-up recognition processes are seamlessly integrated. Through interaction with other operators the results of the relaxation labelling can then result in resegmentation of image regions and the prediction of missing parts or objects.

This approach departs significantly from the processes used in most object recognition systems where there is little or no feedback between the various stages during recognition. This chapter describes the process of actually generating the hypotheses. Chapter 8 continues with a presentation of the hierarchical relaxation labelling processes which dynamically update these initial hypotheses.

One innovation in *Cite* is that of considering all components of the data structure, including the hierarchical segmentation structures, as hypotheses. This view enables an elegant and general application of relaxation labelling as the constraint propagation process.

7.1 Hypothesis Types and Storage

Every aspect of the visual interpretation and scene interpretation graphs is considered as an *hypothesis* within *Cite*. The existence of a VI node or a SI node is a *unary hypothesis*, and the links between these nodes and through to the knowledge base are called *binary hypotheses*[1]. Each hypothesis contains a weight which represents the probability of that hypothesis being true.

In the visual interpretation graph, a VI node represents the probability that a certain group of pixels can be described as a single object or scene element. The VI-VI links represent the child VI node as being a valid partial decomposition of the parent VI node.

In the scene interpretation graph, a SI node represents the probability that an instance of an object exists in the scene. The SI-SI links represent the child SI node as being a valid component, view or type of the parent SI node. The VI-SI links represent the probability of the VI node providing image support for the SI node, and the SI-KB links represent the probability of labelling the SI node as an instance of the KB node in question.

Cite's data structures essentially form a decomposition web and analysis at multiple levels of abstraction which range from image pixels through to knowledge base nodes. This unifying concept allows common treatment of hypotheses and permits powerful relaxation operations because all aspects of the data structures are weighted and can thus be modified in a similar manner.

7.1.1 Hypothesis Notation

Table 7.1 describes each of the hypothesis types and where they are found in the graph structures. For completeness, we could also include P_i^K and $P_{i,j}^K$ and define them as the probability of KB node i existing and the probability of KB node i being a child of KB

[1]Note that unary and binary hypotheses are unrelated to unary and binary *features* or *matching*.

Name	Location	Description
P_i^V	VI Node	Probability that VI node i exists. Interpret this as the probability that VI node i is a correct segmentation of the pixels it contains.
$P_{i,j}^V$	VI-VI Edge	Probability that VI node i is a child of VI node j, where both are in the same image.
$P_{i,j}^{VS}$	VI-SI Edge	Probability that VI node i is the segment of the image corresponding to SI node j.
P_i^S	SI Node	Probability that SI node i exists.
$P_{i,j}^S$	SI-SI Edge	Probability that SI node i is a child of SI node j.
$P_{i,j}^{SK}$	SI-KB Edge	Probability that SI node i is an object matching KB node j.

Table 7.1: **Notation of Hypothesis Types**

node j respectively. However, these values would always be 1.0, and the probability normalisation would not be applicable because the KB graph can remain as a graph with each node having multiple parents.

Figure 7.1 illustrates the different hypothesis types and where they appear in the graph structures. Starting with the visual interpretation graph, which represents the hierarchical segmentation, we have P_i^V being the probability that VI node i exists, and $P_{i,a}^V$ being the probability that VI node i is a child of VI node a. Linking this with the scene interpretation graph, $P_{i,j}^{VS}$ is the probability that SI node j is represented by VI node i in the image. Inside the scene interpretation graph, P_j^S is the probability that SI node j exists, and $P_{b,j}^S$ is the probability that SI node j is the parent of SI node b. Making the final connection between the scene interpretation and the knowledge base, $P_{j,k}^{SK}$ is the probability that SI node j is an instance of object k. Note that this figure is for notation purposes only, and typical full graphs contain multiple parent connections and multiple labelling hypotheses for each node. Examples of these graphs are given later when the hypothesis relaxation processes are discussed.

In order to simplify the hypothesis update expressions, it is necessary to define a notation for the child and parent lists in each of the three graph structures. Table 7.2 contains a description of these index sets. The list of all parents of a SI node, for example, is S_i^P.

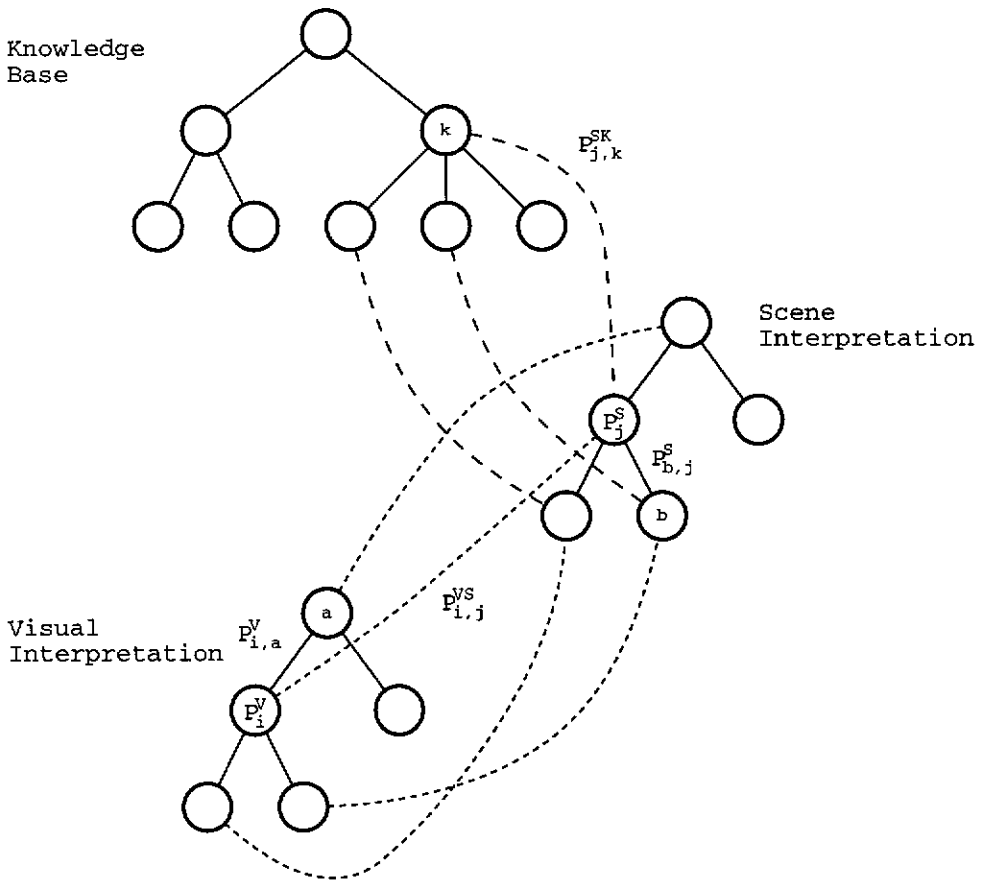


Figure 7.1: **Hypothesis Notation in** *Cite*

Name	Location	Description
V_i^C	VI	Indices of children of VI node i.
V_i^P	VI	Indices of parents of VI node i.
V_i^S	VI	Indices of SI nodes with hypothesis links from VI node i.
S_i^C	SI	Indices of children of SI node i.
S_i^P	SI	Indices of parents of SI node i.
S_i^K	SI	Indices of KB nodes with hypothesis links from SI node i.
K_i^C	KB	Indices of children of KB node i.
K_i^P	KB	Indices of parents of KB node i.

Table 7.2: **Notation of Node Index Sets**

7.1.2 Hypothesis Storage

Unary hypotheses, dealing with the existence of VI and SI nodes are stored simply by their creation and addition to existing data structures. Each of these nodes contains a hypothesis weight which can be interpreted as being the probability of the node being a correct grouping of pixels (leaf VI nodes), a correct grouping of segments (hierarchical VI nodes) or a correct scene hypothesis (SI nodes).

Binary hypothesis weights, which cover VI-VI, SI-SI, VI-SI and SI-KB hypotheses, are stored alongside the reference to the link destination node. The VI-VI, SI-SI and VI-SI binary hypotheses are stored both at the source and destination nodes. The weights on these hypotheses are stored only at one of the nodes to avoid duplication. The weights themselves are stored as a float (32 bit real value) in the range [0.0,1.0]. In addition, there is a *temporary* weight storage used by the first phase of the relaxation labelling process, and a weight history stack which records the epoch (iteration number) and value of the weight whenever it is changed. The weight history is only used to graph the weight for analysis purposes.

Figure 7.2 illustrates the various components of the weight storage. Every hypothesis in *Cite* is stored in this manner, including all parent lists, child lists, KB-SI and VI-SI hypotheses and VI and SI node hypotheses.

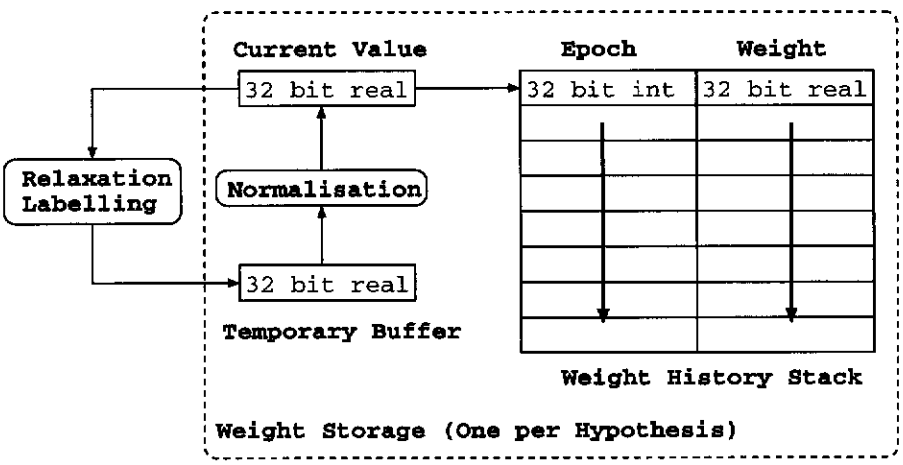


Figure 7.2: **Storage of Hypothesis Weights**

7.2 Hypothesis Generation Procedures

In *Cite*, a hypothesis can represent a scene element labelling (SI-KB), a grouping of pixels (VI), hierarchical groupings of parts (VI-VI), existence of objects in the scene (SI), visual support for these scene objects (VI-SI), and hierarchical groupings of scene objects (SI-SI).

There are two basic types of hypothesis generation procedure. The generation of a SI or a VI node is achieved by *unary* hypothesis procedures, and the generation of VI-VI, VI-SI, SI-SI and SI-KB hypotheses is performed by *binary* hypothesis procedures. Note that there is no connection between unary and binary hypothesis generation procedures and unary and binary feature related procedures.

There will be some situations where multiple binary hypotheses can be generated as a result of a single unary hypothesis. For example, in clique resolving by connected components a parent VI node is generated with binary hypotheses to the image regions that are its children. This can only be sensibly achieved in a single procedure, which can generate both unary and binary hypotheses.

In general, VI-VI and SI-SI hypotheses are simple and are handled by the unary VI and SI generation procedures. The more complex VI-SI and SI-KB binary hypotheses are generated by separate procedures.

7.3 Unary Hypothesis Generation

This section describes unary hypothesis generation procedures which construct the VI and SI nodes and can build VI-VI and SI-SI hypothesis links as required. These processes are listed in Table 7.3.

Generates	From	Operator	Description
VI nodes	Image	SEGMENTINITIAL SEGMENTRUN RESEGMENT	Image segmentation and knowledge driven resegmentation
VI nodes	VI graph	SEGMENTCC	Connected components segmenter
VI nodes	SI graph	VIFROMSI	Top-down hierarchy building
SI nodes	VI graph	SIFROMVI	Bottom-up hierarchy building
SI nodes	KB graph	SIFROMKB	Top-down hierarchy building

Table 7.3: **Unary Hypothesis Generation Algorithms**

7.3.1 Generating VI Nodes from Image Data

The process of image segmentation is usually a low-level bottom-up procedure which is executed as the first stage in most object recognition and scene understanding systems. Segmentation is invariably fragile and requires significant parameter adjustment to provide even vaguely useful results. *Cite* improves significantly on the standard bottom-up segmentation process by also including top-down knowledge driven segmentation. That is, multiple resegmentations of regions within the image can occur as *Cite* builds up evidence for the current scene interpretation.

The knowledge driven segmentation process and specific details of the segmentation algorithms are described in detail in Chapter 9. At this point, it suffices to describe the segmentation processes as producing VI leaf nodes, each of which represents one region in the image when viewed at the lowest level of detail. VI nodes themselves are generated by the PROCESSSEGMENTRUN operator, which executes the segmentation processes loaded into the visual interpretation structures by the PROCESSSEGMENTINITIAL operator and the PROCESSRESEGMENT operator.

VI nodes generated by the first segmentation will always end up as leaf nodes added under the top level VI node. Nodes generated by subsequent resegmentations will be attached as leaf nodes at the appropriate point in the hierarchy, which is typically under the parent VI node from which the resegmentation occurred. There will be occasions when nodes generated from resegmentations will be placed higher in the hierarchy, but this is described later in Section 9.1.1.

Visual interpretation nodes in *Cite* are considered as unary hypotheses which can also be interpreted as being the probability that the group of pixels represented by each VI node corresponds to some scene element as expressed in the knowledge base. Considering the VI nodes as unary interpretation hypotheses makes the application of a more general form of relaxation labelling possible.

7.3.2 Generating VI Nodes from The VI Graph

Intermediate parent VI nodes can be generated directly from the image data and visual interpretation structures when the connected components operator, PROCESSGROUPINGCC, is activated. Connected components grouping is only used during early training when the system is shown single objects and has not yet learned enough to reliably run the clique resolving process. The decision to switch from the connected components grouper to the clique resolving grouper is controlled from the user interface.

The algorithm for the connected components grouping is straight forward and moderately efficient[2]. A connectivity matrix $\mathcal{C}$ is generated such that $\mathcal{C}(i,j)$ is the number of boundary pixels region i shares with region j. This is computed using Algorithm 2. A label array is then allocated, and unique labels are propagated using a recursive paint fill operation on the label array, constrained by the connectivity matrix. Identically labelled regions are then grouped and a parent VI node is created and added to the VI graph.

```
NUMREGIONS
C = ARRAY [0..NUMREGIONS-1][0..NUMREGIONS-1]
for Y = 1 TO HEIGHT-1 do
  for X = 1 TO WIDTH-1 do
    C[IMAGE[X][Y]][IMAGE[X-1][Y]]++
    C[IMAGE[X][Y]][IMAGE[X][Y-1]]++
  end for
end for
```

Algorithm 2: Connectivity Matrix Algorithm

[2]A full discussion of linear time connected components labelling is given in [Dillencourt, Samet and Tamminen, 1992]

This two stage process of building the connectivity matrix from the image and then performing a paint fill on the label array is much more efficient than simply performing the paint fill operation directly on the image. Paint fill algorithms are not efficient, but reducing the number of data points from order 100,000 (number of pixels) down to order 10 (number of regions) removes any problems of computational speed.

The concept of forming objects by simple connected components analysis is common in object recognition systems. It fails when there are multiple overlapping objects and in this case must be replaced by a more sophisticated algorithm. The connected components grouping algorithm is essentially a boot-strap algorithm that is deactivated after the first few objects have been learnt.

7.3.3 Generating VI Nodes from the SI Graph

There are two processes which build VI nodes by examining the SI graph structure. The first is the resegmentation process which will take a VI node and its leaf node children and apply a new segmentation process to this based on the SI node knowledge base hypothesis. The second is the top-down process of ensuring that higher level labelling structures (such as taxonomic expansions) are reflected in the visual interpretation.

Both of these processes are top-down, one representing knowledge driven resegmentation, the other knowledge driven image analysis. The resegmentation process has already been discussed in Section 7.3.2, and the algorithm for initiating resegmentation is described in Chapter 9.

The PROCESSVIFROMSI operator is responsible for propagating hierarchical scene interpretation information down to the visual interpretation structure. This algorithm is described in Figure 7.3 which executes recursively on the visual interpretation structure, and is listed in Algorithm 9 in Appendix A.1.3. The basic principle of this algorithm is to scan the visual interpretation graph looking for nodes and their children whose SI nodes have an extra layer or layers of nodes between them. This implies that there is hierarchical structure in the scene interpretation which needs to be propagated down

to the visual interpretation. Once such a node has been found, an appropriate SI node from the middle SI layers is located. The final stage is that an intermediate VI node is inserted and linked to the intermediate SI node. The algorithm returns from deep within the nested loops because at the point where the modification is made the VI graph becomes different from that used by the system during the recursion process and the algorithm must terminate after each modification. This property of the operators is discussed in more detail in Appendix A.2.

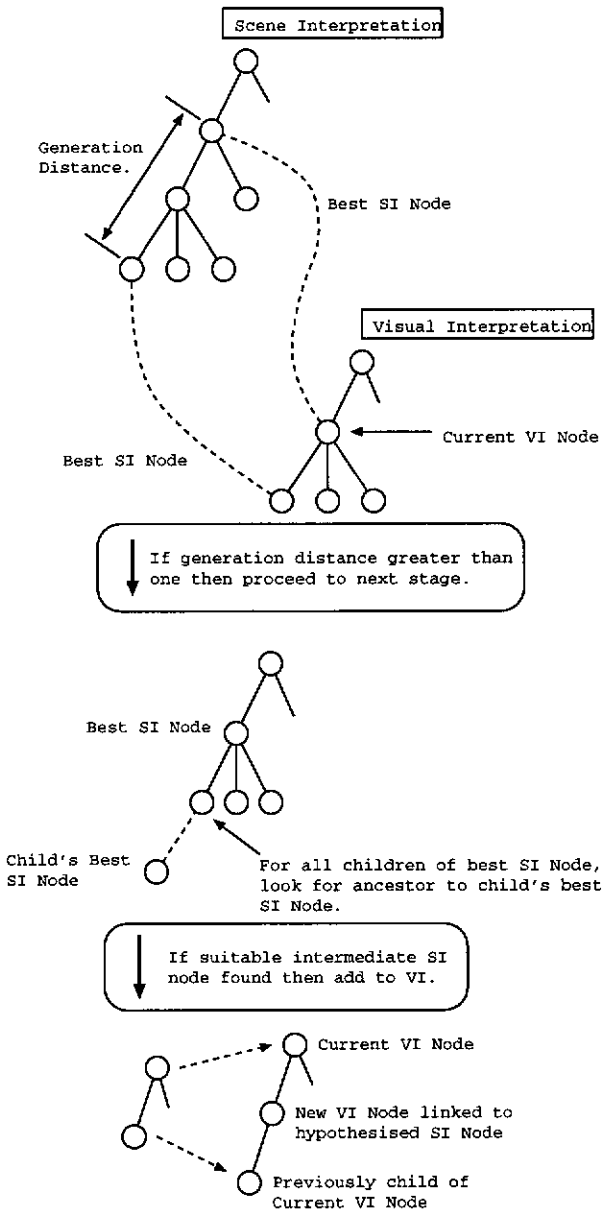


Figure 7.3: **Description of ProcessVIFromSI Algorithm**

7.3.4 Generating SI Nodes from the VI Graph

When grouping occurs in the VI graph this structure must also be translated into the SI graph. Region grouping will occur in the VI graph as a result of connected components grouping, which is only used during early training, and the more sophisticated region grouping by clique resolving which is used a majority of the time.

The translation of visual interpretation structures up to the scene interpretation is done by the PROCESSSIFROMVI operator. The algorithm for this operator is reasonably straight forward, and is listed in Algorithm 10 in Appendix A.1.3.

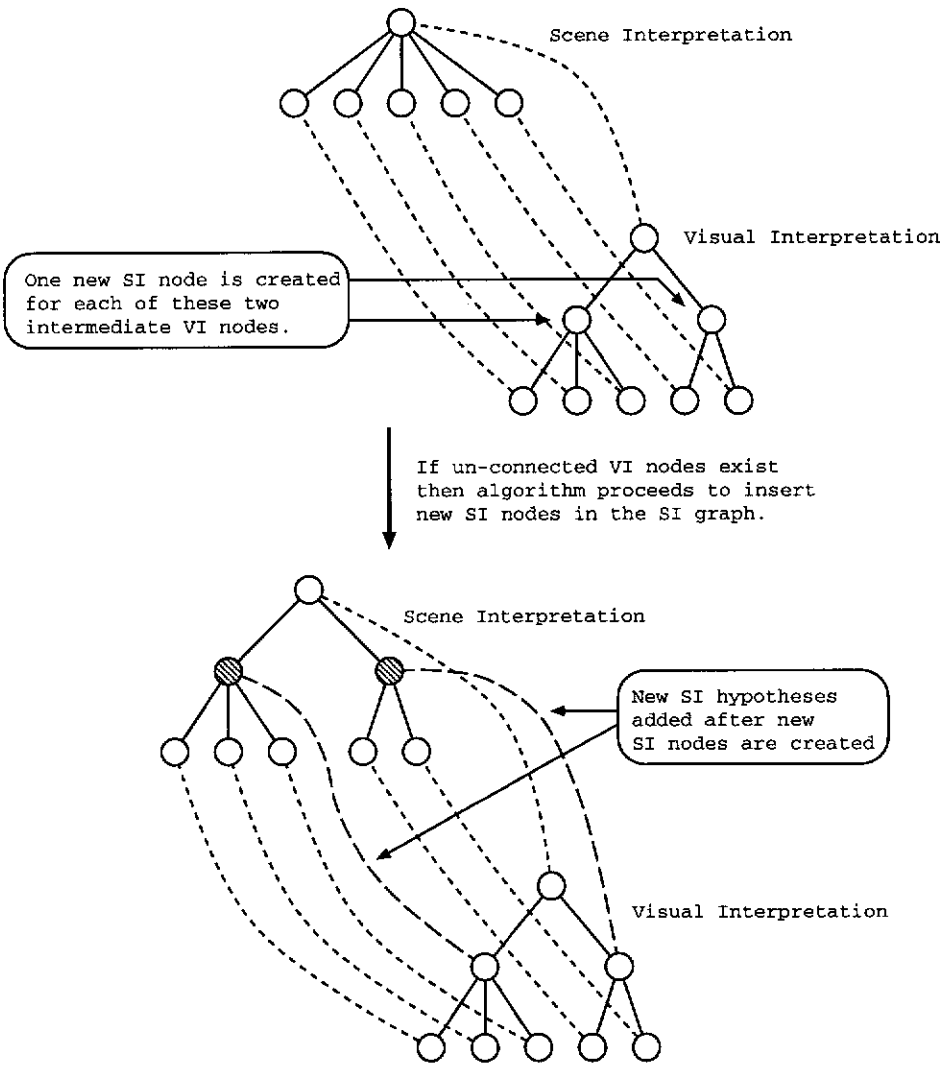


Figure 7.4: **Description of ProcessSIFromVI Algorithm**

This algorithm is run recursively on the visual interpretation structure and operates on VI nodes with no SI hypothesis but with children that each have at least one SI hypothesis, as shown in Figure 7.4. Once such a node is found, the MOSTCOMMONPARENT algorithm (see Appendix A.1.5) is used to determine the attachment point in the SI structure, and an intermediate SI node is created. This new SI node has as its children the SI nodes of the VI node's children.

7.3.5 Generating SI Nodes from the KB Graph

The PROCESSSIFROMKB operator runs on the scene interpretation structure and examines the knowledge base looking for missing nodes to insert into the scene interpretation graph. This is a top-down process which expands the scene interpretation graph to include taxonomy and view information relating to already hypothesised existing objects. This algorithm is listed in Algorithm 7 in Appendix A.1.1.

Figure 7.5 illustrates the three main stages of the algorithm, which is not greatly different from the PROCESSVIFROMSI algorithm. The first stage determines nodes in the scene interpretation which require insertion of their descendants based on the structure of the knowledge base. There must exist missing structure, and the best hypothesis for the SI node in question must be sufficiently greater than the others. In the second stage of the algorithm, the most appropriate KB to insert is determined by looking for commonality of ancestry of the best KB node's children. Once this new KB has been determined, a new SI node is added in stage three with one hypothesis pointing to this KB node.

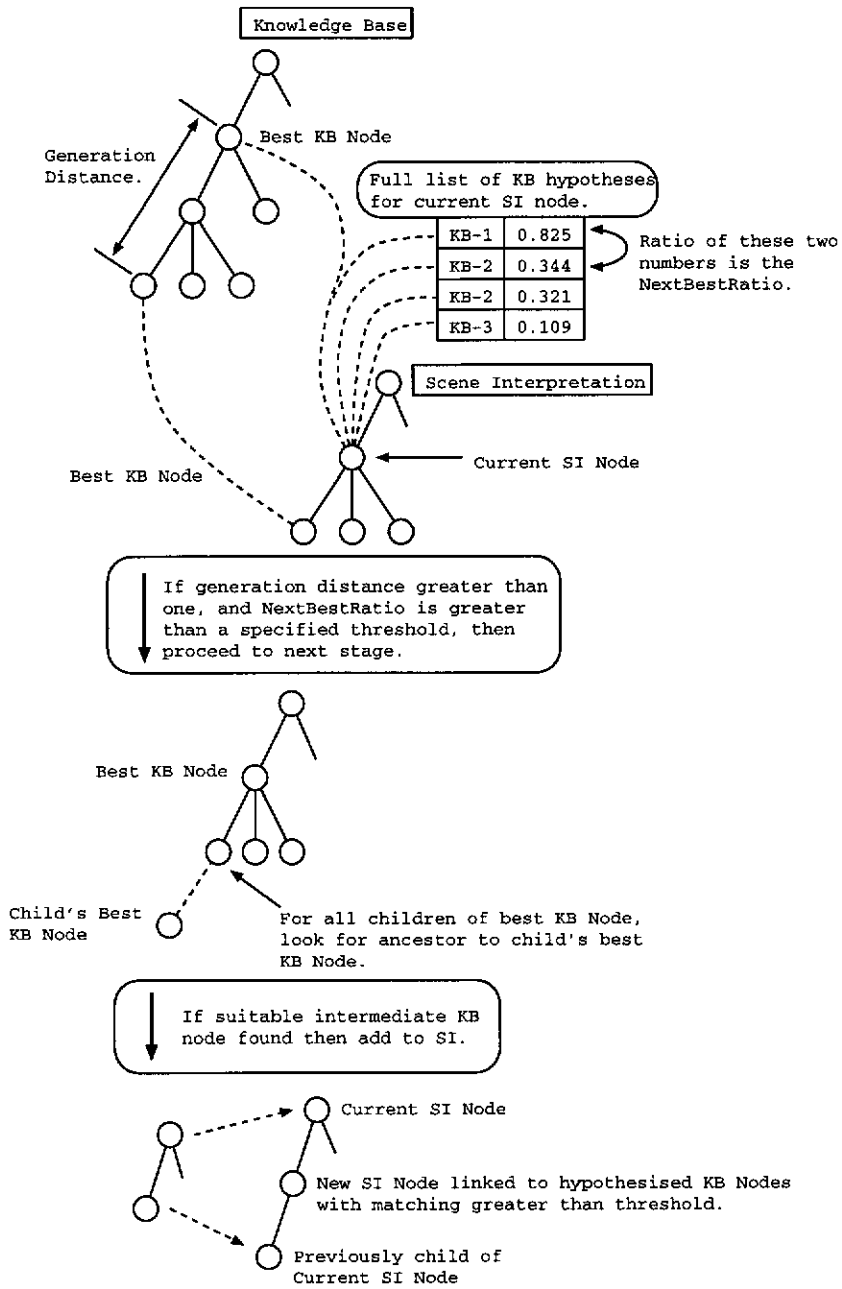


Figure 7.5: **Description of ProcessSIFromKB Algorithm**

7.4 Binary Hypothesis Generation

This section describes the binary hypothesis generation of VI-SI and SI-KB hypothesis links. These procedures can also generate SI and VI nodes as required, and are listed in Table 7.4.

Generates	Operator	Description
VI-SI, SI-KB	BASICUNARY	Basic Unary Matching
SI, SI-SI	CLIQUE	Clique Resolving
SI-KB	MATCHKB	Hierarchical KB Matching

Table 7.4: **Binary Hypothesis Generation Algorithms**

7.4.1 Hypothesis Generation by Basic Unary Matching

The basic unary matching operator generates SI-KB hypotheses for leaf VI nodes by matching them to KB nodes. This is achieved by running the EVALUATEPOINT method in the unary matching strategy stored in the KB node being matched to. The algorithm for the PROCESSUNARYMATCH is described in Algorithm 3 which is run recursively on the visual interpretation structure.

```
if ||CHILDLIST|| = 0 AND ||SILIST|| = 0 then
  KBASE.LEAFUNARYMATCH(UNARY,MATCHWEIGHT,MATCHNODE)
  ADDPOINT = SCENE
  if ||PARENTLIST|| then
    if ||PARENT[0].SILIST|| then
      ADDPOINT = PARENT[0].BESTSINODE
    end if
  end if
  NEWSI = NEW SINODE
  BEST = MAX(MATCHWEIGHT).INDEX
  for all NODE ∈ MATCHNODE do
    if MATCHWEIGHT[NODE] > MATCHWEIGHT[BEST] α_umatch then
      if MATCHWEIGHT[NODE] > α_umin then
        NEWSI.ADDKBNODE(NODE,MATCHWEIGHT[NODE])
      end if
    end if
  end for
  ADDSINODE(NEWSI)
  ADDPOINT.ADDCHILD(NEWSI)
end if
```

Algorithm 3: Algorithm for Generating Basic Unary Matches

7.4.2 Resolving Clique Membership

One important factor in categorising object recognition and scene understanding systems is their ability to isolate and identify multiple touching or overlapping objects. Consider an image in which there are n regions grouped together such that they are path connected across adjacent boundaries. Let us suppose there are m possible objects in the database, each with m_i parts where $m_i \geq 1$ for all i. Without unary or binary pruning, the number of label combinations is q^n where $q = \sum_i m_i$, assuming no objects have shared parts. If we use winner-takes-all unary matching we can reduce the number of label combinations to 1, but we are still left with the number of region grouping combinations being of order ${}^nC_p$ where p is the number of grouped objects in the image. The process of grouping parts or image features into objects or scene elements is called the *clique membership* or *parts clique* problem. This problem becomes extreme when parts-based recognition is used in industrial inspection style bin-of-parts problems.

Connectivity is a common constraint used to reduce the complexity of the clique membership problem in conventional object recognition systems. In this situation, the connectivity constraint simply states that any object or scene element in the image will be constructed from connected regions. This constraint is useful for physically constructed objects, but is too strong for higher level scene elements. For example, Figure 7.6 shows an aerial view of an airport with a firetruck close to a plane. We may wish to describe these two objects as an "emergency" even though the two objects are not physically touching. Note that object recognition systems which are restricted to single isolated objects do not address the clique membership problem.

The clique membership problem is solved in *Cite* using interaction between a number of different processes. Operating on the VI graph, the SEGMENTGROUPCLIQUE process generates sets of possible VI parent nodes, each of which represents a clique. Intermediate SI nodes are then generated by the MATCHSIFROMVI process operating on the SI graph. These generated nodes then interact with the normal relaxation labelling update processes until stable hypothesis weights are achieved. During this process,

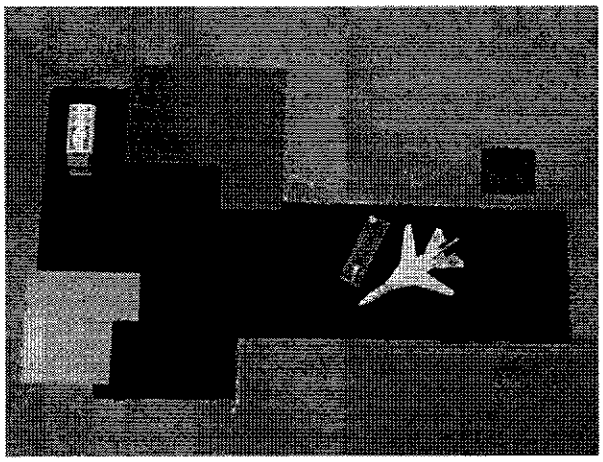


Figure 7.6: **Aerial View of Airport Illustrating Plane and Firetruck in an Emergency**

weak hypotheses (in the form of both SI nodes and SI-KB links) can be removed by the REMOVEWEAK process operating on the SI graph.

The clique membership process SEGMENTGROUPCLIQUE operates on groups of VI nodes that have SI hypotheses which cover KB nodes with different parents. This situation is better understood by examining Figure 7.7. The VI node on which the algorithm is being run is indicated as the "current vi". This VI node has a number of children, each of which has an SI hypothesis. The SI nodes each have hypotheses linking them to the knowledge base. One can see in Figure 7.7 that children of objects labelled "A", "B" and "C" have each been hypothesised.

The clique membership process performs a non-trivial operation only when the cardinality of the set of these KB parent nodes is greater than one. For completeness, the algorithm still operates when only one KB parent node has been hypothesised. In addition, any existing hypothesised KB parent nodes are removed from the set of KB parent nodes. This is important because the clique membership process must interact with other processes which may have grouped VI nodes already, but not separated them from the ungrouped VI nodes.

The objective of the clique membership process is to produce possible groupings of VI nodes based on their common KB parent hypotheses. This is done by indexing the VI nodes against the children of each node in the parent KB node list (denoted "kbplist" in

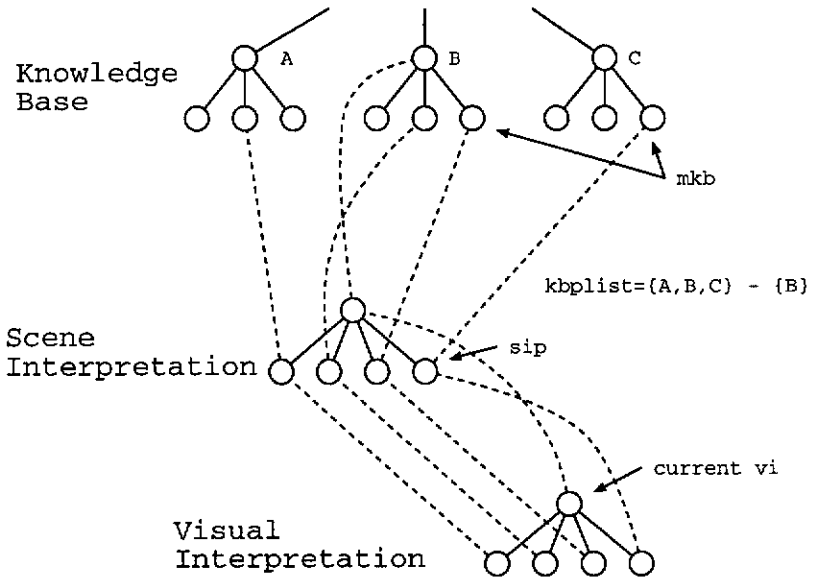


Figure 7.7: **Illustration of Variables for Clique Membership Algorithm**

Figure 7.7). A full set of possible matching child VI nodes (a clique) is constructed by enumerating these index sets. The typical result is a small number of possible cliques that match each KB parent node.

Following this, the clique membership process generates all of what now becomes an intermediate layer of VI nodes, each one representing part of a possible decomposition of the original set of VI nodes (the children of the "current vi" node). This process does not generate any SI hypotheses, or any binary hypotheses between VI and SI or SI and KB nodes. Unary and binary features of the cliques are calculated as a consequence of the construction of new VI nodes. This is done by the feature calculation processes as soon as they detect recently created VI nodes. The clique membership algorithm is listed in Algorithm 4.

An important consequence of this type of approach to solving the clique membership problem is that it is essentially a local operation. Once created, each clique is treated independently of the others. Multiple parent hypotheses in the VI graph are resolved through relaxation labelling processes. Searching for a global optimal solution is neither computationally or performance desirable. For example, strongly matched cliques may require no more processing, while image regions corresponding to weakly matched cliques may undergo resegmentation, merging or further feature extraction. Attempt-

```
KBPLIST = {LIST OF BEST KB NODE PARENTS}
KBPLIST = KBPLIST - {LIST OF EXISTING KB NODE PARENTS}
for all KBP ∈ KBPLIST do
  for all VI ∈ CURRENT_VI.CHILDLIST do
    SIP = VI.BESTSI
    for all MKB ∈ SIP.KBLIST do
      INDEXLIST[MKB] = INDEXLIST[MKB] + {VI}
    end for
  end for
  if ∏CKBP∈KBP.CHILDLIST ||INDEXLIST[CKBP]|| ≠ 0 then
    for all INDEX ∈ PERM(INDEXLIST) do
      NEWVI = NEW VI NODE
      NEWVI.CHILDLIST = CURRENT_VI.CHILDLIST.SUBSET(INDEX)
    end for
  end if
end for
```

Algorithm 4: Algorithm for Resolving Clique Membership

ing to define a global metric that could incorporate these factors would be difficult (if not impossible) and would give a better solution only in extreme cases of bin-of-parts problems where the gradient descent nature of relaxation labelling might fail.

It is important to note that the local nature of this clique membership resolving algorithm is local *with respect to the knowledge base*, not with respect to the image. This is an important knowledge-based constraint that is based on the *structure* of the knowledge base, and not on the structure of the viewed image. Had the connectivity constraint been used, the process would have been unable to handle complex natural-world constructions such as those illustrated in the airport scenes example in Section 11.3.

7.4.3 Hypothesis Generation by KB Matching

When groups exist in the scene interpretation structure they need to be matched against the knowledge base to determine the most likely object labelling. This involves examining the unary matching strengths in addition to executing the binary matching strategy. *Cite* uses unary matchings to prune the possible parent object labellings, and the process which conducts the matching is the PROCESSMATCHKB operator. The algorithm for this operator is listed in Algorithm 12 in Appendix 7.4.3.

Figure 7.8 illustrates the principal of the PROCESSMATCHKB operator. Possible parent labellings are pruned by the child matches, and the degree of match which is eventually loaded into the parent SI node hypothesis combines unary and binary matching strengths.

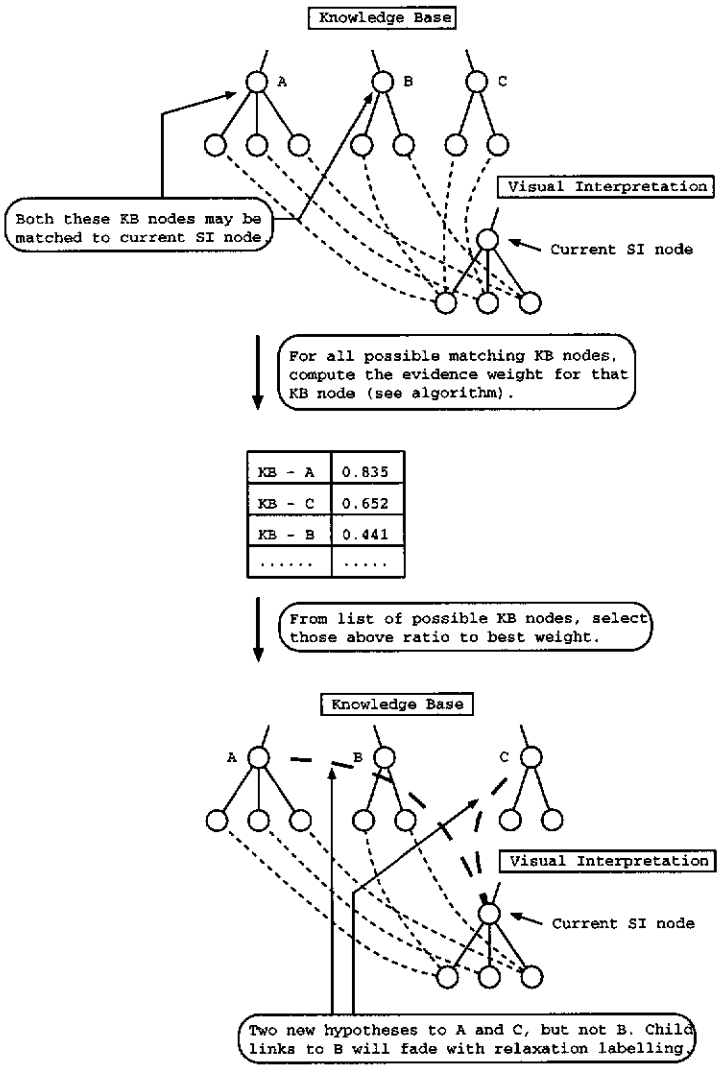


Figure 7.8: **Description of ProcessMatchKB Algorithm**

7.4.4 Weak Hypothesis Removal Process

The clique resolving process can generate multiple possible parent nodes for each grouping of child nodes in the visual interpretation, and this structure is translated into the scene interpretation. As the matching and relaxation labelling operations stabilise, it

is advantageous to remove weak hypotheses to simplify the final "best" interpretation.

The objective of the PROCESSWEAKREMOVAL operator is to remove weak hypotheses such that the scene interpretation and visual interpretation *graphs* are reduced to *trees* where each node has exactly one parent (other than the top level node). Any cycle in the SI or VI implies an ambiguous or unresolved labelling. This operator is straight forward because the relaxation labelling process will reduce the *unary hypothesis weight* of the least successful SI nodes identically to 0.0, which is easily detected.

When hypotheses are removed, the issue of overall system stability becomes important. If, for example, the PROCESSWEAKREMOVAL operator were to remove hypotheses (unary or binary) before another operator considered them to be suitably stable to operate on then the originating process would continually create the hypotheses just to have them prematurely removed. For this reason, the threshold at which the PROCESSWEAKREMOVAL operator functions is always higher than any other operator, implying that it should be the last to operate on any hypothesis configuration. The direct result of this is that this operator is essentially a *post-processing* operator and plays an insignificant role in the dynamic behaviour of the system.

Chapter 8

Relaxation Labelling with Hierarchical Constraints

As described in the previous chapter, *Cite* generates labelling hypotheses for each region in the viewed image, as well as for groupings of regions and the existence of higher level nodes within the scene interpretation and visual interpretation graphs. Typically there are multiple labellings and hypothesised memberships for each node.

The method by which these multiple hypotheses are resolved against the unary and binary rule structures of the knowledge base is, in general, the main focus of most implicit-representation object recognition research. In CRG, for example, a path through alternating unary and binary decision spaces is generated for each region, and then cliques are formed to construct objects. Graph matching techniques exhaustively enumerate possible label sets to determine optimal groupings. BONSAI uses rule activation table lookup, and more symbolic representation systems such as ACRONYM evaluate region properties against stored symbolic descriptions of objects in a winner-takes-all approach.

Relaxation labelling is a method of resolving multiple hypothesis labellings with respect to a compatibility function which computes the compatibilities of two pairwise labellings. Relaxation labelling is essentially gradient descent, and consequently cannot

guarantee a globally optimal solution [Rosenfeld, Hummel and Zucker, 1976]. However, the iterative nature of the relaxation labelling process is ideal for the purposes of *Cite*.

This chapter begins with a discussion of conventional non-hierarchical relaxation labelling. A method of making this hierarchical is discussed, and the relaxation labelling processes used in *Cite* are then explained in detail.

8.1 Non-Hierarchical Relaxation Labelling

Relaxation labelling [Hummel and Zucker, 1983] is a process in which initial probabilities for object labellings are updated iteratively according to predefined compatibilities of these object labellings. One property of this process is the ability to propagate local constraints through the interaction of compatible or incompatible labels. This property has been used successfully at the pixel level in line and curve enhancement [Duncan and Birkholzer, 1992]. In this section, the notation by Rosenfeld, Hummel and Zucker [Rosenfeld et al., 1976] (from [Pelillo and Refice, 1994]) is used.

Let $\mathbf{B}$ be a set of objects $\{b_1, ..., b_n\}$, and $\mathbf{\Lambda}$ be a set of labels $\{1, ..., m\}$. For each object b_i we can obtain an initial label probability vector $\mathbf{P}_i^0 = (p_{i1}^0, ..., p_{im}^0)$ where $0 \leq p_{ij}^0 \leq 1$, for $i = 1...n$ and $j = 1...m$, and $\sum_j p_{ij}^0 = 1$, for $i = 1...n$. This is illustrated in Figure 8.1 where the probabilities are represented as the lines connecting objects to labels.

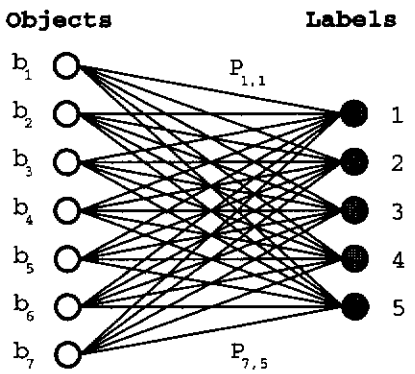


Figure 8.1: **Relaxation Labelling Notation**

Each initial probability vector is interpreted as the prior probability distribution of labels for that object. In an object recognition system, these prior probabilities can be computed from the initial unary and binary matching.

In a relaxation labelling scheme, the objective is to obtain a unique label for each object subject to the specified compatibility constraints. These constraints can be described as a $n \times n$ block matrix $\mathbf{R}$:

$$R = \begin{bmatrix} R_{11} & \cdots & R_{1n} \\ \vdots & \ddots & \vdots \\ R_{n1} & \cdots & R_{nn} \end{bmatrix} \tag{8.1}$$

where each R_{ij} is a $m \times m$ matrix of non-negative real-valued compatibility coefficients:

$$R_{ij} = \begin{bmatrix} r_{ij}(1,1) & \cdots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \cdots & r_{ij}(m,m) \end{bmatrix} \tag{8.2}$$

The coefficient $r_{ij}(\lambda, \mu)$ is a measure of the compatibility between object b_i being labelled λ and object b_j being labelled μ. The relaxation labelling algorithm iteratively updates the probability vectors $\mathbf{P}$ using a normalised weighted sum equation:

$$p_{i\lambda}^{t+1} = \frac{p_{i\lambda}^t q_{i\lambda}^t}{\sum_{\mu=1}^{m} p_{i\mu}^t q_{i\mu}^t} \tag{8.3}$$

where the denominator is the normalisation factor and:

$$q_{i\lambda}^t = \sum_{j=1}^{n} \sum_{\mu=1}^{m} r_{ij}(\lambda, \mu) p_{j\mu}^t \tag{8.4}$$

The objective is to end up with a unique label for each object. Depending on the com-

patibility coefficients $r_{ij}(\lambda, \mu)$ it is not possible to guarantee convergence. As relaxation labelling is essentially gradient descent there is no guarantee that the final outcome will be optimal with respect to the compatibility coefficient matrix R. Optimality can only be guaranteed with near-exhaustive search, which has been shown to be NP-complete [Haralick, Davis, Rosenfeld and Milgram, 1978].

Most research into relaxation labelling takes the form described above. In this form, the number of objects is constant and the number of iterations, t, is the same for each object. However, *Cite* contains hierarchical knowledge in the form of a semi-restricted graph and can generate and remove image regions dynamically. Static hierarchical relaxation labelling has been discussed briefly in the literature [Davis and Rosenfeld, 1981], but not considered further.

8.2 Relaxation Labelling in *Cite*

To simplify the notation, the relaxation labelling process is described in terms of operations on the probabilities listed in Table 7.1. Relaxation labelling is achieved in a two stage process. The first stage involves computing the un-normalised estimator for the probability, and the second involves normalising this against other competing hypotheses.

There are a total of six different types of hypothesis in *Cite*. There are two unary existence hypothesis types which represent the probability that any given VI or SI node exists. There are two inter-graph hypotheses which represent the probability that a VI or SI node is a child of its parent. Finally, and generally most importantly, there are the intra-graph hypotheses which represent the probability that a given SI node is represented in the image by a given VI node, and that a given SI node is an instance of a given KB node. In many respects, the most important of these is the SI-KB hypothesis which represents the object labelling of each scene element.

One advantage in having a hierarchical knowledge base (and as a consequence of top-down hierarchy matching processes, hierarchical scene and visual data graphs) is that

these hierarchies can be used to prune the hypothesis update process. This effectively puts large blocks of 0-value into the compatibility matrix **R** in situations where the hierarchical structure of the knowledge base would be violated. As such, it is more efficient to recast the relaxation labelling process as a hypothesis update process and only store non-zero hypotheses.

The very sparse compatibility matrix used in *Cite* also needs to be dynamically recomputed with the creation and destruction of hypotheses. It is more efficient to decompose the compatibility matrix down to constituent parts applicable to each of the six types of hypothesis, and explicitly represent these as dynamic calculations which are computed at each iteration. These equations are dynamic, not in the dynamic programming sense, but in that the index sets over which the partial summations are computed can change from one iteration to the next.

The following six sections describe the update processes for each of the six hypothesis types.

8.2.1 Updating SI-KB Link Hypotheses

SI-KB link hypotheses represent the best matches of each SI node to nodes in the knowledge base. They are updated according to the level of support given by the SI children *and* SI parents, as follows:

$$\widehat{P}^{SK}_{i,j} = (1-\alpha_{pc}) \sum_{\lambda \in S^C_i} P^S_\lambda P^S_{\lambda,i} \sum_{\xi \in K^C_j} P^{SK}_{\lambda,\xi} + \alpha_{pc} \sum_{\lambda \in S^P_i} P^S_\lambda P^S_{i,\lambda} \sum_{\xi \in K^P_j} P^{SK}_{\lambda,\xi} \quad (8.5)$$

The initial hypothesis value is set by the unary and/or binary matching from the operator which created the given SI-KB hypothesis. The update ratio of parent and child support (α_{pc}) reveals some asymmetry in the update procedure in terms of the relative importance of children and parents. This is necessary because, in general, there are fewer parents of a SI node than children.

In Equation 8.5, the double summations represent the summing over what are termed *compatibility cycles* between the scene interpretation and knowledge base graphs. Figure 8.2 illustrates a compatibility cycle including knowledge base nodes (a) and (b) and scene interpretation nodes (c) and (d). The compatibility cycle is a cycle comprising four hypotheses and is computed through the parent chain (right half of Equation 8.5) and through the child chain (left half of Equation 8.5). There are only three terms, rather than four, in each half of the update equation because the parent-child knowledge base link has a set hypothesis weight of 1.0. An extension to the knowledge base facilitating partial belief in the knowledge structure could be achieved by including a non-unity term into this equation.

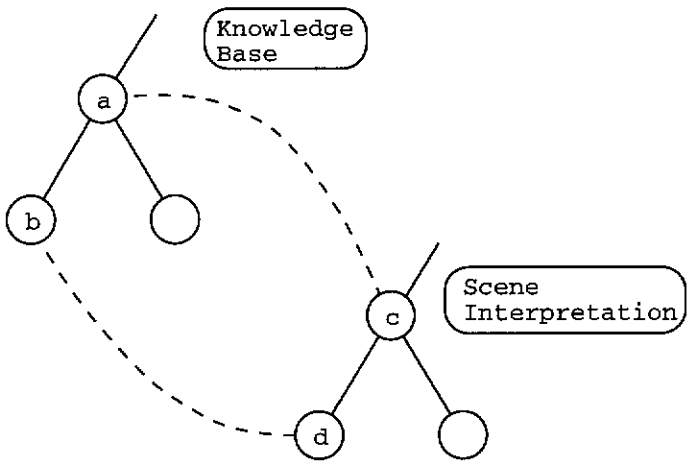


Figure 8.2: **Update Diagram for $P_{i,j}^{SK}$**

8.2.2 Updating SI Link Hypotheses

The SI link hypotheses are only updated in a parent-looking direction. This is done simply because the SI-SI hypothesis links are undirected and the same weight applies to the parent→child interpretation as the child→parent interpretation of the hypothesis. The SI-SI hypothesis update method is straight forward, with each parent weight being updated according to the unary hypothesis strength for the existence of that SI node:

$$\hat{P}_{i,j}^{S} = P_{i,j}^{S} P_{j}^{S} \tag{8.6}$$

This is then subject to the normalisation constraint:

$$\sum_{\lambda \in S_i^P} P_{i,\lambda}^S = 1 \, \forall \, i \tag{8.7}$$

8.2.3 Updating SI Existence Hypotheses

The SI existence hypothesis, P_i^S, is updated from the sum of the visual interpretation support and the SI-KB hypothesis links, as follows:

$$\widehat{P}_i^S = \frac{1}{\|\{j : i \in V_j^S\}\|} \sum_{\lambda \in \{j : i \in V_j^S\}} P_{\lambda,i}^{VS} + \frac{1}{\|S_i^K\|} \sum_{\lambda \in S_i^K} P_{i,\lambda}^{SK} \tag{8.8}$$

This is thresholded to the range [0.0,1.0], and saturates when there is firm belief that the SI node in question actually is a real scene element. When either the matching support is removed (bad matches, or competing cliques), or when the visual interpretation support is removed (better labelled cliques, or better cliques resulting from resegmentation) the SI hypothesis strength drops rapidly to zero.

8.2.4 Updating VI-SI Link Hypotheses

The VI-SI links are updated subject only to the unary SI and VI probabilities, as follows:

$$\widehat{P}_{i,j}^{VS} = P_{i,j}^{VS} P_i^V P_j^S \tag{8.9}$$

These are then subject to the normalisation constraint that:

$$\sum_{\lambda \in V_i^S} P_{i,\lambda}^{VS} = 1 \, \forall \, i \tag{8.10}$$

8.2.5 Updating VI Link Hypotheses

The VI-VI hypotheses are initialised to unity and then normalised at each update stage in a parent-traversing direction weighted by the VI unary hypotheses. The update equation is simply:

$$\widehat{P}_{i,j}^{V} = P_{i,j}^{V} P_j^V \tag{8.11}$$

As with the SI-SI links, this is then subject to the normalisation condition:

$$\sum_{\lambda \in V_i^P} P_{i,\lambda}^{V} = 1 \, \forall \, i \tag{8.12}$$

8.2.6 Updating VI Existence Hypotheses

The visual interpretation nodes are updated according to the number of pixels not uniquely associated with that region and according to the SI unary hypothesis strength for any SI nodes that the VI node is hypothesised to, as follows:

$$\widehat{P}_i^V = \left(\sum_{\lambda \in V_i^S} P_{i,\lambda}^{VS} P_\lambda^S \right) - \alpha_{mr} \text{PixelOverlapRatio}(i) \tag{8.13}$$

The PIXELOVERLAP function returns the fraction (between 0.0 and 1.0) of pixels in

multiple regions, and α_{mr} is a mixing coefficient (default value of 0.1).

8.3 Weight Update Process

The weight update operator, PROCESSUPDATEWEIGHTS, works in two stages. The first stage calculates new estimators for each hypothesis while the second stage transfers these estimators from a temporary buffer into the primary weight variable for each hypothesis. This two stage process is required because each weight calculation uses other weights in the calculation process. This is a standard approach used in many situations where a parallel update is required to be executed on a sequential machine.

Each of the two stages in PROCESSUPDATEWEIGHTS is broken into six functional blocks, each re-estimating or updating one type of hypothesis in the visual or scene interpretation hierarchies.

Chapter 9

Knowledge Driven Segmentation

The purpose of image segmentation is to group pixels into regions that represent parts or objects. Once these groupings have been made, recognition is achieved using computed properties of these parts or objects. A common misconception (caused undoubtedly by the simplicity of the approach) is that it is possible to perform image segmentation in a reliable manner using primitive attributes such as colour and texture with no prior knowledge of image content. There is no foundation for this belief, and *Cite* demonstrates the importance of using knowledge to drive the segmentation process.

Most object recognition and scene understanding systems use uncontrolled segmentation; uncontrolled in the sense that the algorithms and parameterisations of these algorithms are chosen manually after long testing to produce the required results from the input data in question. In *Cite*, segmentation is very much a controlled process in which the current expectation of scene content is used to determine which segmentation algorithm and parameterisation is to be used. This is achieved by storing in each knowledge base node a list of those segmentation procedures and parameterisations which have been the most effective at segmenting the object or scene element represented by that node. Image segmentation may occur on any arbitrary sub-set of the image, and it is not uncommon for *Cite* to be executing half a dozen different segmentations at once.

By closing the loop on segmentation, *Cite* is able to apply multiple segmentation algorithms and parametrisations to different regions of the image based on ongoing hypotheses of scene content. This provides the overall system with greater robustness to noise and distortion, and permits a much broader range of objects and scene elements to be recognised using the one methodology. Some systems do actively search the image based on expectation (SIGMA and SCHEMA) however the mechanisms for doing so are constructed under specific processes, and do not apply in a general way to the knowledge structures or the hypothesis generation process.

This chapter describes the segmentation processes, how they are stored and executed, and the process of resegmenting areas of an image according to the current scene interpretation. Chapter 10 describes the unary and binary features calculated on the segmented images.

9.1 The Segmentation Cycle

Segmentation and resegmentation occur continuously in *Cite*. Every node in the visual interpretation graph has a *segmentation stack* which contains the segmenter currently being executed on that part of the image represented by that node. Segmenters are added to the segmentation stack by bottom-up and top-down processes. The segmenters are executed effectively at random and quite often take varying amounts of time to compute depending on the algorithm being run and the size and complexity of the region.

When an image is first loaded the *default segmenter* is executed on this image. The default segmenter is the segmenter contained in the top level "World" knowledge node. Segments produced from this initial segmentation will then cause labelling hypotheses and part groupings to be generated, and in this way the scene interpretation structure will begin to take form. As the hypothesis generation and relaxation labelling processes proceed, there will be numerous occasions when *Cite* realises that although it may believe a certain object to exist in the scene, it may have an incorrect number of parts or badly matching parts. In these situations *Cite* consults the knowledge base

to determine the best possible segmenter to apply to the sub-image of the ambiguous object or scene element, and loads this segmenter into the segmentation stack in that VI node.

When the new segmenter has been executed, *Cite* may have a better idea of what object is present and may initiate another resegmentation based on this refined knowledge. It may also split the object into a number of different objects, or it may discover that the latest segmentation provides a satisfactory decomposition of that part of the scene.

9.1.1 The Resegmentation Algorithm

Once an initial segmentation has been completed and hypotheses created, *Cite* tests these hypotheses to determine if sub-region resegmentation is possible and appropriate.

The algorithm for determining resegmentation is reasonably simple. If the SI node currently being considered for resegmentation has multiple KB hypotheses and the best of these is significantly better than the others, then this node is considered for resegmentation according to the dictates of the KB node hypothesised. *Cite* accesses the knowledge base to determine the best segmenter to run on the region, and this is then loaded into the segmenter stack of the region's VI node. This algorithm is detailed in Algorithm 5.

To determine the best segmenter to use at each resegmentation stage, *Cite* accesses the most likely KB parent node. If this KB node has a segmenter, this is used. If it does not, the system recursively looks back up through the knowledge base to obtain the next closest node that does contain a segmenter. Once located, the segmenter is compared to those already run on the region by examining the *segmenter history stack* in the VI node for that region. If the new segmenter has not been previously executed on this region, the resegmentation occurs.

An important graph maintenance function is undertaken when resegmentation occurs. The SI nodes and their supporting VI nodes below the SI node to be resegmented are all deleted. In addition, all SI nodes with only one parent and one child above the SI

```
if (CHILDLIST = ∅) ∪ (KBLIST = ∅) then
  RETURN
end if
NUMFOUND = 0
KB = KBLIST.BESTMATCH
for all KC ∈ KB.CHILDLIST do
  FOUND = FALSE
  for all C ∈ CHILDLIST do
    if C.KBLIST = ∅ then
      RETURN
    end if
    if C.KBLIST.BESTMATCH = KC then
      FOUND = TRUE
      BREAK
    end if
  end for
  if FOUND then
    NUMFOUND ++
  end if
end for
if (KB.NUMCHILDREN = NUMCHILDREN) ∩ (NUMFOUND = NUMCHILDREN) then
  RETURN
end if
SEG = KB.SEGMENTER(0)
if VILIST.BESTMATCH.MATCHSEGMENTER(SEG) then
  RETURN
end if
DELETESUBTREE
VILIST.BESTMATCH.INITIALISESEGMENTER(SEG)
```

Algorithm 5: Algorithm for Initiating Knowledge Driven Resegmentation

node to be resegmented are also deleted. This second phase of deletion is important in situations where deep instantiations of taxonomic decomposition have been created before the resegmentation process became possible. If these higher level SI nodes were to remain, the matching of the newly segmented image region would be restricted too far down the knowledge hierarchy. This means that leaf nodes generated by a resegmentation process can, in these circumstances, end up being placed in the visual interpretation structure at a higher level that the parent VI node which originally initiated the resegmentation.

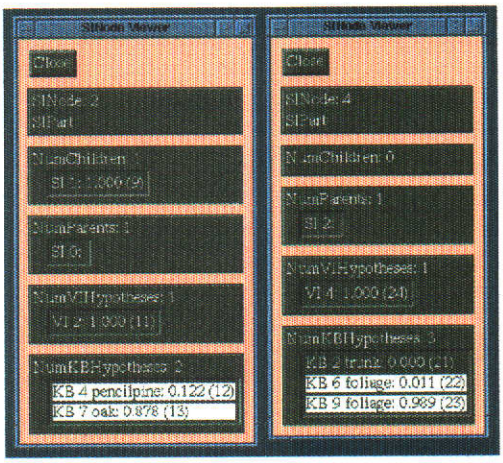


Figure 9.1: **SI Viewing Window Indicating Displayed Hypothesis Weights**

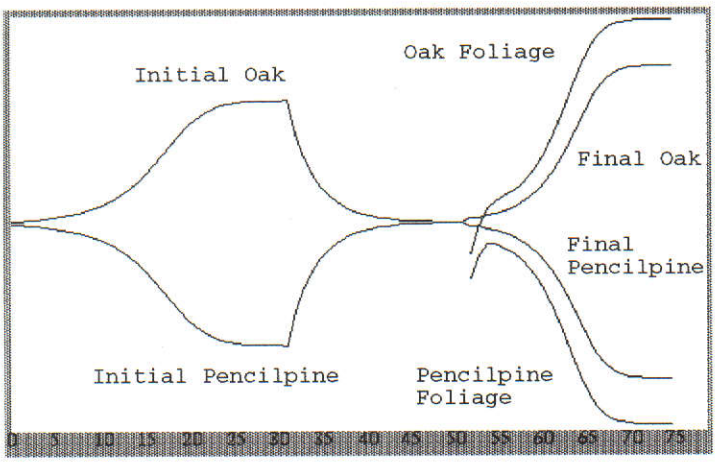


Figure 9.2: **Hypothesis Graph Window Illustrating Selected Hypothesis Weights**

9.1.2 Interaction of Relaxation Labelling and Resegmentation

The relaxation labelling and resegmentation processes in *Cite* interact quite closely in a dynamic way, which is illustrated by a simple example of a two stage segmentation and recognition of an oak tree. Figure 9.2 illustrates four hypotheses as a function of time as the relaxation labelling and resegmentation processes interact. Figure 9.1 shows the SI node viewing windows for the parent SI node and one of the subsequently generated child SI nodes (corresponding to the "foliage" part of the tree). In this example, the object in question is a single tree that matches well to the "oak" and to the "pencilpine" after the initial segmentation. The object has been imperfectly segmented (in this case

under-segmented) and only one part is available for matching.

In this example, initial segmentation occurs at iteration 0. Relaxation then occurs from iteration 0 to iteration 30, at which point resegmentation occurs. Note that the hypothesis weights move towards 0.5 from iteration 30 through to iteration 52. This occurs because the supporting child node of the parent object node has been removed during the resegmentation process. The lack of bottom up support effectively lets *Cite* return to a state of equal "confusion". At iteration 52, when the resegmentation process is complete, new child nodes are added. Matches are made between the foliage part of the oak and pencilpine tree, and these have been graphed in addition to the parent node hypotheses. The relaxation labelling process continues to apply, and drives the hypothesis weights towards a conclusion which in this case is correct. Note also that the final weight of the oak tree hypothesis is higher than that which would have been obtained had resegmentation not occurred (the value at iteration 30). Without resegmentation, a weakly matched under-segmented result would have been the best that could have been obtained.

9.1.3 Execution of Segmentation Procedures

The resegmentation process chooses a segmenter from the knowledge base hierarchy based on the expectation of finding the node in question in the image region which is to be segmented. This segmenter is loaded into the VI node segmenter stack and executed in due course by the PROCESSSEGMENTRUN operator.

During execution, the segmentation instance is passed a region description and an image to segment. The region description is in the form of a visual interpretation node, and the image is in an internal object format. When complete, the sub-regions resulting from segmentation are added as children to the parent specified and the segmenter is added to the VI node segmenter history stack.

To enable more fluid operation with the user interface the segmenter routine is called repeatedly and returns control to the process scheduler at regular intervals. To facili-

tate this operation cleanly each segmenter contains all necessary code and variables to remember what processing stage it was up to at each call of the segmentation function. The segmenter is initialised by passing the image and region description as arguments.

9.2 Evidence of the Power of Resegmentation

The Botanist knowledge base classification results are given as an illustration of the power of knowledge driven resegmentation. The Botanist knowledge base used is shown in Figure 5.2 (in Chapter 5), and a sampling of the images covered by this knowledge base is shown in Figure 9.3.

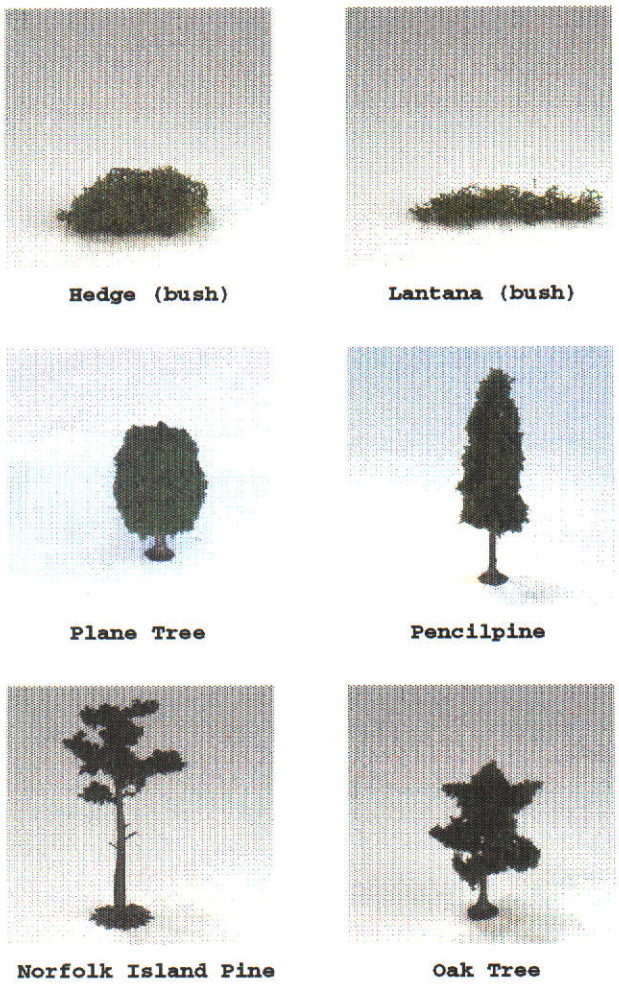


Figure 9.3: **Sample of Images used in Botanist Knowledge Base**

Image	NIPine Seg	PencilPine Seg	Oak Seg	Plane Seg
nipine1	Correct	Correct (2→3)	Correct	Correct (5→2)
nipine2	Correct	Correct	Correct (3→2)	Correct (7→3→2)
nipine3	Correct	Correct	Correct	Correct (4→2)
nipine4	Correct	Correct	Correct	Correct (3→2)
pencilpine5	Correct (1→2)	Correct	Correct (3→2)	Correct (7→3→2)
pencilpine6	Correct (1→2)	Correct	Correct	Correct (3→2)
pencilpine7	Correct (1→2)	Correct	Correct (1→2)	Correct (11→2)
pencilpine8	Correct (1→2)	Correct	Correct	Correct (5→2)
oak5	Correct (1→2)	Correct	Correct	Correct (4→2)
oak6	Correct (1→2)	Correct	Correct	Correct (5→2)
oak7	Correct (1→2)	Correct	Correct	Correct (4→2)
oak8	Correct (1→2)	Correct	Correct	Correct (9→2)
plane3	Correct	Correct	Correct	Correct
plane4	Correct (1→2)	Correct (1→2)	Correct (1→2)	Correct
plane5	Correct (1→2)	Correct (1→2)	Correct (1→2)	Correct
hedge1	Correct	Correct	Correct	Correct
hedge2	Correct	Correct	Correct	Correct
lantana1	Correct	Correct	Correct	Correct
lantana2	Correct	Incorrect	Correct	Correct

Table 9.1: **Botanist Knowledge Base Classification Results**

The classification results are shown in Table 9.1. Each column in this table represents results obtained with a different default segmenter (the segmenter stored in the top-level knowledge base node). The system achieves an average classification rate of %99 when operating normally. Each instance where a resegmentation has occurred is noted; for example (7→3→2) means that the initial segmentation gave 7 parts, the first resegmentation gave 3 parts and the second resegmentation gave 2 parts. In each case of resegmentation, the initial hypothesis or group of hypotheses will have represented either an ambiguous classification, or a weak classification with an incorrect number of children. If the knowledge driven resegmentation is deactivated (any process can be selectively *not* loaded into the process scheduler during startup) the average classification rate drops to %59.

This example illustrates the value of knowledge driven resegmentation. One might think that a single segmenter could correctly segment each of the objects shown in Figure 9.3 because they are all so similar. This example illustrates that even when three segmentation algorithms are available, subtle differences exist between even sim-

ilar objects that give rise to the necessity for knowledge driven segmentation. When one considers a knowledge base containing vastly different objects and scene elements, knowledge driven resegmentation becomes even more important.

9.3 Modification of Segmentation Lists

Through the graphical user interface, the user can add or remove segmenters from any given KB node, and can also change the parameterisations. Figure 9.4 shows the segmenter list editing window open on a KB node, and two segmenters with their parameterisations listed.

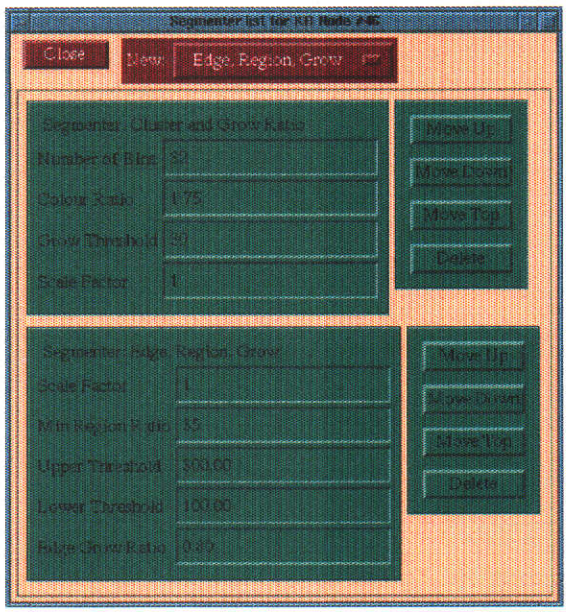


Figure 9.4: **Knowledge Base Segmenter List Editor**

9.4 Segmentation Algorithms

This section describes each segmentation algorithm in *Cite* in detail. The segmentation algorithms have a varying number of parameters, and these are listed and described in each section.

All of the segmenters have a number of common properties, as follows:

- Each segmenter must be able to operate on an arbitrary subset of any of the images currently being processed, which may itself form multiple contiguous regions in the image.
- The segmenter must be able to operate on integrally scaled images, scaled down in width and height by the scale factor θ_s where θ_s is typically 1, 2 or 4.
- The segmenter must be able to run in parallel with other segmenters running on the same or different parts of the same or other images. This is important because it then permits the knowledge driven resegmentation process to operate locally without consideration for other segmentation processes that may be running.
- The segmenter algorithm must be re-entrant to enable smooth operation of the user interface, and contain suitable hooks for the user interface specification of parameters.

In general it is not difficult to obtain these four properties. The most difficult is the ability to operate on an arbitrary subset of an image because of complex boundary effects. The other properties can be achieved by effective software engineering.

One problem that occurs in designing segmentation algorithms is the difference between known and unknown segmentation regions with background removal activated. When the region is not defined (and hence the entire image is being processed) the estimate of the true region size is slightly larger than the resulting region size. This is because the background and the region border may contain noisy pixels which become merged into the background rather than into the region. The main consequence of this is that when

knowledge driven resegmentation is activated, the minimum region size ratio should be reduced by between 3% and 5% when the segmenter is loaded into the knowledge base node corresponding to that object. Quite often this is not necessary because of the stability in the segmentation algorithms, but may be required where the extraction of small noise-sensitive regions is important. In the Tree Taxonomies, for example, the oak tree has such a small trunk in comparison to the foliage that this adjustment to the minimum region size ratio is necessary.

The next three sections describe the SEGMENTSEED, SEGMENTSEEDRATIO and SEGMENTEDGE segmentation algorithms currently used in *Cite*.

9.4.1 Segmentation by Clustering and Growing

The SEGMENTSEED algorithm initially builds a small RGB histogram to find seed points for region growing. The histogram has an equal number of equally sized bins in the red, green and blue axes. The number of bins is given in the parameter θ_b, and typical values are 8, 16 or 32, yielding small memory requirements of 2, 16 and 128 Kbytes respectively[1].

A seed point is chosen as the most common colour in the RGB histogram. Regions are then grown from all pixels of this colour. The regions may incorporate pixels of slightly different colours according to a colour distance threshold, λ_c. This is measured as the sum of the squares of the differences in the red, green and blue planes. All pixels labelled are removed from the histogram. This process is repeated until the histogram is empty and all pixels in the area of the image to be segmented have been labelled. The parameters for the algorithm are obtained from the best hypothesised KB node.

Each region in the label image is then relabelled so that the labels are unique. Regions are then grown to remove small regions. A small region is one whose size is less than the total cover divided by θ_m. In the region growing algorithm, regions are relabelled to the nearest neighbour with the highest matched colour. The region growing process continues until there are no small regions.

[1]The amount of memory required is sizeof(int) * θ_b^3

A full list of parameters for this algorithm is given in Table 9.2. Increasing the colour growing distance λ_c allows region growing to extend further making regions larger. Increasing the minimum region ratio will result in smaller regions being accepted.

Parameter	Symbol	Description	Type	Default
NumBins	θ_b	Number of bins	Integer	16
Distance	λ_c	Colour growing distance	Integer	4000
GrowThreshold	θ_m	Minimum region size	Integer	200
Scale	θ_s	Image scale ratio	Integer	2

Table 9.2: **Parameters for SegmentSeed Segmentation Algorithm**

The SEGMENTERSEED algorithm contains a number of optimised sub-algorithms. After the initial labelling process occurs, the regions will not necessarily be labelled uniquely. This effectively requires a paint-fill style operation which is optimised by building row segments of uniquely labelled pixels and then propagating labels via a connectivity matrix. This speeds up this phase of the segmentation because the paint fill is reduced from a two dimensional to a one dimensional problem. The SEGMENTERSEED algorithm also contains code for computing minimum region size under a number of different circumstances; with and without background removal and under different sub-image conditions.

9.4.2 Segmentation by Clustering and Ratio Growing

The SEGMENTERSEED algorithm was found to work well for a large number of objects, but failed on objects with low contrast parts. This was mainly because colour distance is calculated as an absolute measure, so could not detect changes in badly illuminated objects. A modified version of this segmenter was constructed which computed all colour differences as ratios rather than absolute differences.

A list of parameters for the SEGMENTERSEEDRATIO object is given in Table 9.3. The SEGMENTERSEED and SEGMENTERSEEDRATIO algorithms are similar in design, yet provide sufficiently non-overlapping operational domains to warrant providing both in *Cite.*

Parameter	Symbol	Description	Type	Default
NumBins	θ_b	Number of bins	Integer	16
Distance	λ_c	Colour growing ratio	Real	1.5
GrowThreshold	θ_m	Minimum region size	Integer	200
Scale	θ_s	Image scale ratio	Integer	2

Table 9.3: **Parameters for SegmentSeedRatio Segmentation Algorithm**

9.4.3 Segmentation by Edge Extraction and Merging

The SEGMENTEDGE algorithm initially performs a standard edge detection based on a small local window and added across each of the three colour planes. The edge operator is as follows:

$$\begin{aligned} \text{edge}(x,y) \quad = \quad & |\text{pixel}(x,y) - \text{pixel}(x-1,y)| + \\ & |\text{pixel}(x,y) - \text{pixel}(x+1,y)| + \\ & |\text{pixel}(x,y) - \text{pixel}(x,y-1)| + \\ & |\text{pixel}(x,y) - \text{pixel}(x,y+1)| \end{aligned} \tag{9.1}$$

where $|p|$ is the absolute value of p and:

$$\begin{aligned} |\text{pixel}(\mathbf{a}) - \text{pixel}(\mathbf{b})| \quad = \quad & |\text{pixel}(\mathbf{a}).\text{red} - \text{pixel}(\mathbf{b}).\text{red}| + \\ & |\text{pixel}(\mathbf{a}).\text{green} - \text{pixel}(\mathbf{b}).\text{green}| \\ & |\text{pixel}(\mathbf{a}).\text{blue} - \text{pixel}(\mathbf{b}).\text{blue}| \end{aligned} \tag{9.2}$$

The set of edge pixels E is created by thresholding all pixels in the edge map against an upper threshold λ_u such that all $(\mathbf{a}) \in E$ satisfy $\text{edge}(\mathbf{a}) \geq \lambda_u$. From these edge pixels, the edges are grown in an eight-connected fashion from $\mathbf{a} \in E$ to $\mathbf{b} \notin E$ if $\text{edge}(\mathbf{b}) \geq \alpha_r \text{ edge}(\mathbf{a})$ and $\text{edge}(\mathbf{b}) \geq \lambda_l$. The lower threshold λ_l is chosen such that $\lambda_l \leq \lambda_u$ and the edge growing ratio α_r is chosen such that $0 \leq \alpha_r \leq 1$. This edge growing is iterated until there are no more pixels being labelled as edges. Threshold parameters are obtained from the current best hypothesised KB node for the VI node being segmented.

```
REGIONS = {1,...,NUMREGIONS}
repeat
  OUTERNOCHANGE = TRUE
  for all R ∈ REGIONS do
    if SIZE(R) < TOTALSIZE/θ_m then
      LARGE = 0
      for all S ∈ NEIGHBOURHOOD(R) do
        if SIZE(S) ≥ TOTALSIZE/θ_m then
          LARGE = LARGE + 1
        end if
      end for
      if LARGE = 0 then
        repeat
          NOCHANGE = TRUE
          for all S ∈ NEIGHBOURHOOD(R) do
            if SIZE(S) < TOTALSIZE/θ_m then
              RELABEL S → R
              REGIONS = REGIONS - {S}
              RECOMPUTE NEIGHBOURHOOD
              RECOMPUTE SIZE
              NOCHANGE = FALSE
              OUTERNOCHANGE = FALSE
            end if
          end for
        until (REGIONS = ∅) ∪ NOCHANGE
      end if
    end if
  end for
until (REGIONS = ∅) ∪ OUTERNOCHANGE
for all R ∈ REGIONS do
  if SIZE(R) < TOTALSIZE/θ_m then
    S = LARGEST NEIGHBOUR OF R
    RELABEL R → S REGIONS = REGIONS - {R}
  end if
end for
```

Algorithm 6: Region Merging in SegmenterEdge Algorithm

Once edge growing is complete, uniquely labelled regions are created by a standard recursive paint filling algorithm. The edge pixels E are then eroded away to leave a full covering of the region of the image being segmented.

Regions below a specified fraction of the overall region size are then merged according to Algorithm 6 using parameters obtained from the hypothesised KB node. This algorithm can be broken into two parts. The first part considers small regions only connected to other small regions, and then propagates their labels into other small regions, where a region is small if it covers fewer pixels than the total coverage divided by θ_m. The

second part of the algorithm looks at all remaining small regions and relabels them as their closest large neighbour, for which there will always be at least one.

This region merging is ideal for complex textured regions which yield a large number of small regions placed close together. In cases where small regions may be generated not only from large textured surfaces, a different segmenter can be chosen. A full set of parameters for this segmentation algorithm are described in Table 9.4.

Parameter	Symbol	Description	Type	Default
Scale	θ_s	Image scale ratio	Integer	2
GrowThreshold	θ_m	Minimum region size	Integer	200
Upper	λ_u	Upper edge threshold	Float	300.0
Lower	λ_l	Lower edge threshold	Float	100.0
Ratio	α_r	Edge growing ratio	Float	0.8

Table 9.4: **Parameters for Edge Segmentation Algorithm**

Chapter 10

Feature Extraction

Segmentation and feature extraction are the two most dramatic data reduction stages in a machine vision system. Segmentation places pixels together into groups that have something in common, and feature extraction produces measures on those groups of pixels. Quite often an image region which might represent tens of kilobytes of data can be summarised by a few numbers after segmentation and feature extraction.

The very earliest object recognition systems used classic pattern recognition approaches where features were calculated on the entire object considered as one image entity. For some objects, this works sufficiently well to be useful, but for more complex objects the method fails because the object may contain a number of very distinct parts. When the object is considered as a whole, the useful properties which make these parts so recognisable are lost. A good example of this would be an object like a car, where individually the wheels are a distinctive black colour and very circular, and the body has a uniform colour as well. However if one considers the car as a whole, it no longer has a uniform colour, and the circularity feature would be virtually undetectable.

For this reason, object recognition research very quickly moved to breaking objects into parts using either range, intensity or colour information. Parts were either image regions (or surface patches in the case of range segmentation [Besl and Jain, 1986]) or boundary shape segments. Some systems, such as the Local-Feature-Focus method

[Bolles and Cain, 1982] used a combination of boundary segments and interior features. In all of these methods, features must be computed to describe the regions or boundary segments *and* the relationships between them. Features describing a single region are called *unary* features, and features describing the relationship between two regions are called *binary* features.

Unary features typically include colour, texture, size and shape descriptions of the region. Binary features typically concentrate on geometric properties such as the distance between two regions, length of common border and so on. However, it is conceivable (although not common) that binary features could describe unusual relational features such as how similarly coloured or textured the two parts were.

While some research continues into global features for object recognition (such as Fourier descriptors, radial basis decomposition and occluding contour descriptions [Kriegman and Ponce, 1990]), it is generally held that segmentation into distinct parts or surface patches, each of which is independently described, is the most effective approach. While this may be true for some types of objects, it is unlikely to be true in general due to the difficulty of obtaining robust segmentation. One hint that this is where many problems in machine vision lie is the realisation that segmentation and feature extraction processes reduce the information content by as much as four decimal orders of magnitude, which is by far the largest information reduction step in any vision system.

Cite uses both unary and binary features, which can be calculated across the visual interpretation hierarchy. This represents a broadly quantised contextual scale invariance in the feature extraction process that goes a small way to address this issue of such large information reduction.

This chapter describes the various unary and binary features used in *Cite*, how they are computed, and how they are stored in the visual interpretation structure.

10.1 Feature Storage, Calculation, and Matching

When calculated, features are stored in the visual interpretation nodes. Each visual interpretation node can store all unary and binary features calculated on that node. Binary features stored at a node describe the relationships between the children of the node, whereas the unary features stored at a node describe the node itself, as illustrated in Figure 10.1.

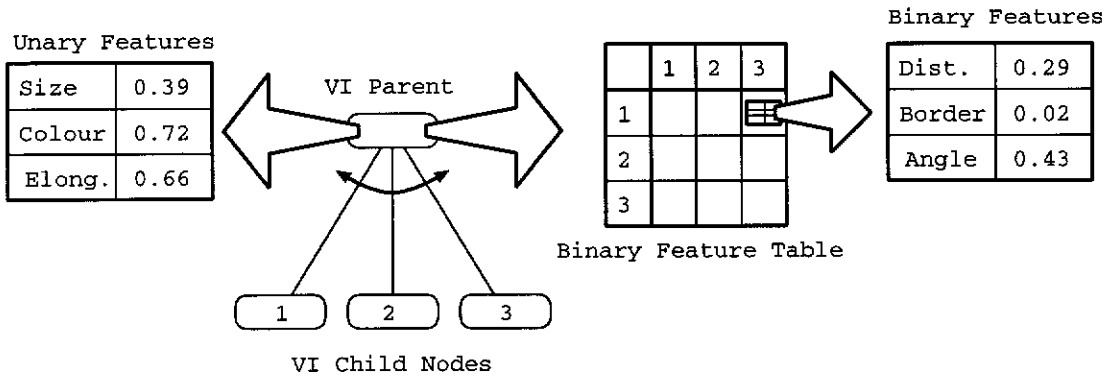


Figure 10.1: **Storage of Unary and Binary Features in VI Nodes**

For the purposes of presenting the learning and classification systems with uniform data, all unary and binary features within *Cite* are scaled to the range 0.0 to 1.0. In the case of features which have no upper bound, a reasonable scaling is applied and then the feature is clipped to 1.0.

Feature *matching* is the process whereby computed features are used to determine object classification. In *Cite*, this process is achieved by the hypothesis generation operators which use the matching algorithms stored in the knowledge base. This process is covered in detail in Chapter 7, but it is important to understand that these operators are responsible for taking the computed features and matching them to the knowledge base.

In *Cite*, unary and binary features are stored in a manner such that feature operators may be easily added and removed from *Cite* to conduct experiments of performance with respect to feature operators. Knowledge base matching strategies that directly use unary and binary feature representations can save these to file in a human readable,

annotated format. The visual interpretation structures can also be saved to file in this manner, facilitating the use of *Cite*'s hierarchical segmentation and feature extraction with other systems. An example of a VI node file format is contained in Figure 10.2

```
VINode
ID 10
NumChildren 0
ChildId
NumParents 1
ParentId  9
NumPixels 6144
PixelXY
 211 116 212 116 213 116 214 116 215 116 216 116
 217 116 218 116 219 116 220 116 221 116 222 116
 ...
Unary
  UF_ILLUMINATION 0.478168
  UF_RED 0.570500
  UF_GREEN 0.453590
  UF_BLUE 0.410413
  UF_ELONGATION 0.535825
  UF_HEIGHTWIDTH 0.328467
  UF_VARRED 0.539429
  UF_VARGREEN 0.447481
  UF_VARBLUE 0.355886
  UF_SIZE 0.238813
  UF_STRAIGHTNESS 0.000000
  UF_AONPSQR 0.875182
  UF_EXTENT 0.000000
EndUnary
Binary
NumParts 0
NumSegmenterList 0
```

Figure 10.2: **VI Node File Format Illustrating Unary Feature Storage**

10.2 Notation

The following two sections, Section 10.3 and Section 10.4, describe how unary and binary features are calculated. In these two sections, R denotes the region of interest and, in the case of binary relations, S denotes the second region used in the calculations.

Let R_N and R_S be the number of pixels in region R and S respectively.

We define R_i is the ith pixel of region R, not sorted in any particular order. The pixel **x** and **y** coordinates are denoted x_i and y_i respectively. The red, green and blue colour components of pixel i are r_i, g_i and b_i, and they can each range between 0 and 255.

10.3 Unary Features

This section describes in detail the unary feature operators available within *Cite*. The choice of features calculated for each region is based on knowledge about the current scene interpretation and learning results. Table 10.1 contains a summary of each feature.

Feature	Type	Description
Illumination	Colour	Average region illumination
Red	Colour	Average region red component
Green	Colour	Average region green component
Blue	Colour	Average region blue component
VarRed	Texture	Variance in region red component
VarGreen	Texture	Variance in region green component
VarBlue	Texture	Variance in region blue component
Size	Geometric	Ratio of region to parent
Elongation	Geometric	Ratio of diameter to area
HeightWidth	Geometric	Height to width ratio
Straightness	Geometric	Vertical straightness measure
AOnPSqr	Geometric	Area to Perimeter ratio
Extent	Geometric	Whether region has infinite extent

Table 10.1: **Summary of Unary Features**

10.3.1 Colour Unary Features

The colour of a region can often give important information as to what the region may be. For example, a blue object is more likely to be sky than grass, given that colour is the only description we have.

The most common representation for colour in machine vision is to express the colour as a red-green-blue (RGB) triple. Other *colour spaces* are sometimes used when required, such as HSV and YIQ. The preference for RGB is due mainly to the fact that most computer frame capture hardware produces RGB data[1].

The current implementation of colour features in *Cite* uses the RGB colour space. The average red, green, blue and illumination values of a region are calculated using the following simple formulae:

$$R_{\overline{red}} = \frac{1}{R_N} \sum_{i \in R} \frac{r_i}{255} \tag{10.1}$$

$$R_{\overline{green}} = \frac{1}{R_N} \sum_{i \in R} \frac{g_i}{255} \tag{10.2}$$

$$R_{\overline{blue}} = \frac{1}{R_N} \sum_{i \in R} \frac{b_i}{255} \tag{10.3}$$

$$R_{\overline{illum}} = \frac{1}{3}(R_{\overline{red}} + R_{\overline{green}} + R_{\overline{blue}}) \tag{10.4}$$

10.3.2 Texture Unary Features

Colour variance (or standard deviation) measures are not a very sophisticated measure of texture because they do not code spatial information. However they do give a rough indication of colour variation in a region. They have one advantage in that, apart from pixel aliasing problems, they are invariant to scale. The standard deviation in red, green and blue are calculated using the following formulae:

$$R_{\sigma_{red}} = 2\sqrt{\frac{1}{R_N} \sum_{i \in R} \left(\frac{r_i}{255} - R_{\overline{red}} \right)^2} \tag{10.5}$$

$$R_{\sigma_{green}} = 2\sqrt{\frac{1}{R_N} \sum_{i \in R} \left(\frac{g_i}{255} - R_{\overline{green}} \right)^2} \tag{10.6}$$

[1]The reason most frame capture hardware produces RGB data by default is because all computer monitors have red-green-blue video output, so displaying images is simply a case of copying data.

$$R_{\sigma_{blue}} = 2\sqrt{\frac{1}{R_N}\sum_{i\in R}\left(\frac{b_i}{255} - R_{\overline{blue}}\right)^2} \tag{10.7}$$

A scale factor of 2 is used to normalise the standard deviations to within (0,1.0). The maximum variance occurs if half of the pixel values are 0 and half are 255. To show that this scaled standard deviation normalises to 1, consider the standard deviation in red when the highest variance conditions are present:

$$\begin{aligned}
R_{\sigma_{red}} &= 2\sqrt{\frac{1}{R_N}\sum_{i\in R}\left(\frac{r_i}{255} - R_{\overline{red}}\right)^2} && (10.8)\\
&= 2\sqrt{\frac{1}{R_N}\left(\frac{R_N}{2}\left(\frac{0}{255} - 0.5\right)^2 + \frac{R_N}{2}\left(\frac{255}{255} - 0.5\right)^2\right)} && (10.9)\\
&= 2\sqrt{\frac{1}{R_N}\left(\frac{R_N}{8} + \frac{R_N}{8}\right)} && (10.10)\\
&= 2\sqrt{\frac{1}{4}} && (10.11)\\
&= 1 && (10.12)
\end{aligned}$$

10.3.3 Geometric Unary Features

Colour and texture features often give a good description of an image region. However, it is important to augment these with geometric descriptions as well. This section discusses each of the geometric unary features used in *Cite.*

10.3.3.1 Size Unary Feature

The unary size feature of a region is given as the square root of the ratio of its size (in pixels) to its parent's size, or in the case of the top-level node (the only visual interpretation node with no parents), it is set to 1. If we define P to be the parent of region R then we know that $R \subseteq P$, and that $R_N \leq P_N$, and so the normalised size can be written simply as:

$$R_{size} = \sqrt{\frac{R_N}{P_N}} \tag{10.13}$$

Taking the square root of the pixel ratio increases the dynamic range of the size feature. If this step is not included, small parts of large objects generate unusable size ratios.

10.3.3.2 Height-to-Width Unary Feature

A normalised *height-to-width* ratio of a region is calculated as the ratio between the height of the region and the sum of its height and width. This is done on a bounding box touching the exterior most points of the region, in the following manner:

$$R_{height-width} = \frac{\max\{y\} - \min\{y\} + 1}{\max\{y\} + \max\{x\} - \min\{y\} - \min\{x\} + 2} \tag{10.14}$$

10.3.3.3 Elongation Unary Feature

The *elongation* of a region is defined as twice the ratio of the maximum diameter to the perimeter. Regions with very jagged edges would give erroneously low elongations, but such regions are comparatively rare. The algorithm for calculating elongation involves first obtaining a set of perimeter pixels, R_P as follows:

$$R_P = \{x,y\} : \{x,y\} \in R \wedge (\{x-1,y\} \notin R \vee \{x+1,y\} \notin R \tag{10.15}$$

$$\vee \{x,y-1\} \notin R \vee \{x,y+1\} \notin R) \tag{10.16}$$

If we define the number of pixels in R_P as R_{P_N}, and $\| \|$ as the Euclidean distance metric, then the elongation can be written as follows:

$$R_{elongation} = \frac{2 \max \|R_{P_i} - R_{P_j}\| \forall i,j \in \{1...R_{P_N}\}}{R_{P_N}} \quad (10.17)$$

10.3.3.4 AOnPSqr Unary Feature

Another common region shape descriptor is the ratio of area to perimeter squared, called "A on P squared". This feature gives an indication of the degree of roughness of the region shape. The AOnPSqr feature is unitless and bounded by $\frac{1}{4\pi}$ corresponding to a circle. The smaller the feature measure the rougher the region shape. AOnPSqr is calculated as follows:

$$R_{AOnPSqr} = 4\pi \frac{R_N}{R_{P_N}^2} \quad (10.18)$$

10.3.3.5 Straightness Unary Feature

An estimate of the vertical straightness of a region can be obtained by fitting a quadratic line to the mid-point of each scan line in the region. The straightness of the region is then inversely proportional to the coefficient of the second order term. This measure does not work well when the region is anything other than a long and thin region, because the centre line of the region is not stable. A better solution is to compute the quadratic coefficients of each side of the region (a_l and a_r) and then compare these in the following manner:

$$v = \beta_s(a_l + \alpha_{dc})(a_r + \alpha_{dc}) \quad (10.19)$$

$$R_{straightness} = \begin{cases} 0 & \text{if } v < 0 \\ \sqrt{v} & \text{if } v < 1 \\ 1 & otherwise \end{cases} \quad (10.20)$$

In Equation 10.19, α_{dc} is a small offset term (currently 10^{-3}) to counter aliasing errors in the computation of the second order term. β_s is a scaling term (currently 10^4) which brings the feature into a more useful range. The computation of the second order term in the quadratic fit is given in Appendix A.3.

10.3.3.6 Extent Unary Feature

The unary extent feature is designed to give *Cite* an indication of whether the size of a region is bounded. For example, sky and ground regions can be considered to have "infinite" extent, and this is a useful attribute of those regions. The unary extent feature is set to 1.0 if the region touches the edge of the image, or 0.0 if it does not.

10.4 Binary Features

This section describes in detail the binary feature operators available within *Cite*. The choice of features calculated for each region are based on knowledge about the current scene interpretation and learning results.

Table 10.2 contains a summary of each binary feature. A binary feature is said to be *directed* if F(a,b) $\neq$ F(b,a). Most learning and binary matching systems are designed to use directed features, and they incorporate undirected features simply by computing both the feature using both region orderings. *Cite* adopts this strategy.

Feature	Description	Directed
Size Ratio	Ratio of image sizes	Yes
Distance	Image distance to size ratio	No
Boundary	Ratio of shared boundary	Yes
Above/Below	Proportion above/below region	Yes

Table 10.2: **Summary of Binary Features**

10.4.1 Binary Size Ratio Feature

The binary size ratio feature provides an indication of how large or small one region is compared to another. For two regions a and b with region sizes N_a and N_b respectively, a size ratio is defined as:

$$v = \frac{1}{2} + \frac{1}{2}\left(\sqrt[3]{\frac{N_a}{N_b}} - 1\right) \tag{10.21}$$

$$R_{sizeratio} = \begin{cases} 1 & \text{if } v > 1 \\ v & \text{otherwise} \end{cases} \tag{10.22}$$

The size ratio measure will thus be 1.0 for regions that are equal to or larger than eight times the size of the region being compared.

10.4.2 Binary Image Distance Feature

A common binary feature is to compute the distance between the centroids of the two regions. This would be measured in image pixels and would not be invariant to scaling. One method to achieve scale invariance is to scale the distance by the average area, as follows:

$$v = \sqrt{\frac{(\bar{x_a} - \bar{x_b})^2 + (\bar{y_a} - \bar{y_b})^2}{16(N_a + N_b)}} \tag{10.23}$$

$$R_{distance} = \begin{cases} 1 & \text{if } v > 1 \\ v & \text{otherwise} \end{cases} \tag{10.24}$$

10.4.3 Binary Boundary Feature

Another common binary feature, also implemented in *Cite*, is the fraction of shared boundary between two regions. Given the hierarchical nature of *Cite*'s segmentation structure, and the fact that pixels may belong to a number of regions at the same level in the hierarchy, there are two ways to compute the binary boundary feature, as illustrated in Figure 10.3

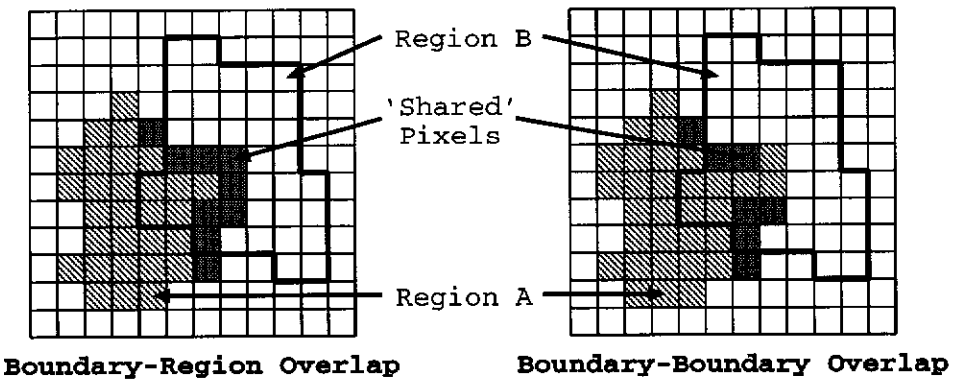


Figure 10.3: **Two Methods for Computing Shared Boundary Ratio**

The *boundary-region* method counts up all boundary pixels of region A that are within one (four connected) pixel of region B. The *boundary-boundary* method counts up all boundary pixels of region A that are within one (four connected) pixel of the boundary pixels of region B. Both methods give identical results when regions are not permitted to overlap, but because in *Cite* this restriction is lifted, the boundary-region method is used. This method is better behaved with regions that significantly overlap. The actual binary boundary feature is the ratio of the overlapping boundary pixels in region A to the total boundary length of region A.

10.4.4 Binary Above/Below Feature

The above/below feature computes the number of pixels of region A that are above region B, n_{above}, and the number that are below region B, n_{below}, and computes the above/below feature as:

$$R_{abovebelow} = \frac{1}{2}\left(1 + \frac{n_{above} - n_{below}}{N_a}\right) \tag{10.25}$$

In Equation 10.25, N_a is the total number of pixels in region A, which means that $R_{abovebelow}$ is correctly bounded between 0.0 and 1.0. The above/below feature gives a good indication of the relative vertical position of objects, and is useful in describing relationships between objects that are always viewed in certain orientations. Figure 10.4 illustrates some examples of this binary feature, which is well behaved in situations where region A has pixels both above and below region B.

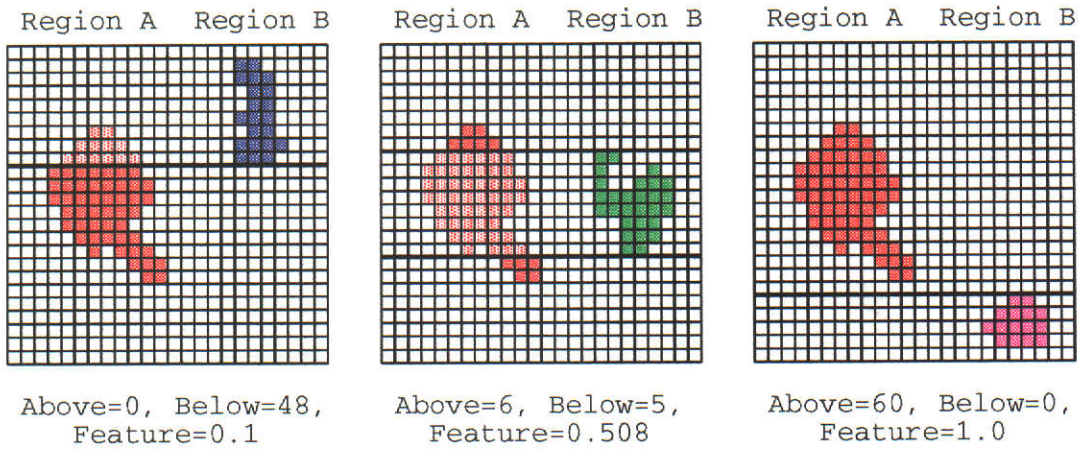


Figure 10.4: **Examples of the Binary Above/Below Feature**

10.5 Feature Invariances

An important characteristic of both unary and binary features is their invariance characteristic. Of most importance to *Cite* is invariance to two dimensional translation, rotation, scale and illumination. Translation invariance is typically the easiest to obtain, however for reasons of global positioning of objects, sometimes features are used that are not invariant to translation. For example, in outdoor scenes, the ground is typically found at the bottom of the image and the sky is typically found at the top of the image.

10.5.1 Unary Feature Invariances

Table 10.3 summarises the invariance properties of the unary features, and Table 10.4 summarises the invariance properties of the binary features. In each of these tables, "yes" means that the feature is invariant to changes in that attribute.

Feature	Illumination	Rotation	Scale	Translation
Red	No	Yes	Yes	Yes
Green	No	Yes	Yes	Yes
Blue	No	Yes	Yes	Yes
Illumination	No	Yes	Yes	Yes
Var. Red	No	Yes	Yes	Yes
Var. Green	No	Yes	Yes	Yes
Var. Blue	No	Yes	Yes	Yes
Size Ratio	Yes	Yes	Yes	Yes
Height-Width	Yes	No	Yes	Yes
Elongation	Yes	Yes	Yes	Yes
AOnPSqr	Yes	Yes	Yes	Yes
Straightness	Yes	No	Yes	Yes
Extent	Yes	Yes	Yes	Yes

Table 10.3: **Invariance Properties of Unary Features**

10.5.2 Binary Feature Invariances

The directedness (or *symmetry*) of a binary feature is an important characteristic. Some binary features, such as relative orientation or relative size, are not symmetric. As mentioned earlier, *Cite* handles symmetric and asymmetric features equally well. Table 10.4 describes each binary feature in terms of its invariance characteristics. An invariance to translation, scale or rotation implies that the feature gives the same result when such an operation is performed about a common (but arbitrary) origin.

Feature	Illumination	Rotation	Scale	Translation	Symmetric
Image Distance	Yes	Yes	Yes	Yes	Yes
Size Ratio	Yes	Yes	Yes	Yes	No
Boundary Ratio	Yes	Yes	Yes	Yes	No
Above/Below	Yes	No	Yes	Yes	No

Table 10.4: **Invariance Properties of Binary Features**

Chapter 11

System Performance and Results

This chapter presents complete results on four different scenarios; views of street scenes, views of office objects, aerial views of airports, and the context sensitive tree taxonomies. The results summary for these scenarios is listed in Table 11.1 and the results for each scenario are contained in seperate sections of this chapter.

In total, 253 scene elements were detected from 38 images of the four scenarios. The knowledge bases for the scenarios were constructed from a collection of 85 images. The knowledge bases themselves contain a total of 155 scene elements at different levels of abstraction. Of the 253 detected scene elements, an average classification success of %96 was achieved.

Scenario	KB Elements	Elements Tested	% Correct
Street Scenes	24	38	95
Office Scenes	64	73	95
Airport Scenes	19	70	100
Tree Taxonomies	48	72	93
Total	155	253	96

Table 11.1: **Summary of Results Presented**

11.1 Views of Street Scenes

The street scene scenarios provide a simple demonstration of *Cite* in a general environment. The knowledge base, shown in Figure 11.1 was constructed from isolated and in-situ examples of objects. In the case of objects of a finite extent such as the buildings and vehicles, these were learnt from isolated examples as shown in Figure 11.2. The objects of unlimited extent such as the various ground covers and the sky were learnt from landscapes containing just those scene elements.

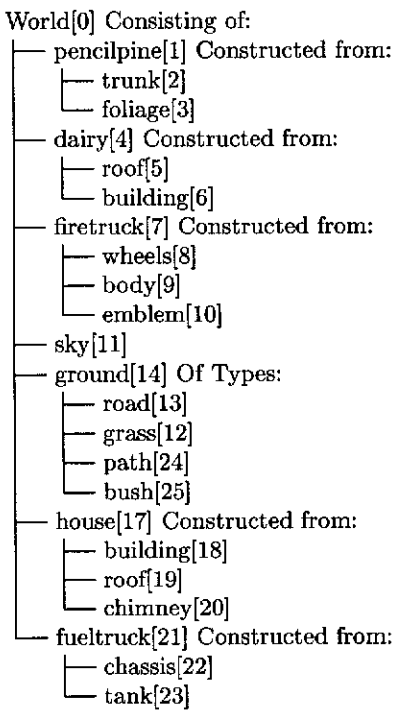


Figure 11.1: **Knowledge Base for Town Scenes**

The following pages contain two example scenarios testing the street scene knowledge base. In the first example, *Cite* correctly recognises the objects present on the first pass and then uses the knowledge driven resegmentation to obtain a finer resolution description. The resulting object labels and segment boundaries are shown overlaying the original image in Figure 11.3. The results are shown both at the leaf node description (most detailed) and at the next level up.

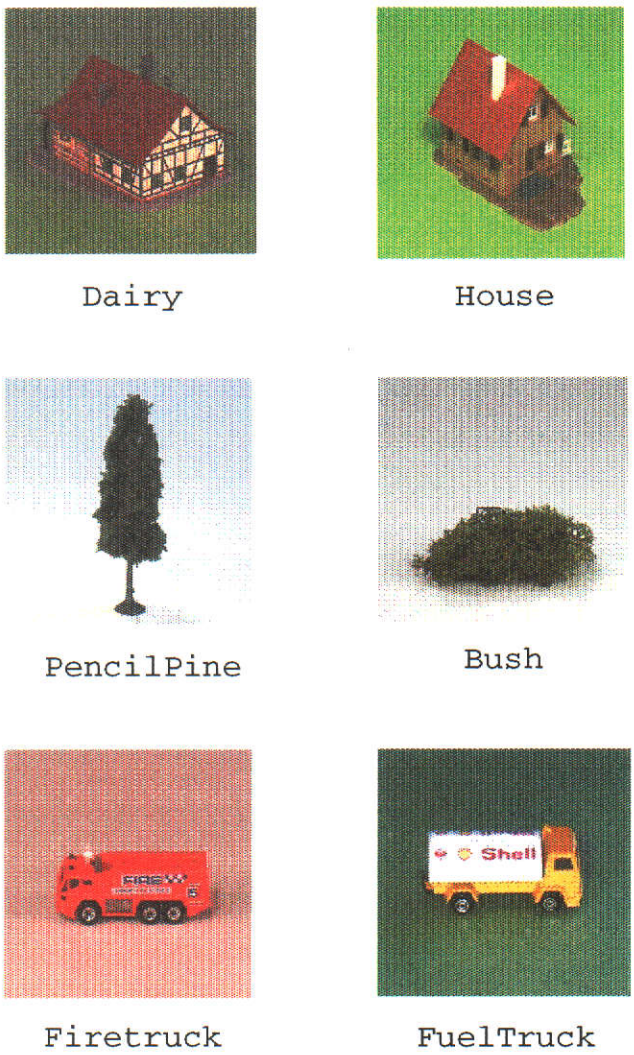


Figure 11.2: **Sample of Images used to Build Street Scene Knowledge Base**

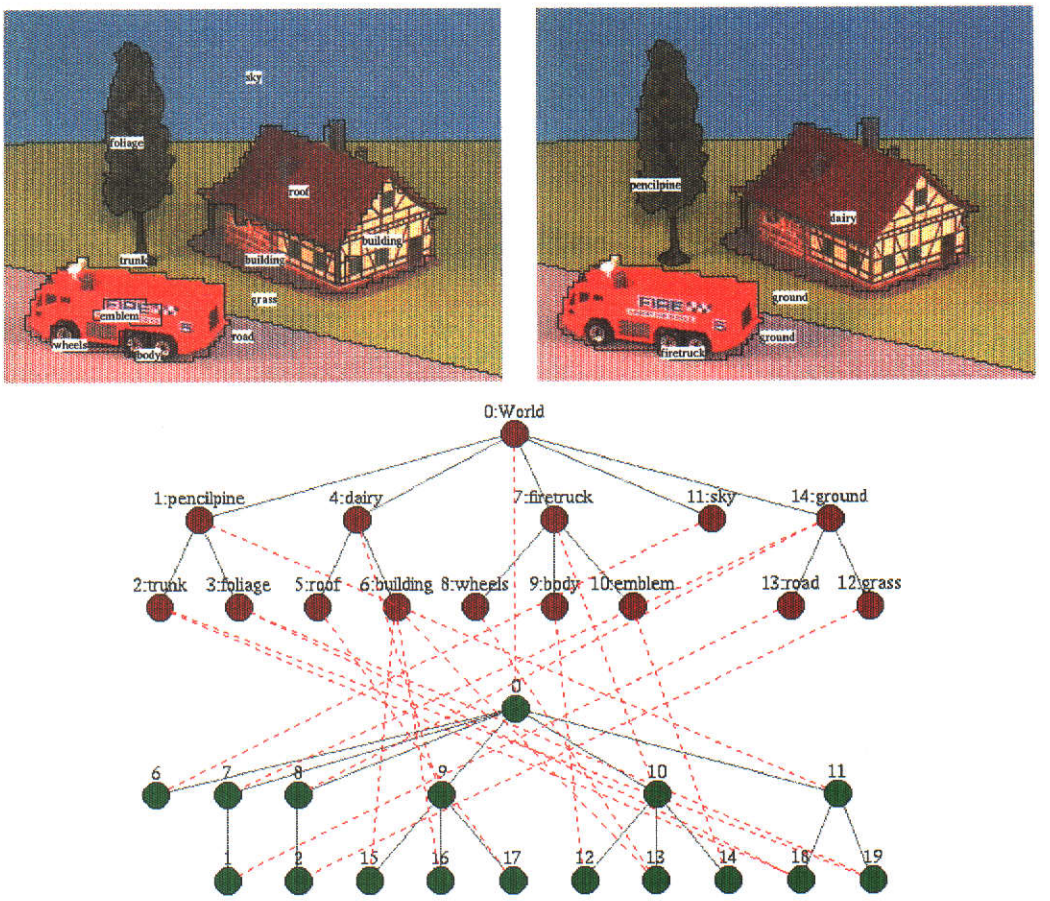


Figure 11.3: **Images and Analysis Graph of Simple Street Scene**

World[0] (1.000) Consisting of:
- sky[6] (1.000)
- ground[7] (1.000) Of Type:
  - road[1] (1.000)
- ground[8] (1.000) Of Type:
  - grass[2] (1.000)
- dairy[9] (1.000) Constructed from:
  - building[15] (1.000)
  - building[16] (1.000)
  - roof[17] (1.000)
- firetruck[10] (1.000) Constructed from:
  - body[12] (1.000)
  - wheels[13] (1.000)
  - emblem[14] (1.000)
- pencilpine[11] (1.000) Constructed from:
  - trunk[18] (0.981)
  - foliage[19] (0.981)

Figure 11.4: **Text Description of Street Scene**

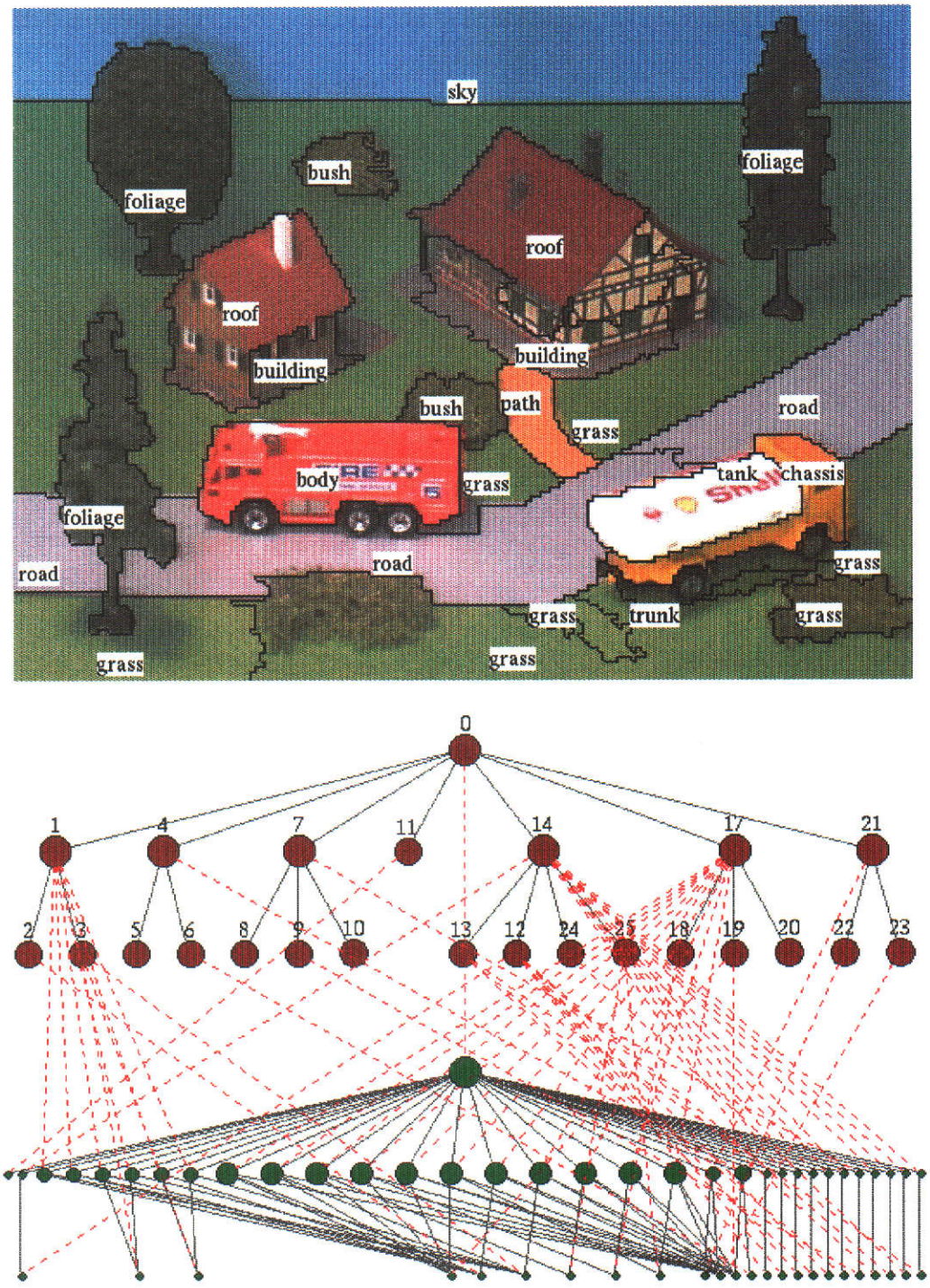


Figure 11.5: **Images and Analysis Graph of Complex Street Scene**

In the more complex street scene shown in Figure 11.5 a total of 24 scene elements have been detected. Of these, one is a mis-classification and three are the result of spurious segmentations. One of the spurious segmentations, the shadow under the fuel-truck,

generates an incorrect "trunk" object. The other two are under and over segmentations, but are correctly labelled as "grass'. Figure 11.6 shows the full text description generated by *Cite*, including the groupings of the low level parts into constituent objects.

```
World[0] (1.000) Consisting of:
├─ sky[25] (0.876)
├─ ground[26] (1.000) Of Type:
│  └─ grass[1] (0.825)
├─ pencilpine[27] (1.000) Constructed from:
│  ├─ foliage[9] (0.932)
│  └─ trunk[5] (0.878)
├─ pencilpine[29] (1.000) Constructed from:
│  └─ foliage[22] (0.922)
├─ pencilpine[31] (1.000) Constructed from:
│  └─ foliage[23] (0.939)
├─ house[36] (0.666) Constructed from:
│  ├─ building[19] (1.000)
│  └─ roof[18] (1.000)
├─ fueltruck[44] (1.000) Constructed from:
│  ├─ chassis[11] (1.000)
│  └─ tank[10] (0.920)
├─ dairy[45] (1.000) Constructed from:
│  ├─ roof[21] (0.934)
│  └─ building[20] (1.000)
├─ ground[46] (1.000) Of Type:
│  └─ grass[2] (0.868)
├─ ground[47] (1.000) Of Type:
│  └─ grass[3] (0.848)
├─ ground[48] (1.000) Of Type:
│  └─ grass[4] (0.563)
├─ ground[49] (1.000) Of Type:
│  └─ road[6] (1.000)
├─ ground[50] (1.000) Of Type:
│  └─ grass[7] (0.687)
├─ ground[51] (1.000) Of Type:
│  └─ road[8] (0.830)
├─ firetruck[52] (1.000) Constructed from:
│  └─ body[12] (0.842)
├─ ground[53] (1.000) Of Type:
│  └─ grass[13] (1.000)
├─ ground[54] (1.000) Of Type:
│  └─ grass[14] (0.716)
├─ ground[55] (1.000) Of Type:
│  └─ path[15] (1.000)
└─ ground[56] (1.000) Of Type:
   └─ road[16] (1.000)
```

Figure 11.6: **Text Description of Street Scene**

11.2 Views of Office Objects

This section illustrates *Cite* in operation recognising small collections of objects from a database of twenty objects arranged in a hierarchy containing sixty four elements. Examples of the isolated objects as used in training are shown in Figure 11.7.

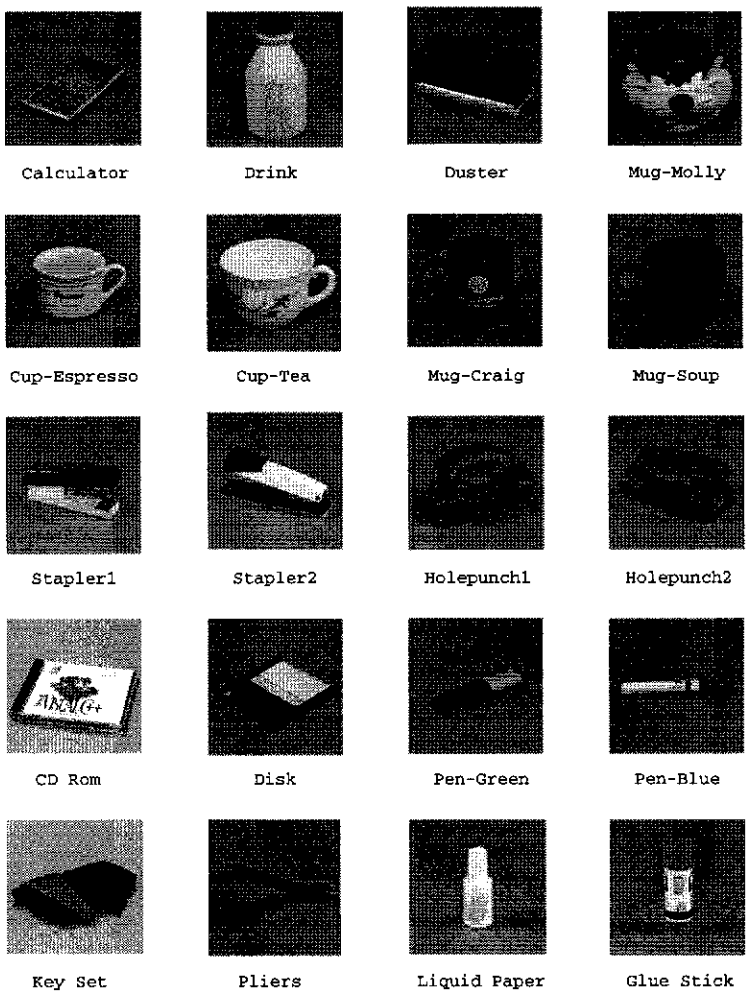


Figure 11.7: **Sample of Images used in Office Knowledge Base**

The office knowledge base as seen in the graph display window in *Cite* is shown in Figure 11.8 with some nodes labelled. The full knowledge base is listed in an expanded text form in Figure 11.9.

Following Figure 11.9 are five pages showing the results for the system recognising small collections of partially overlapping objects. The first and third (Figures 11.10

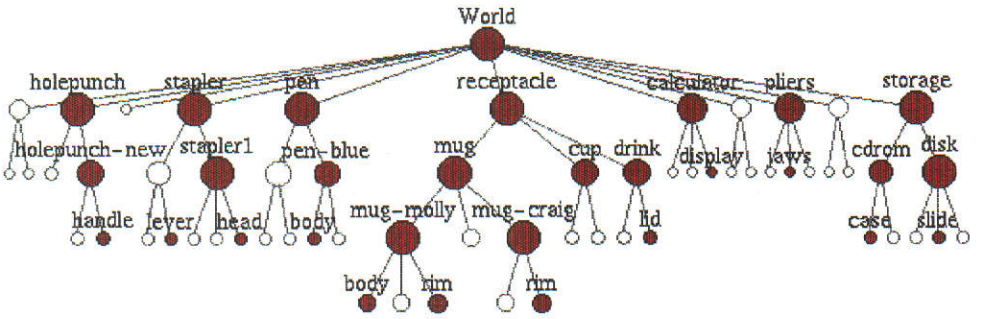


Figure 11.8: **Office Knowledge Base**

and 11.14) show perfect recognition and scene decomposition. Figure 11.12 shows the background seen through the handle of the coffee mug generating an incorrect "holepunch-old" object because there is no background object in the knowledge base. The actual objects in the scene have been correctly recognised.

Four of five objects in Figure 11.16 are correctly recognised, with the tea-cup being mis-matched for part of a type of mug. Figure 11.18 illustrates a situation where the initial segmentation has over-segmented the image and produced an incorrect object instance (the "slide"). These two examples can both be corrected with subsequent incremental supervised learning, but have been included to demonstrate the possible failure modes of *Cite*.

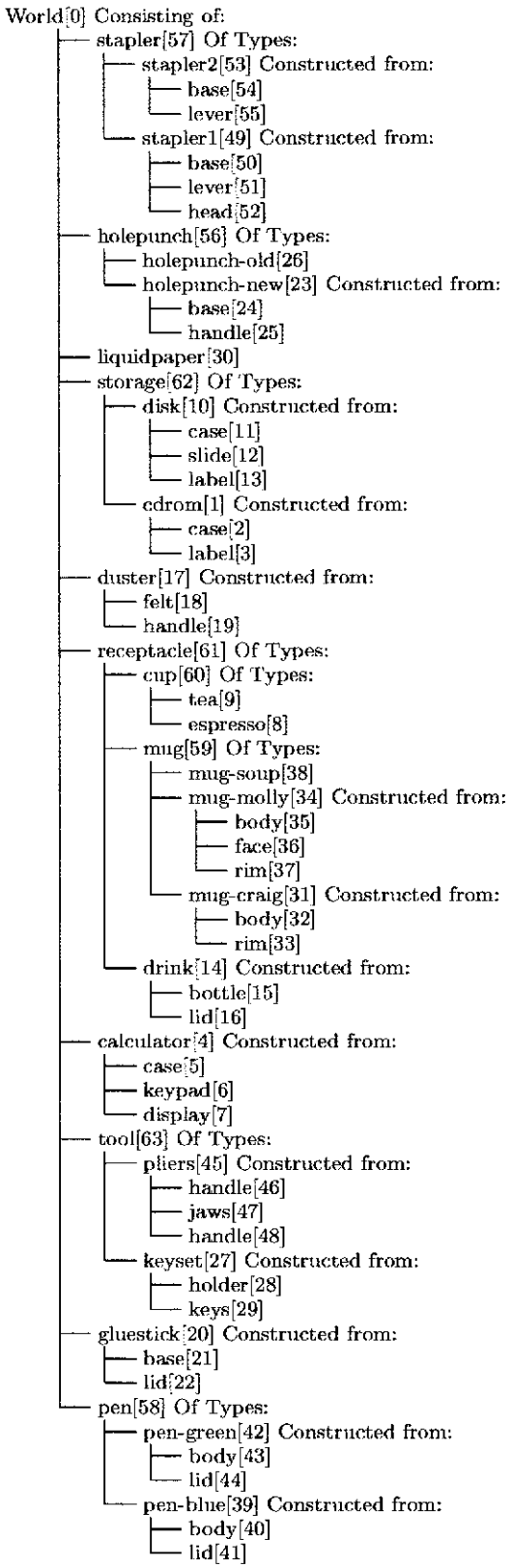


Figure 11.9: **Text Description of Office Knowledge Base**

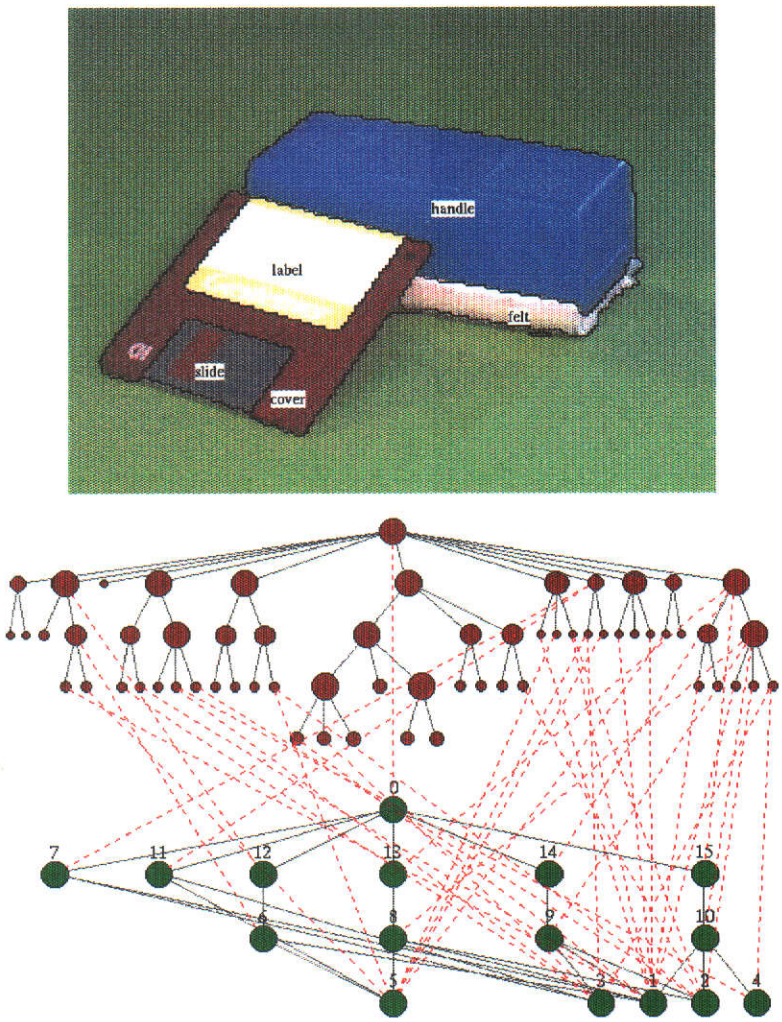


Figure 11.10: **Disk and Duster Image and Analysis Graph**

```
World[0] (1.000) Consisting of:
├─ duster[11] (1.000) Constructed from:
│   ├─ felt[3] (1.000)
│   └─ handle[5] (0.956)
└─ storage[15] (1.000) Of Type:
    └─ disk[10] (1.000) Constructed from:
        ├─ cover[1] (1.000)
        ├─ slide[2] (1.000)
        └─ label[4] (1.000)
```

Figure 11.11: **Text Description of Disk and Duster Scene**

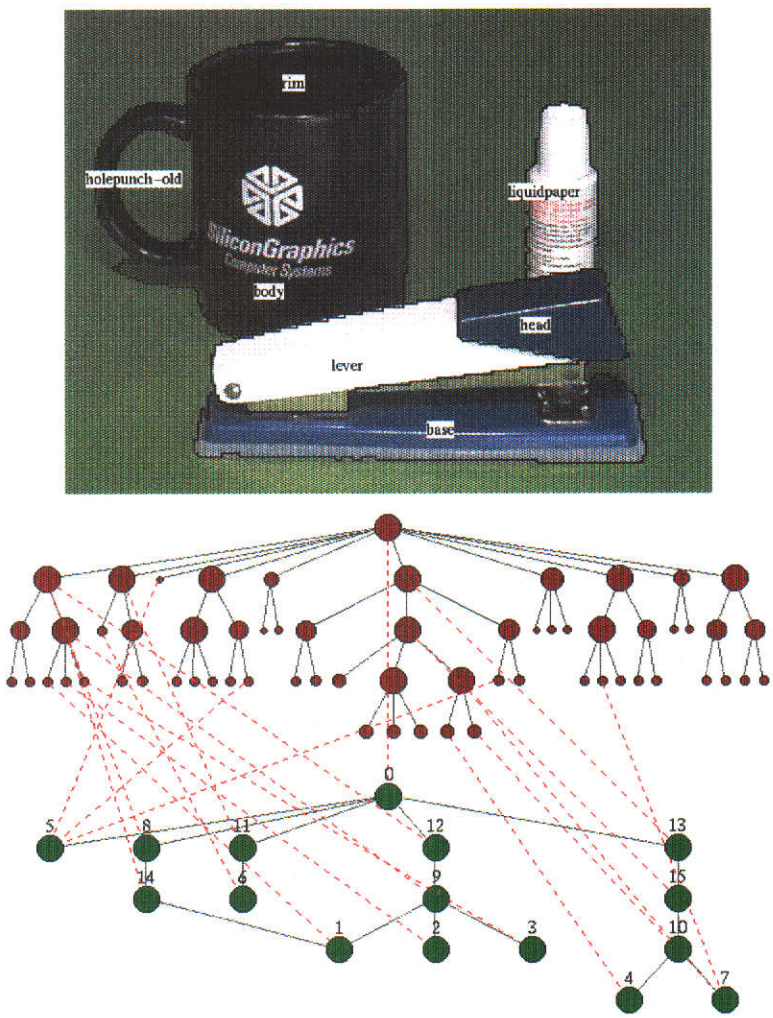


Figure 11.12: **Mug, Stapler and LiquidPaper Image and Analysis Graph**

```
World[0] (1.000) Consisting of:
├── liquidpaper[5] (0.962)
├── holepunch[11] (1.000) Of Type:
│   └── holepunch-old[6] (1.000)
├── stapler[12] (1.000) Of Type:
│   └── stapler1[9] (1.000) Constructed from:
│       ├── base[1] (1.000)
│       ├── lever[2] (1.000)
│       └── head[3] (1.000)
└── receptacle[13] (1.000) Of Type:
    └── mug[15] (1.000) Of Type:
        └── mug-craig[10] (1.000) Constructed from:
            ├── body[4] (1.000)
            └── rim[7] (1.000)
```

Figure 11.13: **Text Description of Mug, Stapler and LiquidPaper Scene**

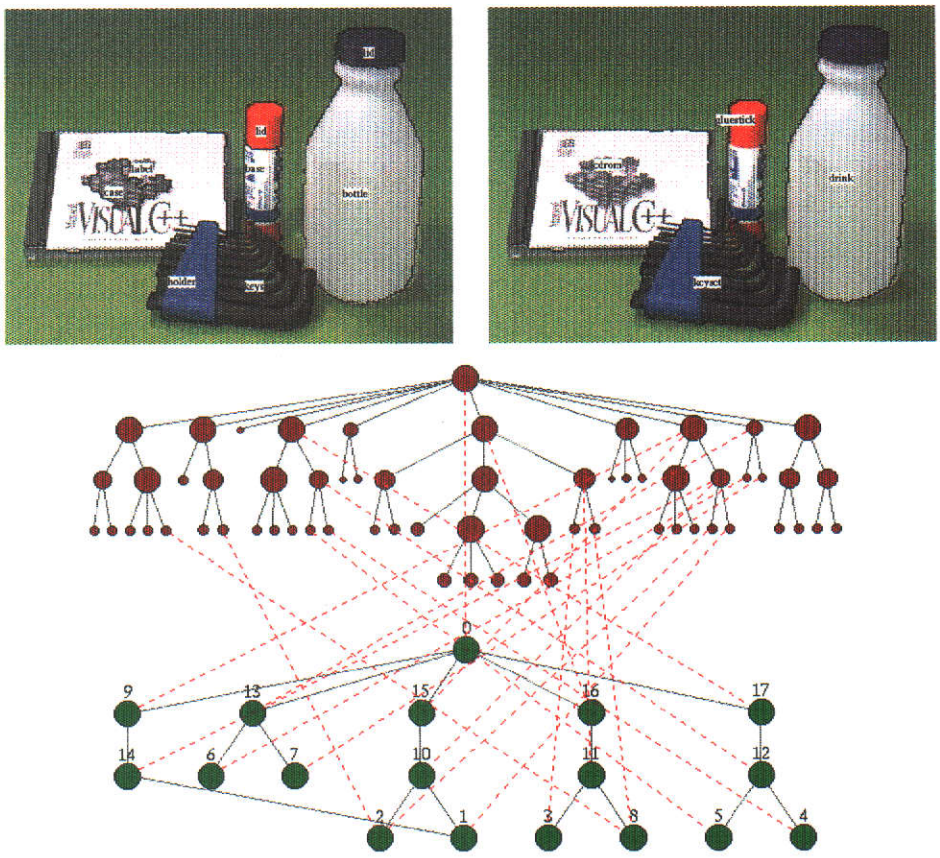


Figure 11.14: **CD, Keyset, Gluestick and Drink Scene**

```
World[0] (1.000) Consisting of:
├─ gluestick[13] (1.000) Constructed from:
│   ├─ base[6] (1.000)
│   └─ lid[7] (0.831)
├─ tool[15] (1.000) Of Type:
│   └─ keyset[10] (1.000) Constructed from:
│       ├─ holder[2] (1.000)
│       └─ keys[1] (1.000)
├─ receptacle[16] (1.000) Of Type:
│   └─ drink[11] (0.896) Constructed from:
│       ├─ bottle[3] (0.906)
│       └─ lid[8] (0.891)
└─ storage[17] (1.000) Of Type:
    └─ cdrom[12] (0.545) Constructed from:
        ├─ case[5] (0.725)
        └─ label[4] (0.706)
```

Figure 11.15: **Text Description of CD, Keyset, Gluestick and Drink Scene**

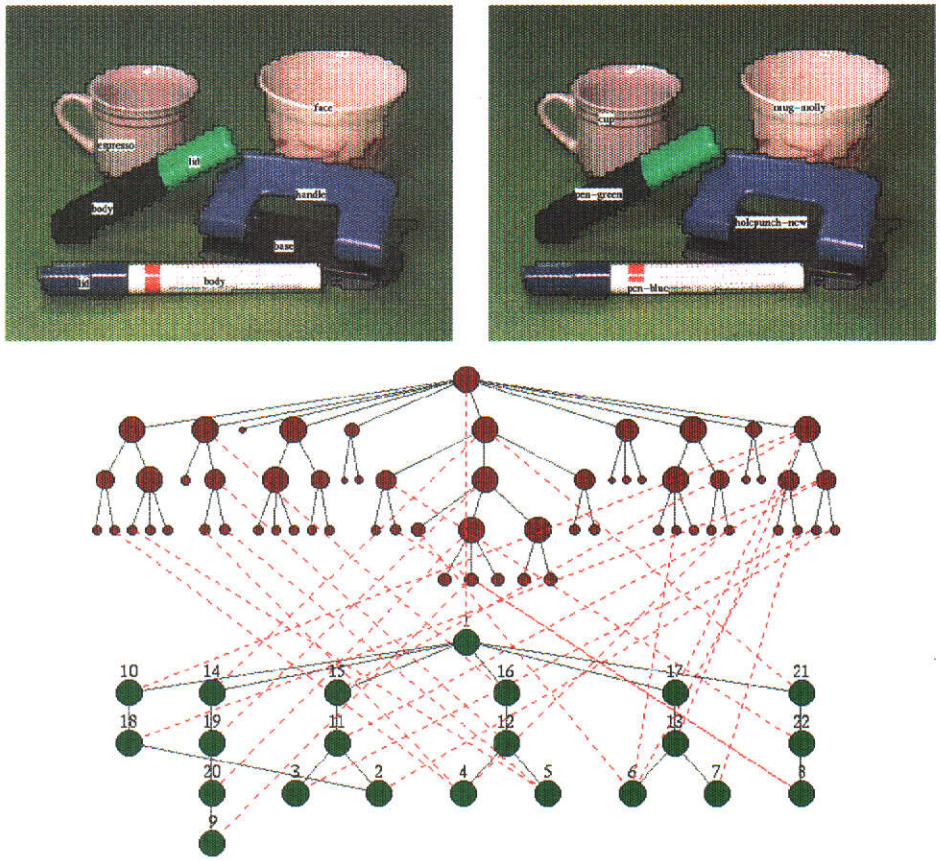


Figure 11.16: **Two Pens, Two Cups and Holepunch Scene**

```
World[1] (1.000) Consisting of:
├── receptacle[14] (1.000) Of Type:
│   └── mug[19] (1.000) Of Type:
│       └── mug-molly[20] (1.000) Constructed from:
│           └── face[9] (1.000)
├── pen[15] (1.000) Of Type:
│   └── pen-blue[11] (1.000) Constructed from:
│       ├── body[3] (1.000)
│       └── lid[2] (1.000)
├── holepunch[16] (1.000) Of Type:
│   └── holepunch-new[12] (1.000) Constructed from:
│       ├── base[4] (1.000)
│       └── handle[5] (1.000)
├── pen[17] (1.000) Of Type:
│   └── pen-green[13] (1.000) Constructed from:
│       ├── body[6] (1.000)
│       └── lid[7] (1.000)
└── receptacle[21] (1.000) Of Type:
    └── cup[22] (1.000) Of Type:
        └── espresso[8] (1.000)
```

Figure 11.17: **Text Description of Pens, Cups and Holepunch Scene**

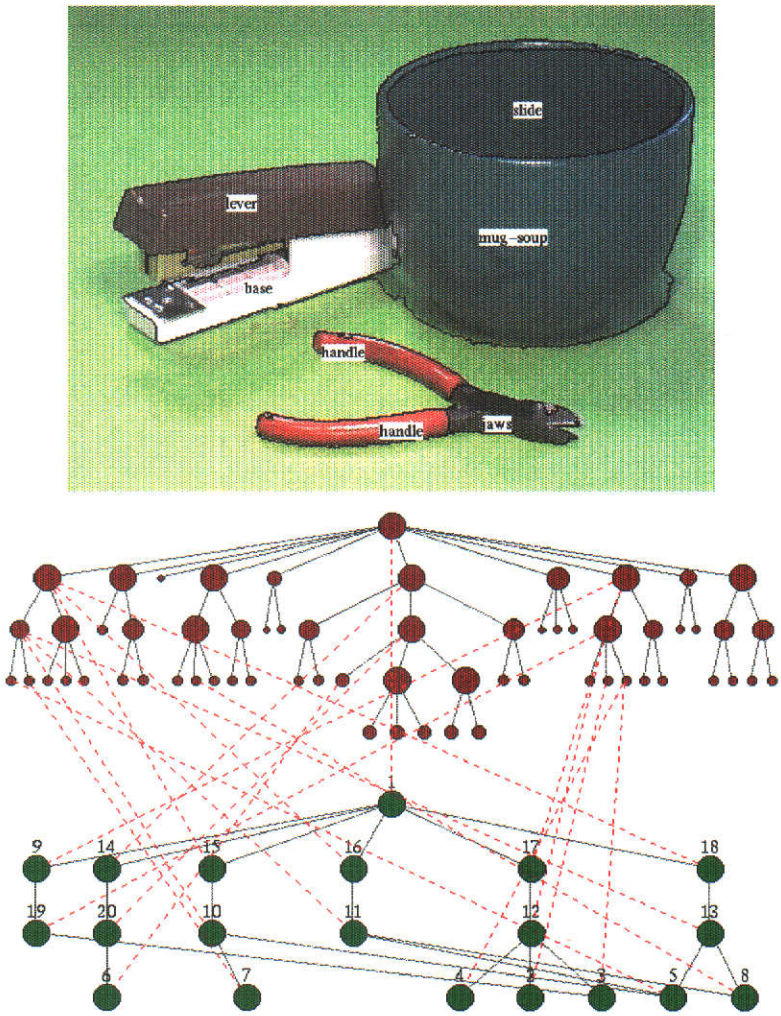


Figure 11.18: **Stapler, Mug and Pliers Scene**

```
World[1] (1.000) Consisting of:
├── receptacle[14] (1.000) Of Type:
│   └── mug[20] (1.000) Of Type:
│       └── mug-soup[6] (1.000)
├── stapler[15] (1.000) Of Type:
│   └── stapler2[10] (1.000) Constructed from:
│       ├── base[5] (1.000)
│       └── lever[7] (1.000)
├── slide[8] (0.638)
└── tool[17] (1.000) Of Type:
    └── pliers[12] (0.666) Constructed from:
        ├── handle[4] (0.558)
        ├── jaws[2] (1.000)
        └── handle[3] (1.000)
```

Figure 11.19: **Text Description of Stapler, Mug and Pliers Scene**

11.3 Aerial Views of Airports

The objective of this section is to demonstrate the operation of the clique resolving process and the hierarchical relaxation labelling as a constraint propagation mechanism. To demonstrate these features of *Cite*, seven aerial images of an airport are analysed to determine not only the labelling of the constituent parts, but the local configurations that they are in. These local configurations have been chosen to heavily overlap so that the system must rely on relational information to resolve ambiguities.

The full knowledge base in shown in Figure 11.20. This was constructed by learning each object from a single image, followed by learning each higher level configuration from full scene examples. As can be seen, the "plane" object plays a central role, being involved in five higher level objects; the "loading", the "emergency", the "service", the "landing" and the "refuelling" objects. When a plane is detected, the system must use relational information to determine which of the five possible parent objects the plane is most likely to belong to, and then propagate this through the scene interpretation graph using relaxation labelling. The clique resolving operator generates the most likely set of possibilities, and relaxation labelling resolves these to determine a subset of high compatability.

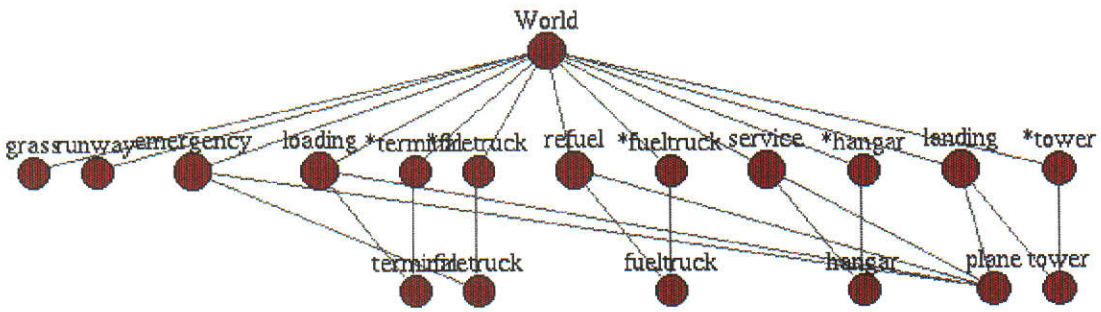


Figure 11.20: **Full Knowledge Base used in Airport Analysis**

A further complication in these scenes is that a number of objects in the database can appear in isolation, or as part of a higher level object. The determination for this in each case rests solely on the strength of the binary matching, and the propagation of the resulting hypotheses through the scene interpretation structure. These nodes are

those whose names are preceeded by an asterisk as described in the Not-Always-Partof section (Section 3.3).

The first five pages show instances of a single "situation" and the recognition results obtained. The last two pages show pairs of situations in which two planes are involved. In these two test, *Cite* must determine not only if a particular situation is occuring, but which objects are involved in that situation. In all cases this is determined solely by the propagation of relational matching through the hierarchical scene description. In all seven test images, *Cite* correctly recognises the various objects and the higher level situations in which they are involved.

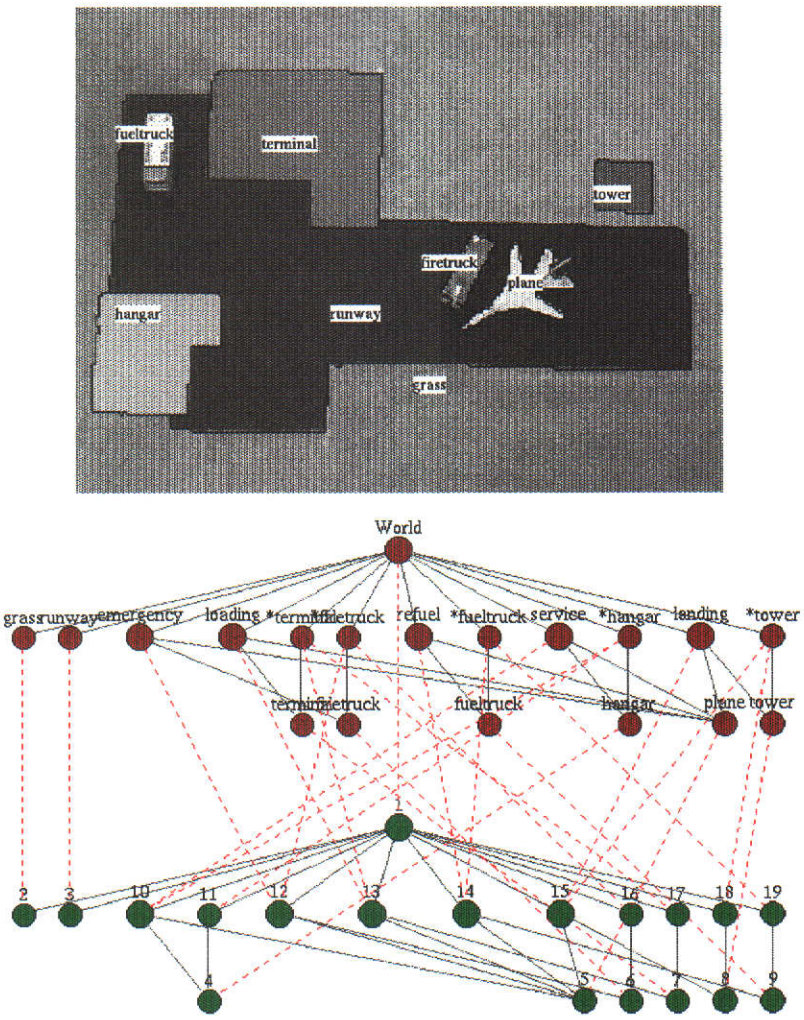


Figure 11.21: **Emergency Image and Analysis Graph**

World[1] (1.000) Consisting of:
- grass[2] (1.000)
- runway[3] (1.000)
- *hangar[11] (1.000)
- emergency[12] (0.946) Constructed from:
  - plane[5] (1.000)
  - firetruck[6] (1.000)
- *terminal[17] (1.000)
- *tower[18] (1.000)
- *fueltruck[19] (1.000)

Figure 11.22: **Text Description of Emergency Scene**

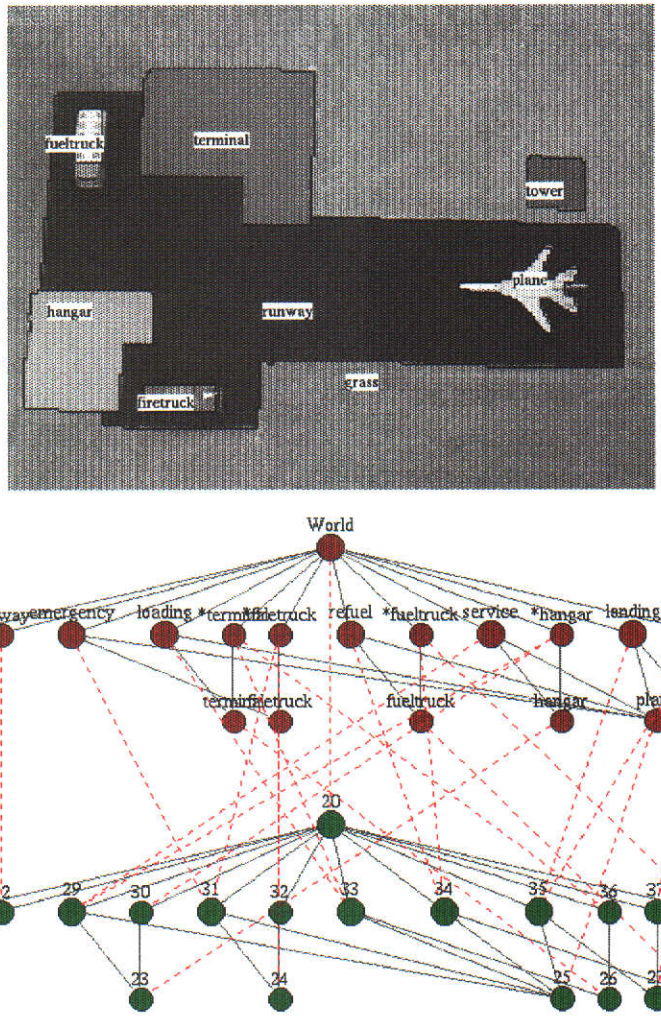


Figure 11.23: **Landing Image and Analysis Graph**

World[20] (1.000) Consisting of:
- grass[21] (1.000)
- runway[22] (1.000)
- *hangar[30] (1.000)
- *firetruck[32] (1.000)
- landing[35] (0.946) Constructed from:
  - tower[27] (1.000)
  - plane[25] (1.000)
- *terminal[36] (1.000)
- *fueltruck[38] (1.000)

Figure 11.24: **Text Description of Landing Scene**

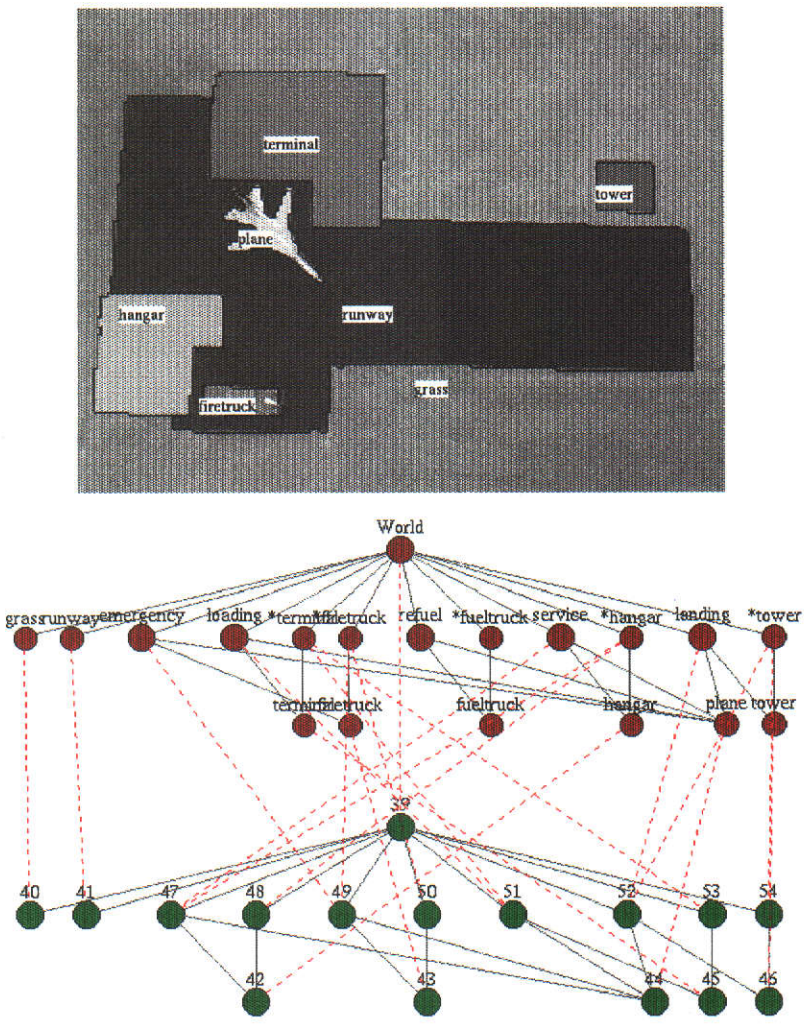


Figure 11.25: **Loading Image and Analysis Graph**

World[39] (1.000) Consisting of:
- grass[40] (1.000)
- runway[41] (1.000)
- *hangar[48] (1.000)
- *firetruck[50] (1.000)
- loading[51] (0.946) Constructed from:
  - plane[44] (1.000)
  - terminal[45] (1.000)
- *tower[54] (1.000)

Figure 11.26: **Text Description of Loading Scene**

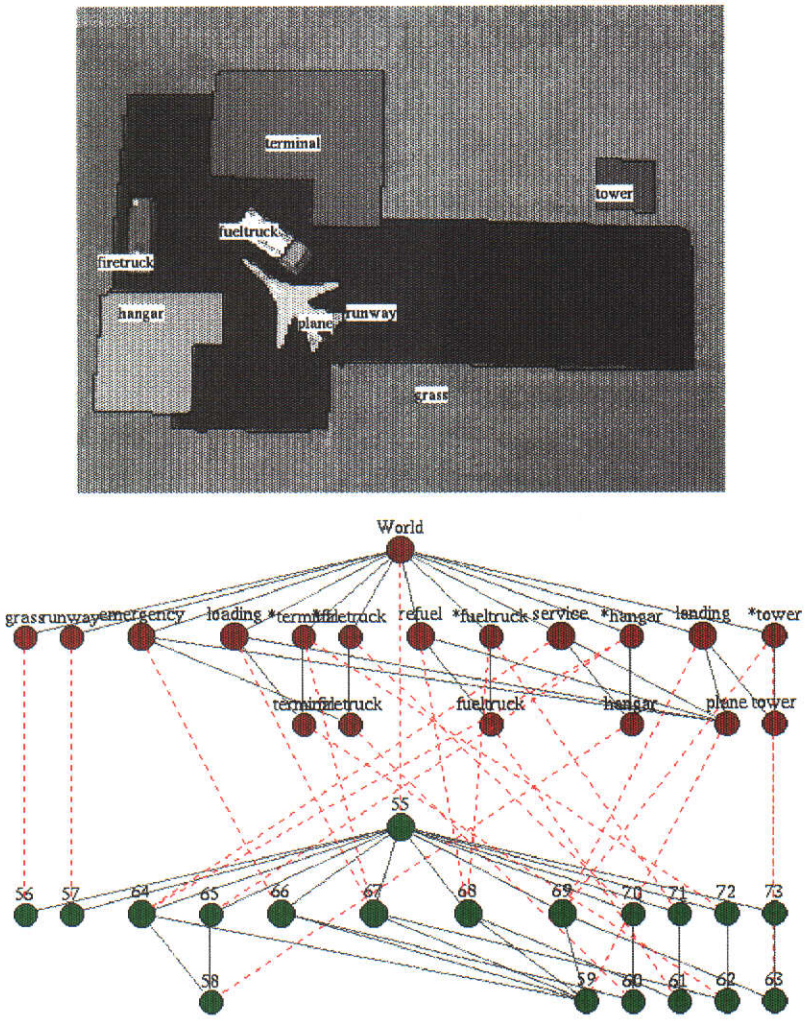


Figure 11.27: **Refuelling Image and Analysis Graph**

World[55] (1.000) Consisting of:
- grass[56] (1.000)
- runway[57] (1.000)
- *hangar[65] (1.000)
- refuel[68] (0.946) Constructed from:
  - fueltruck[61] (1.000)
  - plane[59] (1.000)
- *firetruck[70] (1.000)
- *terminal[72] (1.000)
- *tower[73] (1.000)

Figure 11.28: **Text Description of Refuelling Scene**

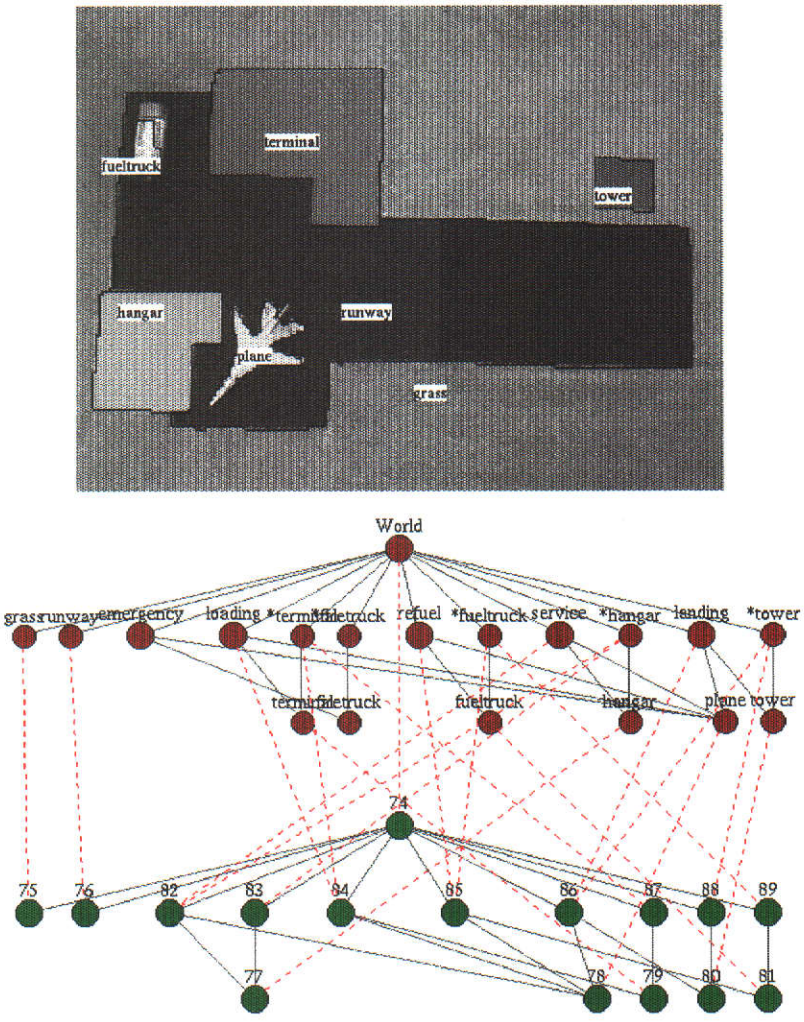


Figure 11.29: **Service Image and Analysis Graph**

World[74] (1.000) Consisting of:
- grass[75] (1.000)
- runway[76] (1.000)
- service[82] (0.946) Constructed from:
  - hangar[77] (1.000)
  - plane[78] (1.000)
- *terminal[87] (1.000)
- *tower[88] (1.000)
- *fueltruck[89] (1.000)

Figure 11.30: **Text Description of Service Scene**

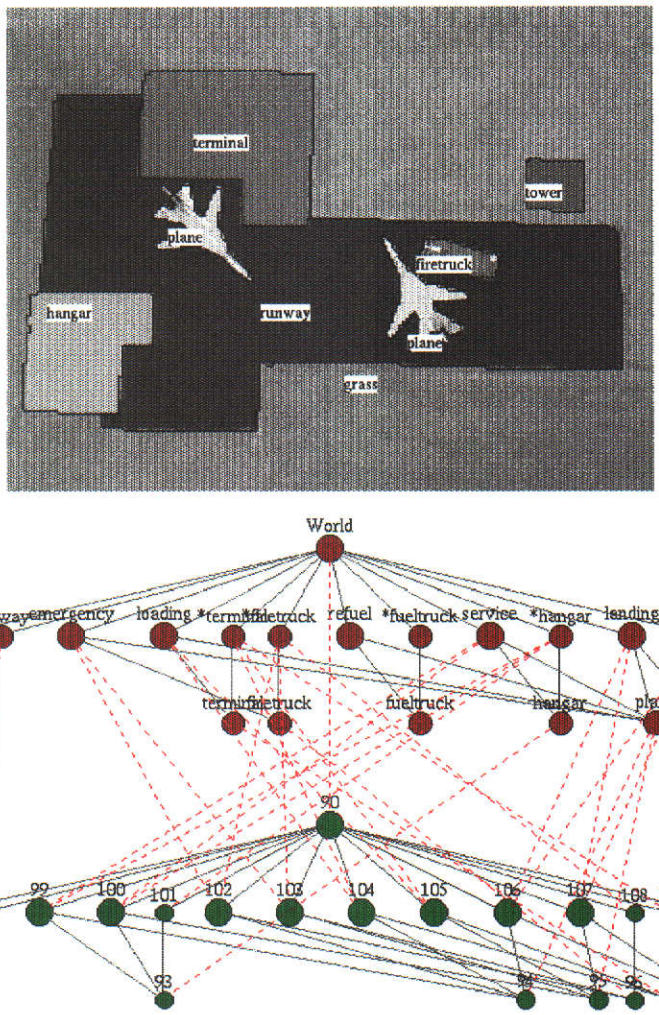


Figure 11.31: **Emergency and Load Image and Analysis Graph**

World[90] (1.000) Consisting of:
- grass[91] (1.000)
- runway[92] (1.000)
- *hangar[101] (1.000)
- emergency[102] (0.946) Constructed from:
  - plane[94] (1.000)
  - firetruck[96] (1.000)
- loading[105] (0.946) Constructed from:
  - plane[95] (1.000)
  - terminal[97] (1.000)
- *tower[110] (1.000)

Figure 11.32: **Text Description of Emergency and Load Scene**

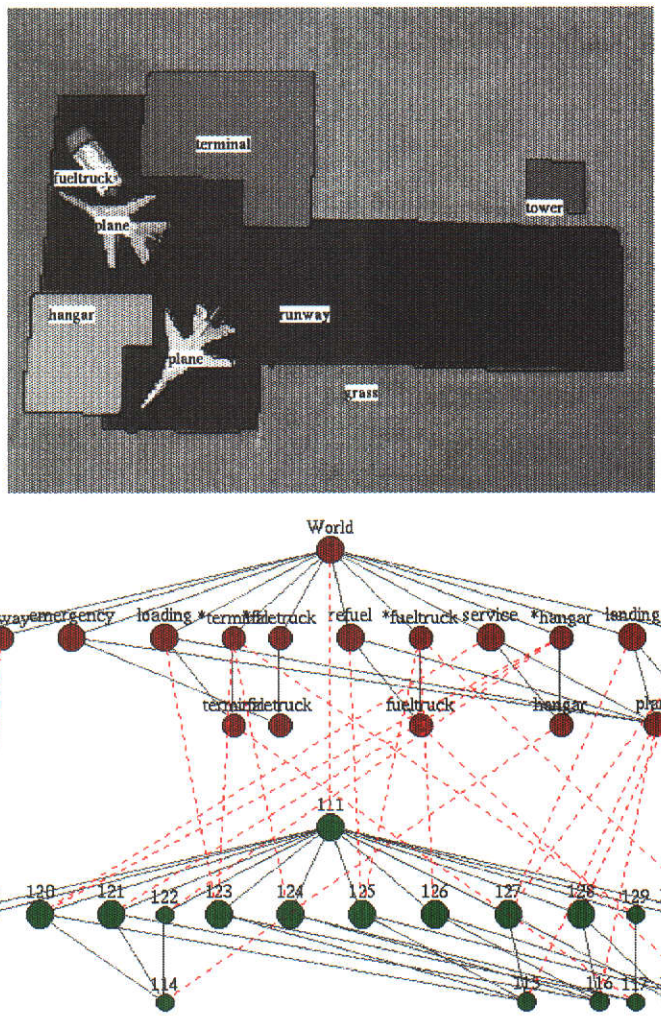


Figure 11.33: **Service and Refuel Image and Analysis Graph**

```
World[111] (1.000) Consisting of:
├── grass[112] (1.000)
├── runway[113] (1.000)
├── *hangar[122] (1.000)
├── emergency[123] (0.946) Constructed from:
│   ├── plane[115] (1.000)
│   └── firetruck[117] (1.000)
├── loading[126] (0.946) Constructed from:
│   ├── plane[116] (1.000)
│   └── terminal[118] (1.000)
└── *tower[131] (1.000)
```

Figure 11.34: **Text Description of Service and Refuel Scene**

11.4 Context Sensitive Taxonomies

To illustrate the context sensitive nature of the knowledge base in *Cite*, three knowledge bases have been built which describe exactly the same objects in different ways. Objects with a high degree of similarity are best suited to demonstrate the sensitivity of different knowledge hierarchies. For this, five tree types and two shrub types have been classified into three taxonomies as might be constructed by a botanist, a forester and a landscape gardener.

Examples of the seven objects are shown in Figure 11.35. Figure 11.36 shows the three taxonomies, and Table 11.2 lists which tree and shrub images are classified into each category in each of the three taxonomies.

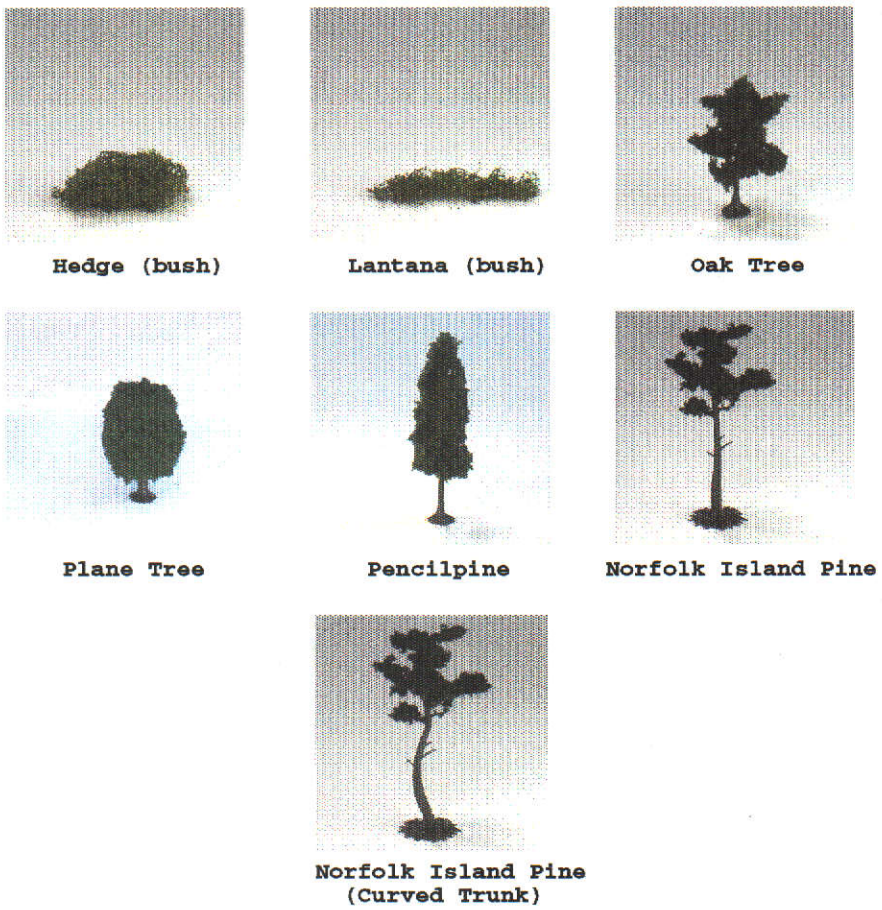
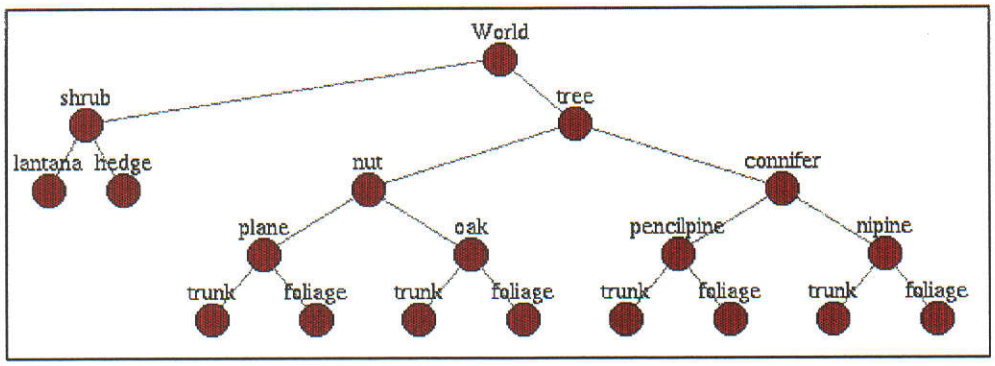
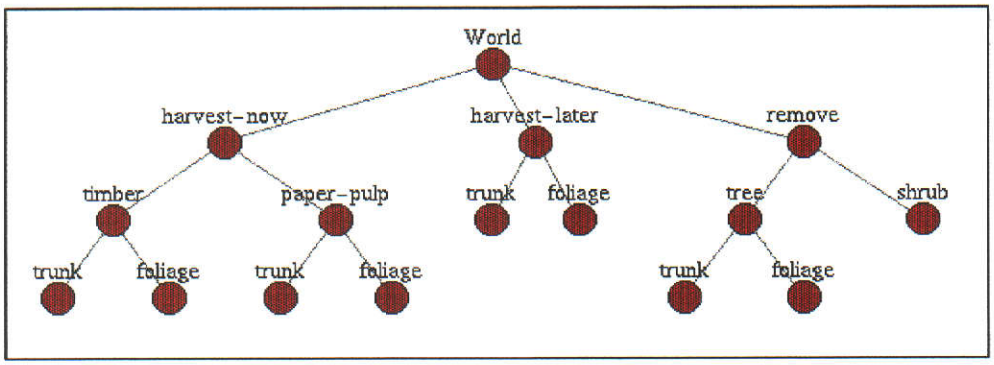


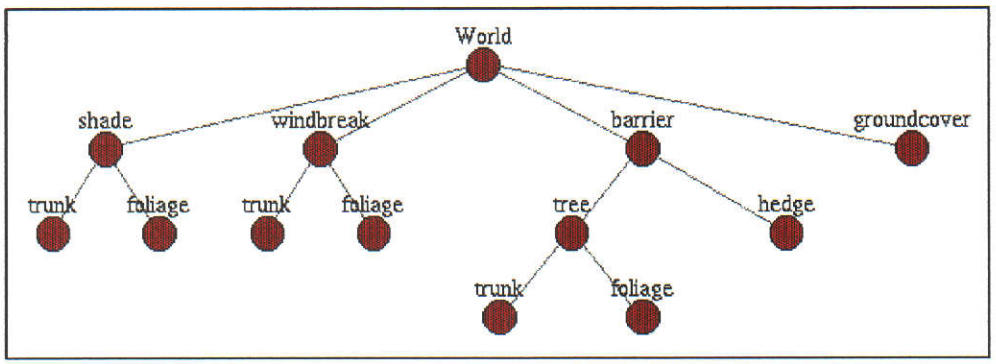
Figure 11.35: **Sample of Images used in Context Sensitive Knowledge Base**



Botanist Knowledge Base



Forester Knowledge Base



Gardener Knowledge Base

Figure 11.36: **Three Context Sensitive Knowledge Bases**

Each of the three taxonomy knowledge bases were tested against a sequence of 24 images, 7 of which had been used in the initial creation of the knowledge base. The results shown in Table 11.3 are examining the higher level classifications of the tree and shrub objects into their respective categories. Of all 72 tests, *Cite* correctly decomposed the

objects into their constituent parts, and was able to combine the unary and relational evidence for correct higher level matching in 67 cases. This represents a %93 correct classification rate when differentiating similar objects based on an externally imposed contextual hierachy.

Image	No.	Botanist	Forrester	Gardener
Plane	8	Plane:Nut:Tree	Tree:Remove	Tree:Barrier
NIPine	4	NIPine:Conifer:Tree	Timber:HarvestNow	Shade
NIPineBent	4	NIPine:Conifer:Tree	Tree:Remove	Shade
PencilPine	8	PencilPine:Conifer:Tree	PaperPulp:HarvestNow	WindBreak
Oak	8	Oak:Nut:Tree	HarvestLater	WindBreak
Hedge	2	Hegde:Shrub	Shrub:Remove	Hedge:Barrier
Lantana	2	Lantana:Shrub	Shrub:Remove	GroundCover

Table 11.2: **Image Categories for Three Different Taxonomies**

Image	No. Tested	Botanist No. Correct	Forrester No. Correct	Gardener No. Correct
Plane	4	4	4	4
NIPine	4	4	4	4
NIPineBent	4	4	3	4
PencilPine	4	4	4	4
Oak	4	4	2	2
Hedge	2	2	2	2
Lantana	2	2	2	2
Total	24	24	21	22

Table 11.3: **Classification Results for Three Different Taxonomies**

Chapter 12

Conclusion

This dissertation has presented a new theory of machine vision which integrates bottom-up and top-down processing to integrate object recognition and scene understanding capabilities within the same framework. The more novel aspects of this dissertation are in the use of hierarchical structures to describe world knowledge in addition to scene and visual decompositions, knowledge driven resegmentation, incremental supervised learning methods, and hierarchical relaxation labelling.

Explanatory least generalisation on decision trees is presented as one method of converting non-incremental learning algorithms into an incremental form that obeys the identified desirable qualities of sub-linear growth in memory and computational complexity. Results are presented illustrating the performance improvement gained by closing the loop on segmentation with the knowledge driven resegmentation algorithm.

This dissertation also describes the *Cite* system which has been built to experimentally test the theories presented. *Cite* provides an ideal test-bed for comparing various segmentation, feature extraction, matching and learning algorithms.

12.1 Directions for Further Research

The *Cite* system can be extended in a number of different ways. The knowledge driven resegmentation could be augmented with a data driven supervised process whereby the user could outline individual regions in the image and the system would search through the available segmentation routines to determine the optimal segmenter for that region. This *supervised segmentation* is itself an extremely complex process, but would be ideal to develop under *Cite*'s operational framework.

The hierarchical knowledge base is potentially sub-optimal in that it reflects directly the user's taxonomic and hierarchic description of the world. One possible extension of the hierarchical construction process is to balance the tree-like components of the knowledge base automatically by clustering children into intermediate nodes such that the classification expectation at each level is as uniform as possible. This would bias the knowledge hierarchy towards optimal search efficiency while maintaining the user's imposed contextual representation.

The process of *tree matching* [Ramesh and Ramakrishnan, 1992] (as opposed to tree search or decision trees) is concerned with matching two trees or subtrees. This process is irrelevant to matching sensed data and decision tree or interpretation tree approaches in conventional non-hierarchical object recognition systems. However, because *Cite* provides both a hierarchical knowledge base and a hierarchical scene description, such tree matching could be valuable in providing a useful lower comparison for the performance of the hierarchical matching strategy.

Bibliography

Adelson, E. and Wang, J. (1992). Single lens stereo with a plenoptic camera, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(2): 99–106.

Bareiss, R., Porter, B. and Weir, C. (1988). Protos: an exemplar-based learning apprentice, *International Journal of Man-Machine Studies* pp. 549–561.

Barrow, H. and Tenenbaum, J. (1981). Interpreting line drawings as three-dimensional surfaces, *Artificial Intelligence* **17**: 75–116.

Barth, E., Caelli, T. and Zetzsche, C. (1993). Image encoding, labeling and reconstruction from differential geometry, *Computer Vision, Graphics and Image Processing* **55**(6): 428–46.

Besl, P. and Jain, R. (1985). Three-dimensional object recognition, *Computing Surveys* **17**(1): 75–145.

Besl, P. and Jain, R. (1986). Invariant surface characteristics for 3d object recognition in range images, *Computer Vision, Graphics and Image Processing* **33**: 33–80.

Bischof, W. F. and Caelli, T. (1994). Learning structural descriptions of patterns: A new technique for conditional clustering and rule generation, *Pattern Recognition* **27**(5): 689–97.

Bolle, R., Califano, A. and Kjeldsen, R. (1992). A complete and extendable approach to visual recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(5): 534–548.

Bolles, R. and Cain, R. (1982). Recognising and locating partially visible objects: The local-feature-focus method, *Robot Vision* pp. 43–82.

Bolles, R. and Horaud, P. (1986). 3DPO: A three-dimensional part orientation system, *International Journal of Robotics Research* **3**(5): 3–26.

Brooks, R. (1981). Symbolic reasoning among 3-d models and 2-d images, *Artificial Intelligence* **17**: 285–349.

Caelli, T. and Drier, A. (1994). Variations on the evidence-based object recognition theme, *Pattern Recogn.* **27**(2): 185–204.

Carbonell, J., Michalski, R. and Mitchell, T. (1983). *An Overview of Machine Learning*, Vol. 1, Morgan Kaufmann, Los Altos, CA, chapter 1, pp. 3–23.

Carpenter, G. A. and Grossberg, S. (1988). The art of adaptive pattern recognition by a self-organizing neural network, *IEEE Computer* **21**(3): 77–88.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. and Freeman, D. (1988). AutoClass: a Bayesian classification system, *Proceedings of the Fifth International Workshop on Machine Learning*, Morgan Kaufman, San Mateo, CA, pp. 296–306.

Connell, J. and Brady, M. (1985). Generating and generalizing models of visual objects, *Massachusetts Institute of Technology, AI Memo 823* .

Davis, L. and Rosenfeld, A. (1981). Cooperating processes for low-level vision: A survey, *Artificial Intelligence* **17**: 245–263.

Dhond, U. and Aggarwal, J. (1991). A cost-benefit analysis of a third camera for stereo correspondence, *International Journal of Computer Vision* **6**(1): 39–58.

Dillencourt, M., Samet, H. and Tamminen, M. (1992). A general approach to connected-component labeling for arbitrary image representations, *Journal of the ACM* **39**(2): 253–280.

Dillon, C. and Caelli, T. (1993). Shape from virtual aperture focus, *Australian and New Zealand Intelligent Information Systems Conference* pp. 382–386.

Draper, B., Collins, T., Brolio, J., Hanson, A. and Riseman, E. (1989). The schema system, *International Journal of Computer Vision* **2**: 209–250.

Duncan, J. and Birkholzer, T. (1992). Reinforcement of linear structure using parameterized relaxation labelling, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(5): 502–515.

Fan, T., Medioni, G. and Navatia, R. (1987). Segmented descriptions of 3-d surfaces, *IEEE Journal of Robotics and Automation* **3**(6): 527–538.

Faugeras, O. and Herbert, M. (1986). The representation, recognition and location of 3-d objects, *International Journal of Robotics Research* **5**(3): 27–52.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering, *Machine Learning* **2**: 173–190.

Flynn, P. and Jain, A. (1991a). BONSAI: 3d object recognition using constrained search, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(10): 1066–1075.

Flynn, P. and Jain, A. (1991b). CAD-based computer vision: From CAD models to relational graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(2): 114–132.

Garvey, T. (1976). An experiment with a system for locating objects in multisensory images, *Int. Joint Conf. on Pattern Recognition* pp. 567–575.

Gennari, J., Langley, P. and Fisher, D. (1989). Models of incremental learning, *Artificial Intelligence* pp. 11–59.

Goodman, A., Haralick, R. and Shapiro, L. (1989). Knowledge-based computer vision, *IEEE Computer* pp. 43–54.

Grimson, E. and Lozano-Perez, T. (1986). Model-based recognition and localization from sparse range data, *Techniques for 3-D Machine Perception* pp. 113–147.

Hansen, C. and Henderson, T. (1989). CAGD-based compter vision, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**(11): 1181–1193.

Haralick, R., Davis, L., Rosenfeld, A. and Milgram, D. (1978). Reduction operations for constraint satisfaction, *Information Science* **14**: 199–219.

Hopcraft, J. and Karp, R. (1973). A $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs, *J. SIAM Comp* **2**: 225–231.

Hummel, R. A. and Zucker, S. W. (1983). On the foundations of relaxation labeling processes, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**(3): 267–287.

Hwang, V., Davis, L. and Matsuyama, T. (1986). Hypothesis integration in image understanding systems, *Computer Vision, Graphics and Image Processing* **36**: 321–371.

Ikeuchi, K. (1987). Precompiling a geometrical model into an interpretation tree for object recognition in bin-picking tasks, *Proc. DARPA Image Understanding Workshop* pp. 321–339.

Jain, A. and Hoffman, R. (1988). Evidence-based recognition of 3-d objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10**(6): 783–802.

Kak, A. and Kim, W. (1991). 3-d object recognition using bipartite matching embedded in discrete relaxation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(3): 224–251.

Kay, G. and Caelli, T. (1994). Inverting the phong lighting model from range and intensity maps, *CVGIP: Image Understanding* **59**(2): 183–201.

Kim, W. and Kak, A. C. (1991). 3-d object recognition using bipartite matching embedded in discrete relaxation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(3): 224–251.

Kohonen, T. (1990). The self-organizing map, *Proceedings of the IEEE*, Vol. 78, pp. 1464–1481.

Kriegman, D. J. and Ponce, J. (1990). On recognizing and positioning curved 3-d objects from image contours, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(12): 1127–1137.

Lowe, D. (1987). Three-dimensional object recognition from single two-dimensional images, *Artificial Intelligence* **31**: 355–395.

Matsuyama, T. and Hwang, V. (1990). *Sigma: A Knowledge-Based Aerial Image Understanding System*, Plenum Press.

McKeown, D., Harvey, W. and McDermott, J. (1985). Rule-based interpretation of aerial images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **7**: 570–585.

Messmer, B. T. and Bunke, H. (1995). Subgraph isomorphism detection in polynomial time on preprocessed model graphs, *in* S. Li, D. Mital, E. Teoh and H. Wan (eds), *Recent Developments in Computer Vision. Second Asian Conference on Computer Vision, ACCV '95, Singapore, Invited Session Papers*, Springer-Verlag, Berlin, Germany, pp. 373–82.

Michalski, R. and Stepp, R. (1983a). Automated construction of classifications: Conceptual clustering versus numerical taxonomy, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**: 396–409. not held.

Michalski, R. and Stepp, R. (1983b). *Learning From Observation: Conceptual Clustering*, Morgan Kaufmann, Los Altos, CA, chapter 11, pp. 331–363.

Murray, D. and Cook, D. (1988). Using the orientation of fragmentary 3d edge segments for polyhedral object recognition, *International Journal of Computer Vision* **2**: 153–169.

Murray, D. W., Castelow, D. A. and Buxton, B. F. (1989). From image sequences to recognized moving polyhedral objects, *International Journal of Computer Vision* **3**: 181–208.

Nalwa, V. (1988). Line-drawing interpretation: Straight lines and conic sections, *T-PAMI* **10**(4): 514–529.

Nazif, A. and Levine, M. (1984). Low level image segmentation: An expert system, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(5): 574–579.

Nishida, H. and Mori, S. (1992). Algebraic description of curve structure, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(5): 516–533.

Oshmia, M. and Shirai, Y. (1983). Object recognition using three-dimensional information, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**(4): 353–361.

Pearce, A. R., Caelli, T. and Bischof, W. F. (1994). Rulegraphs for graph matching in pattern recognition, *Pattern Recognition* **27**(9): 1231–47.

Pelillo, M. and Refice, M. (1994). Learning compatability coefficients for relaxation labelling processes, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(9): 933–945.

Quinlan, J. (1986). Induction of decision trees, *Machine Learning* **1**: 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations, *Machine Learning* **5**: 239–266.

Raja, N. and Jain, A. (1992). Obtaining generic parts from range data using a multiview representation, *SPIE proceedings on Application of Artificial Intelligence: Machine Vision and Robotics* .

Ramesh, R. and Ramakrishnan, I. (1992). Nonlinear pattern matching in trees, *Journal of the ACM* **39**(2): 295–316.

Rosenfeld, A., Hummel, R. and Zucker, S. (1976). Scene labeling by relaxation operations, *IEEE Transactions on Systems, Man and Cybernetics* **6**(6): 420–433.

Schlimmer, J. and Fisher, D. (1986). A case study of incremental concept induction, *Proceedings of the Fifth National Conference on Artificial Intelligence* pp. 496–501.

Seibert, M. and Waxman, A. (1992). Adpative 3-d object recognition from multiple views, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(2): 107–124.

Shapiro, L. and Haralick, R. (1981). Structural descriptions and inexact matching, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **3**(5): 504–519.

Tarjan, R. E. and Trojanowski, A. E. (1977). Finding a maximum independent set, *SIAM Journal of Computing* **6**(3): 537 – 546.

Ting-Jun Fan, G. M. and Navatia, R. (1989). Recognizing 3-d objects using surface descriptions, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**(11): 1140–1157.

Utgoff, P. (1989). Incremental induction of decision trees, *Machine Learning* **4**: 161–186.

Wallace, R. and Kanade, T. (1989). Finding hierarchical clusters by entropy minimization, *Image Understanding Workshop*, pp. 1105–1116.

Wang, C. and Srihari, S. (1988). A framework for object recognition in a visually complex environment and its application to locating address blocks on mail pieces, *International Journal of Computer Vision* **2**: 125–151.

Wang, Y. and Aggarwal, J. (1986). Surface reconstruction and representation of 3-d scenes, *Pattern Recognition* **19**(3): 197–207.

Weng, J., Ahuja, N. and Huang, T. (1993). Learning recognition and segmentation of 3d objects from 2d images, *Proceedings of the Fourth International Conference on Computer Vision* .

Wong, A., Lu, S. and Rioux, M. (1989). Recognition and shape synthesis of 3-d objects based on attributed hypergraphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**(3): 279–290.

Appendix A

Additional Details

A.1 Algorithm Details

This section lists each of the main algorithms used in *Cite*. Reference should be made to the appropriate chapters for a detailed description of each algorithm.

A.1.1 ProcessSIFromKB Algorithm

The algorithm for the PROCESSSIFROMKB top-down process is listed in Algorithm 7. This algorithm is described in detail in Section 7.3.5 and propagates taxonomic and hierarchical decomposition information from the knowledge base to the scene interpretation graph.

```
BESTKB = BESTKBNODE
if BESTKB.TYPE ≠ KBPART then
  for all CHILD ∈ CHILDLIST do
    if CHILD.KBLIST = ∅ then
      CONTINUE
    end if
    if ‖CHILD.KBLIST‖ > 1 then
      if CHILD.NEXTBESTKBRATIO > α_kbratio then
        CONTINUE
      end if
    end if
    CBESTKB = CHILD.BESTKBNODE
    if BESTKB.RELATIONSHIPDISTANCE(CBESTKB) ≤ 1 then
      CONTINUE
    end if
    for all KBCHILD ∈ BESTKB.CHILDLIST do
      if CHILD.ISANCESTOR(KBCHILD) then
        if KBCHILD.TYPE = KBPART then
          FOUND = FALSE
          for all C ∈ CHILDLIST do
            if C = CHILD then
              CONTINUE
            end if
            for all K ∈ C.KBLIST do
              if C.RELATIONSHIP(K) ≠ 0 then
                FOUND = TRUE
              end if
            end for
          end for
          if NOT FOUND then
            if ‖BESTKB.CHILDLIST‖ = 1 then
              CONTINUE
            end if
          else
            if ‖CHILDLIST‖ ≠ 1 ⋃ ‖BESTKB.CHILDLIST‖ = 1 then
              CONTINUE
            end if
          end if
        end if
        NEWSI = NEW SINODE
        NEWSI.ADDCHILD(CHILD)
        REMOVECHILD(CHILD)
        ADDCHILD(NEWSI)
        NEWSI.ADDKBNODE(KBCHILD)
        RETURN
      end if
    end for
  end for
end if
```

Algorithm 7: Algorithm for Generating SI Nodes from the KB Graph

A.1.2 Algorithm for UnaryBounded Learning

The algorithm for the UNARYBOUNDED incremental learning process is listed in Algorithm 8. This algorithm is one of a number that may be instantiated at each node in the knowledge base for generating unary matching hypotheses, and is described in detail in Section 6.5.1.

```
if NEW.TYPE = POSITIVE then
  SLIST = POSITIVELIST
  OLIST = NEGATIVELIST
else
  SLIST = NEGATIVELIST
  OLIST = POSITIVELIST
end if
BEST = ∅
for all I ∈ SLIST do
  TEST = HULL(I + NEW)
  OVERLAP = FALSE
  for all J ∈ OLIST do
    if J.OVERLAP(TEST) then
      OVERLAP = TRUE
    end if
  end for
  if OVERLAP = FALSE then
    if BEST = ∅ then
      BEST = TEST
    else
      if TEST.SIZE() < BEST.SIZE() then
        BEST = TEST
      end if
    end if
  end if
end for
if BEST = ∅ then
  SLIST = SLIST + NEW
else
  SLIST = SLIST + BEST
end if
SLIST.REDUCE()
```

Algorithm 8: Algorithm for UnaryBounded Learning

A.1.3 ProcessVIFromSI Algorithm

The algorithm for the PROCESSVIFROMSI top down hierarchical structure process is listed in Algorithm 9. This algorithm is described in detail in Section 7.3.3.

```
if CHILDLIST = ∅ then
  RETURN
end if
if SILIST ≠ ∅ then
  for all CHILD ∈ CHILDLIST do
    if CHILD.SILIST ≠ ∅ then
      if BESTSINODE.RELATIONSHIPDISTANCE(CHILD.BESTSINODE) > 1 then
        for all PSCHILD ∈ BESTSINODE.CHILDLIST do
          if CHILD.BESTSINODE.ISANCESTOR(PSCHILD) then
            NEWVI = NEW VINODE
            NEWVI.ADDCHILD(CHILD)
            REMOVECHILD(CHILD)
            ADDCHILD(NEWVI)
            NEWVI.ADDSINODE(PSCHILD)
            RETURN
          end if
        end for
      end if
    end if
  end for
end if
```

Algorithm 9: Algorithm for Building Scene Hierarchies

A.1.4 ProcessSIFromVI Algorithm

The algorithm for the PROCESSSIFROMVI bottom-up process is listed in Algorithm 10. This algorithm is described in detail in Section 7.3.4 and propagates structure from the clique grouping or connected components analysis in the visual interpretation structure up to the scene interpretation structure.

A.1.5 Most Common Parent Algorithm

The MOSTCOMMONPARENT algorithm returns the most common parent of a group of nodes within the same graph. This algorithm is run both on the scene interpretation and knowledge base by a number of different operators within *Cite.* The algorithm

```
if SILIST ≠ ∅ then
  RETURN
end if
TEMPSI = ∅
for all CHILD ∈ CHILDSET do
  if CHILD.SILIST = ∅ then
    RETURN
  end if
  TEMPSI = TEMPSI + {CHILD.BESTSINODE}
end for
ATTACHPOINT = MOSTCOMMONPARENT(TEMPSI)
NEWSI = NEW SINODE
for all ITEM ∈ TEMPSI do
  NEWSI.ADDCHILD(ITEM)
  if ATTACHPOINT ∈ ITEM.PARENTLIST then
    ITEM.REMOVEPARENT(ATTACHPOINT)
  end if
end for
ATTACHPOINT.ADDCHILD(NEWSI)
ADDSINODE(NEWSI)
```

Algorithm 10: Algorithm for Generating SI Nodes from VI Nodes

is shown in Algorithm 11 and recursively descends the graph until it finds a node which is not a descendent of all the nodes for which the most common parent is being determined.

```
ARRAY = {NODE}
for all ITEM ∈ ARRAY do
  if NOT DESCENDENT(ITEM) then
    RETURN ∅
  end if
end for

for all CHILD ∈ CHILDLIST do
  RETURN_VALUE = CHILD.MOSTCOMMONPARENT(ARRAY)
  if RETURN_VALUE ≠ ∅ then
    RETURN RETURN_VALUE
  end if
end for

RETURN THIS
```

Algorithm 11: Algorithm for Determining Most Common Parent

A.1.6 ProcessMatchKB Algorithm

The algorithm for the PROCESSMATCHKB top-down process is listed in Algorithm 12. This algorithm is described in detail in Section 7.4.3 and matches nodes in the scene interpretation graph using both unary and binary hypotheses.

```
POSSIBLEKB = ∅
for all CHILD ∈ CHILDLIST do
  for all KB ∈ CHILD.KBLIST do
    for all PKB ∈ KB.PARENTLIST do
      if PKB ∉ POSSIBLEKB then
        POSSIBLEKB = POSSIBLEKB + {PKB}
      end if
    end for
  end for
end for
for all TESTKB ∈ POSSIBLEKB do
  MATCH[TESTKB] = 0
  for all CHILD ∈ CHILDLIST do
    WEIGHT = 0
    MATCHES = 0
    for all KB ∈ CHILD.KBLIST do
      for all PKB ∈ KB.PARENTLIST do
        if PKB = TESTKB then
          MATCHES = MATCHES + 1
          WEIGHT = WEIGHT + CHILD.KBWEIGHT(KB) / CHILD.KBWEIGHTRANK(KB)
        end if
      end for
    end for
    if MATCHES then
      MATCH[TESTKB] = MATCH[TESTKB] + WEIGHT / MATCHES
    end if
  end for
  if ||TESTKB.CHILDLIST|| then
    MATCH[TESTKB] = MATCH[TESTKB] / ||TESTKB.CHILDLIST||
  end if
end for
BEST = MAX(MATCH).INDEX
for all KB ∈ POSSIBLEKB do
  if MATCH[KB] > MATCH[BEST] α_match then
    ADDKBNODE(KB)
  end if
end for
```

Algorithm 12: Algorithm for Matching SI Parent Nodes to the KB Graph

A.2 Operator Early Termination Requirements

A number of the unary hypothesis generation operators in *Cite* operate recursively on the data structures that they modify. This can cause some subtle but potentially disastrous consequences if the operators are not modified for early termination upon making their first change.

For additive unary hypothesis operators, the recursion code is typically something like the following:

```
for(int i=0;i<NumChildren();i++)
{
    GetChild(i)->Operation();
}
...
code to perform operation on current node
```

If unchecked, and the operator makes a number of changes to the structure, this can cause the system to fail if these changes result in changing the parent-child relationships in the structure. This occurs because the recursion code effectively builds a depth first stack on the system stack, which can lose synchronisation with the actual structure. The worse case scenario is when operators remove and delete nodes from the structure which may still be on the system stack as a result of the recursion code.

The solution to this problem is straight forward and simple to implement. In programming these operators which can get into trouble in this way, the function must simply be able to return back up through what is known to be an unmodified structure. This can easily be achieved by returning control back to the process scheduler as soon as the first change is made. Because every operator in *Cite* scans the structures looking for processing that needs to be done, every part of the structure will be processed. Only few operators require this attention, and the slight performance penalty is not significant.

A.3 Derivation of Vertical Straightness

Working first with horizontal straightness, a quadratic fit to an object may be modelled as $\hat{y}_i = a_1 x_i^2 + a_2 x_i + a_3$. Using a least squared error criterion, the following is to be minimised:

$$\sum_i (a_1 x_i^2 + a_2 x_i + a_3 - y_i)^2 \tag{A.1}$$

Taking the partial derivatives with respect to a_1, a_2 and a_3 and setting these to zero the following three equations are obtained:

$$\sum_i (a_1 x_i^2 + a_2 x_i + a_3 - y_i) x_i^2 = 0 \tag{A.2}$$

$$\sum_i (a_1 x_i^2 + a_2 x_i + a_3 - y_i) x_i = 0 \tag{A.3}$$

$$\sum_i (a_1 x_i^2 + a_2 x_i + a_3 - y_i) = 0 \tag{A.4}$$

Expanding these out yields:

$$a_1 \sum_i x_i^4 + a_2 \sum_i x_i^3 + a_3 \sum_i x_i^2 - \sum_i y_i x_i^2 = 0 \tag{A.5}$$

$$a_1 \sum_i x_i^3 + a_2 \sum_i x_i^2 + a_3 \sum_i x_i - \sum_i y_i x_i = 0 \tag{A.6}$$

$$a_1 \sum_i x_i^2 + a_2 \sum_i x_i + a_3 n - \sum_i y_i = 0 \tag{A.7}$$

Substituting $X_0 = n$, $X_1 = \sum_i x_i$, $X_2 = \sum_i x_i^2$, $X_3 = \sum_i x_i^3$, $X_4 = \sum_i x_i^4$, $Y_1 = \sum_i y_i$, $Y_2 = \sum_i y_i x_i$ and $Y_3 = \sum y_i x_i^2$ the following is obtained:

$$a_1X_4 + a_2X_3 + a_3X_2 - Y_3 = 0 \quad \text{(A.8)}$$

$$a_1X_3 + a_2X_2 + a_3X_1 - Y_2 = 0 \quad \text{(A.9)}$$

$$a_1X_2 + a_2X_1 + a_3X_0 - Y_1 = 0 \quad \text{(A.10)}$$

Solving 3 into 1 and 3 into 2 yields:

$$a_1(X_1X_4 - X_2X_3) + a_3(X_1X_2 - X_0X_3) + X_3Y_1 - X_1Y_3 = 0 \quad \text{(A.11)}$$

$$a_1(X_1X_3 - X_2^2) + a_3(X_1^2 - X_0X_2) + X_2Y_1 - X_1Y_2 = 0 \quad \text{(A.12)}$$

Which when solved gives:

$$a_1 = \frac{(X_2Y_1 - X_1Y_2)(X_1X_2 - X_0X_3) - (X_3Y_1 - X_1Y_3)(X_1^2 - X_0X_2)}{(X_1X_4 - X_2X_3)(X_1^2 - X_0X_2) - (X_1X_3 - X_2^2)(X_1X_2 - X_0X_3)} \quad \text{(A.13)}$$

This then needs to be axis flipped such that it represents the minimisation of the error in $\hat{x} = a_1y^2 + a_2y + a_3$.

After some experimentation, this method of fitting a quadratic to the set of points has a number of failings. Firstly, if a long object has a blob at one end then this blob receives more statistical weight because of the number of pixels present. Secondly, for large blobby objects the calculated straightness varies wildly with the shape and distribution of the region which is not acceptable.

The solution to this is to determine the straightness of the two edges made from the minimum and maximum x value on each line. For objects that are genuinely bent, the two edges will have roughly the same second order term. For blobby objects, the two edges will have large positive and negative values.

The final straightness value is determined by:

$$s_r = (0.001 + a_{1,left})(0.001 + a_{2,right}) \tag{A.14}$$

$$s = \begin{cases} 0 & \text{if } s_r \leq 0 \\ 100\sqrt{s_r} & \text{if } 0 < s_r \leq 10^{-4} \\ 1 & \text{if } s_r > 10^{-4} \end{cases} \tag{A.15}$$

Appendix B

System Operation

B.1 Overview

Cite is implemented as a single X-Windows based Unix program. The graphical user interface uses the Motif widget set, and was developed on a Silicon Graphics Indy using GNU C++. The system dynamically determines whether the live video capture hardware is present, and if so allows the user to grab images directly. However, most of the testing of *Cite* was done using pre-saved images which can be loaded into *Cite* at any point. This appendix describes each of the graphical windows in the *Cite* system, the menus and the user controlled options.

B.2 Operation

Cite may be launched from the command line by typing 'cite' at the Unix prompt without any arguments. The default start-up behaviour is for each main window other than the results window to be displayed. Each of these main windows is described in the following sections.

B.2.1 The Command Window

The command window is the main control window in *Cite*. It lets you enter commands from pull-down menus or from a one line text command prompt. Figure B.1 shows the command window as it normally appears. The top text window contains the status and progress reporting messages from *Cite*. The middle text window is a history of the previous entered commands, and the lower text window is the command prompt. The commands that can be entered at the command prompt are described in Table B.1.

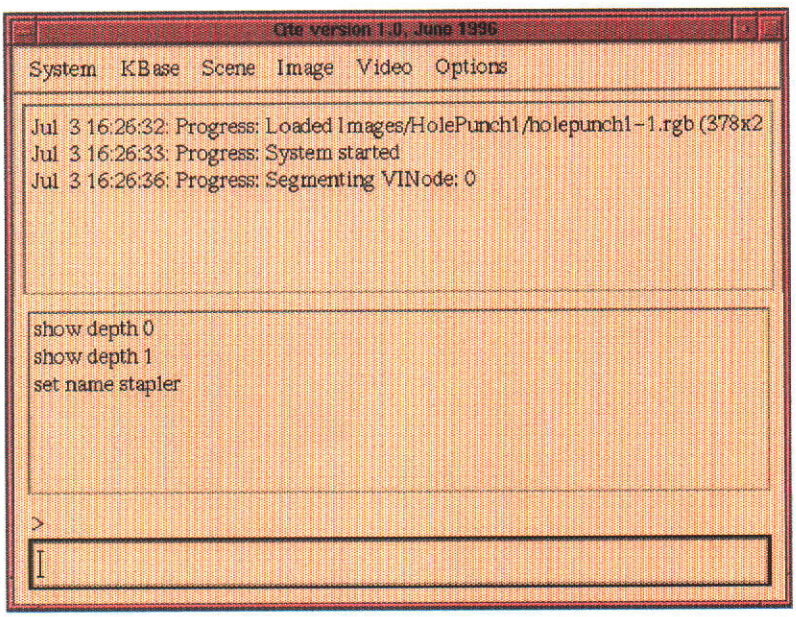


Figure B.1: **The Main Command Window in** *Cite*

Most of the boolean flags that can be set in *Cite* activate one or more of *Cite*'s operators. These are included so that the user can selectively disable an operator, such as the knowledge driven resegmenter, in order to watch the system in operation. *Cite*'s relaxation labelling process is very fast, and it is quite often useful to slow this down using a single-step manual control rather than selecting the continuous update.

Real valued parameters are used mainly by the hypothesis generation and relaxation labelling algorithms. These typically determine how fast hypotheses are updated, how quickly before a resegmentation occurs, and the breadth of hypotheses generated by the matching processes. At the extremes of these parameters the system behaves catas-

trophically - either halting or producing a useless labelling. However, there is a wide operational range around the default values which provides acceptable system operation. These parameters have been placed in the user interface to allow the user to experiment with the stability of the system in different configurations.

Command	Argument	Description
load image	filename	loads an image from disk
load scene	filename	loads the SI from disk
load kbase	filename	loads the KB from disk
save viewset	filename	saves the VI to disk
save scene	filename	saves the SI to disk
save kbase	filename	saves the KB to disk
describe	filename	saves the description to file
show image	integer	shows the n^{th} image
show seg	integer	shows the n^{th} segmentation
show depth	integer	shows segmentation at depth n
quit	n/a	quits *Cite*
new scene	n/a	clears the SI and VI
new kbase	n/a	clears the SI, VI and KB
learn	label	learns selected SI as label
forget	n/a	deletes selected KB nodes
set nobackground	bool	sets background removal
set resegmentation	bool	activates resegmentation
set allchildrenreseg	bool	reseg. even if children match
set autominsize	bool	auto. calculation of minsize
set clique	bool	activate clique resolving
set cliqueconnected	bool	activate connected components
set continuousupdate	bool	continuous weight update
set resegratio	real	ratio before resegmentation
set simatchkbratio	real	ratio of SI in MatchKB
set parentchildratio	real	ratio of child-parent in update
set siunarymatch	real	ratio to best for unary match
set siunarymin	real	minimum absolute unary match
set sigroupsmatch	real	ratio to best in MatchKB
set siparentchildratio	real	parent-child ratio in update
set updateorigmix	real	time-series update ratio
set showweights	bool	display weights on graph window
delete	n/a	delete selected VI/SI nodes
typeof	label	constructs selected KB as typeof label
insert typeof	label	inserts typeof in KB

Table B.1: **Commands in** *Cite*

Under the main menu in the command window are menu options which duplicate the commands that can be entered at the command prompt. In addition, there are menu options for displaying a live video window and for grabbing the live video input and saving it to a file (Silicon Graphics machines only). There is an additional window under the "Options" window which contains a slidebar for each of the real-valued

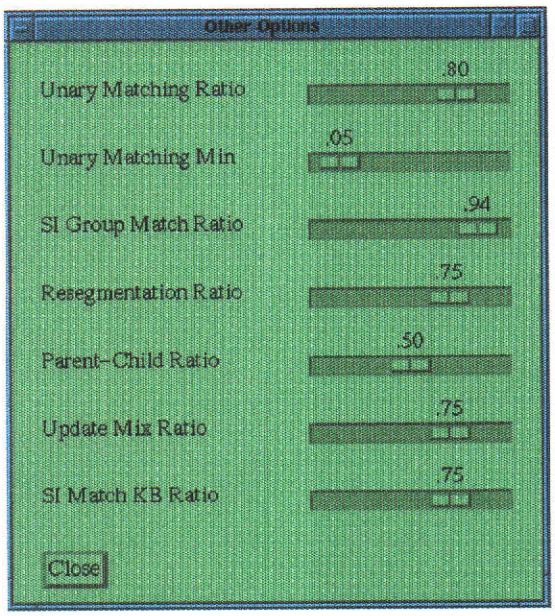


Figure B.2: **The Options Window in** *Cite*

options under the "set" command. This options window is shown in Figure B.2.

B.2.2 The Image Window

The image window contains the images of the current scene and their segmentations. When labelling is complete the labels are placed over the segmented image. The operator can choose to display the segmentation at any level, such as at leaf nodes, at parents of leaf nodes, and so on.

Under the options menu in this window is a toggle for displaying the best hypothesised label on the segment. This defaults to "true", but can be switched off if oversegmentation occurs so that the user can view underlying segments without the clutter of text labels.

B.2.3 The Data Graph Window

The data graph window is the most important visual display in *Cite*. It contains three panels, one above the other. The top panel contains the knowledge base, the middle panel contains the scene interpretation, and the bottom panel contains the visual interpretation for each image.

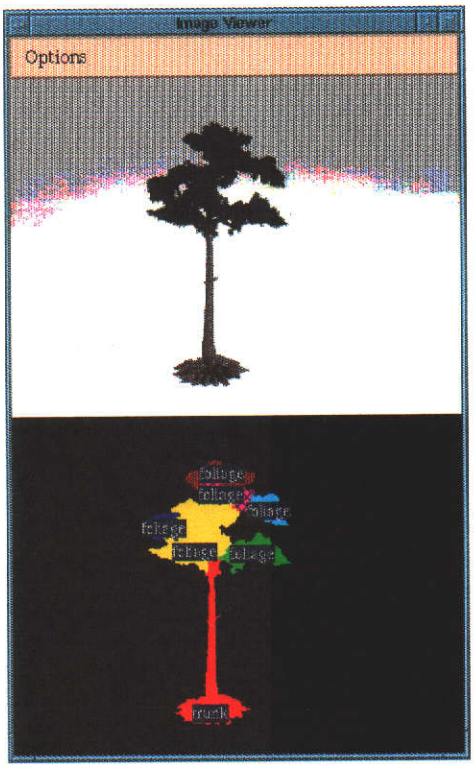


Figure B.3: **The Image and Segmentation Window**

There are many display options which control what is displayed on the screen. This includes displaying hypotheses from the visual interpretations to the scene interpretation, and from the scene interpretation to the knowledge base. The user can also select nodes for editing, grouping or deleting with the mouse. The data graph window is shown in Figure B.4.

The menu options in this window enable various levels of display detail of hypotheses. There is a considerable amount of information in the three graphs represented in this window, and to display all this at once would be virtually impossible. Nodes may be selected by clicking on them with the left mouse button. Clicking on them with the right button brings up the display window for that node. There are three basic types of node display window; one for VI nodes, one for SI nodes and one for KB nodes. An example of each type is shown in Figure B.5.

Selecting a VI node at the same level of display as the segmentation shown in the image window causes that segment to be displayed with a cross-hatch pattern. This is useful

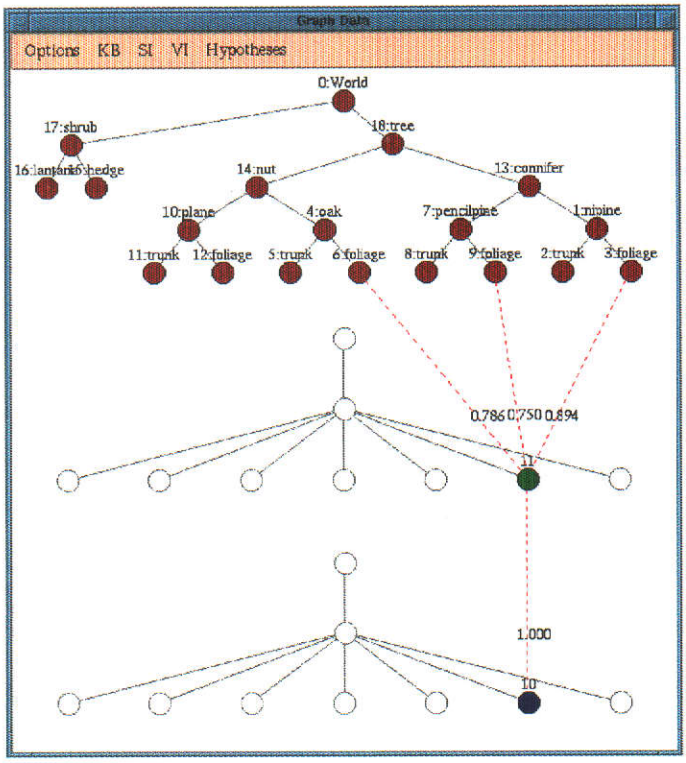


Figure B.4: **The Data Graph Window**

for identifying which region belongs to which VI node. Likewise selecting a region or clicking on the original image causes the region to be displayed with a cross-hatch pattern and the corresponding VI node to become selected.

B.2.4 The Hypothesis Weight Window

A more detailed analysis of the hypothesis weights can be obtained via the hypothesis weight window. This displays a graph of hypothesis weights as a function of time. The weights that are displayed can be individually selected from the node display windows by clicking on the text display of the particular hypothesis of interest. Figure B.6 shows the hypothesis weight window with three SI-KB hypotheses selected. Every hypothesis has a unique integer identifier, and this is displayed on this window as well as the node display window.

The horizontal scale in the hypothesis weight window is the iteration number where

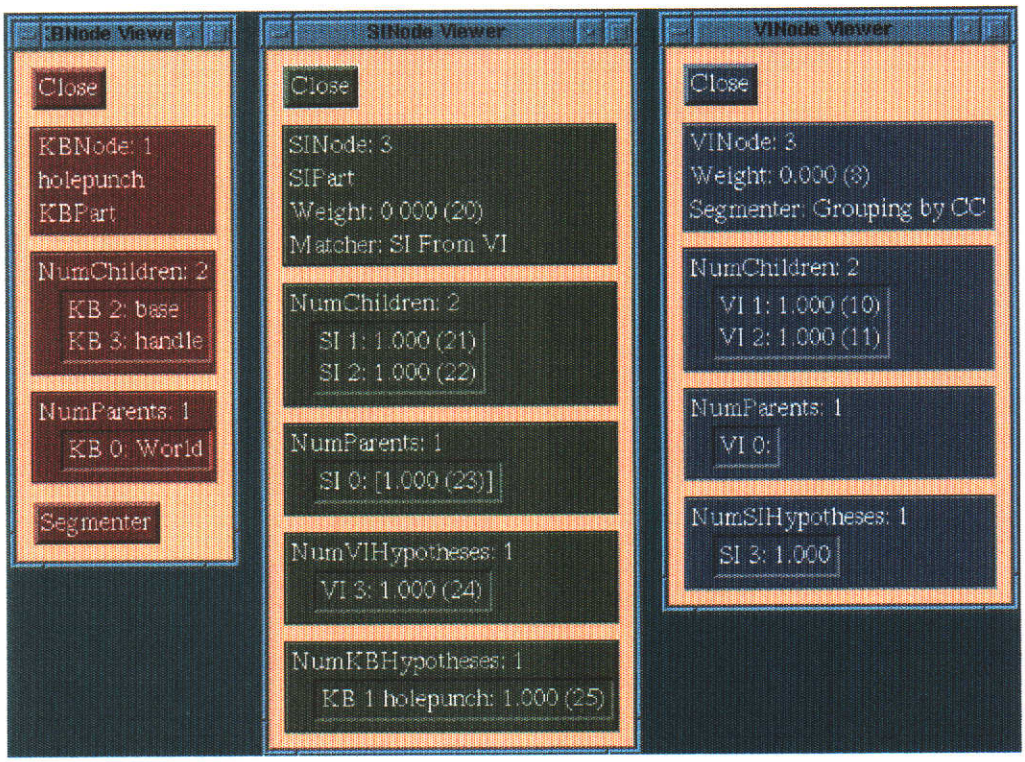


Figure B.5: **Node Display Windows for VI, SI and KB Nodes**

each iteration represents one complete pass through the process stack when a change occurred in at least one hypothesis. That is, a cycle through the process stack is only considered a full iteration if it results in a hypothesis changing its weight. This prevents the system running to infinity with subsequent loss of the interesting parts of the hypothesis curves. The horizontal axis is auto-scaling and the vertical axis ranges between 0.0 and 1.0.

B.2.5 The Profile Window

Cite has a cyclic process scheduler which runs each loaded operator for a short period of time before going to the next. The total amount of time spent in each operator is stored in the operator description class, and displayed in the profile window shown in Figure B.7. The columns in the profile window show the total time spent executing each operator, the overall percentage, a short-term percentage, a long-term percentage, and the absolute time spent in the most previous execution. In addition there is

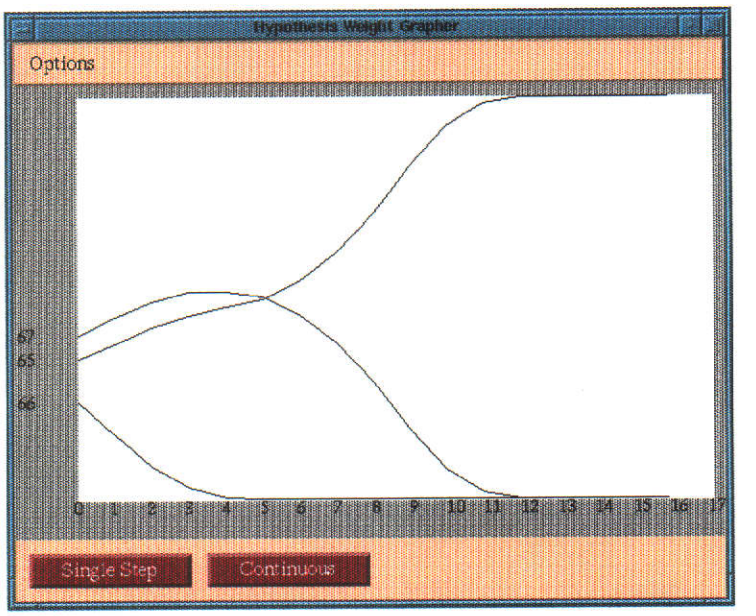


Figure B.6: **The Hypothesis Graphing Window**

a performance measure expressed relative to the original development platform of a 150 MHz R4400 processor. The profiler gives useful information as to the relative computation required for each sub-task within *Cite*.

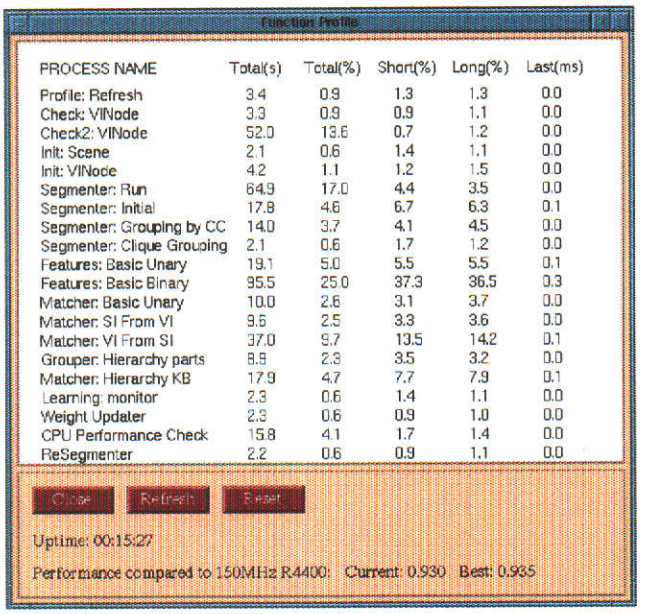


Figure B.7: **The Operator Profile Window**

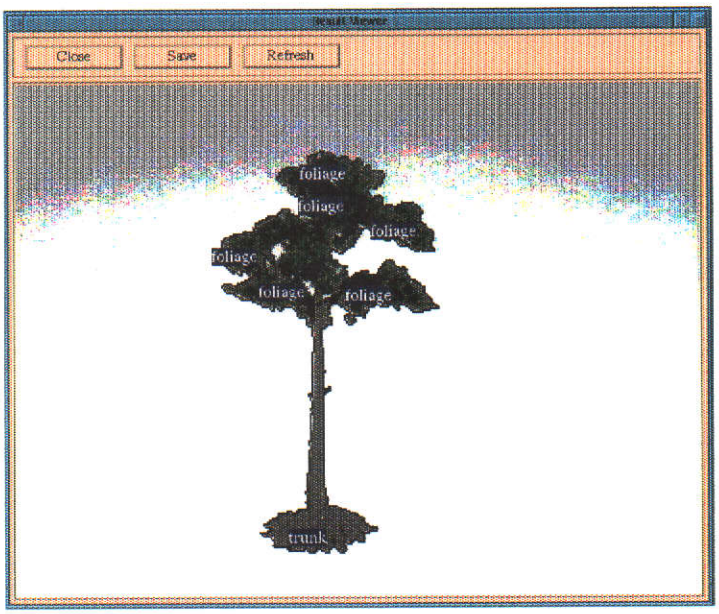


Figure B.8: **The Results Window**

B.2.6 Results Window

In order to produce the highest quality results, there is an additional results window which shows the image in the original size with black-bordered regions and the region labels in a more printable font. The results window is shown in Figure B.8 and is displayed from a menu entry under the main system menu in the command window. The results shown in Chapter 11 were generated using this window.

B.3 *Cite* Files and File Formats

B.3.1 The Log File

Cite generates a log file called *cite.log* which contains time-stamped entries indicating which files were loaded and saved, and any warning or error messages. The log file is appended, so the full operational history of *Cite* can be kept. The following is an example of this log file.

```
Progress [Jul  3 16:21:01] : System started
Progress [Jul  3 16:21:03] : Segmenting VINode: 0
Progress [Jul  3 16:21:27] : Unknown command: swho depth 1
Progress [Jul  3 16:21:33] : quit
Warning  [Jul  3 16:22:00] : Exactly one KB node must be selected to set the name
Progress [Jul  3 16:21:37] : Loaded Images/HolePunch1/holepunch1-1.rgb (378x285)
Progress [Jul  3 16:21:38] : System started
Progress [Jul  3 16:21:42] : Segmenting VINode: 0
Progress [Jul  3 16:24:10] : quit
```

B.3.2 Input Image Format

For historical reasons, the IPRS RGB image format is used for all images loaded from file. This is a binary file format containing a header with pixels stored in correct Cartesian scan format (upside down raster format) at three bytes per pixel. Utilities are available for converting to and from the IPRS image format.

B.3.3 Data Structure File Formats

Each of the knowledge base, scene interpretation and visual interpretation data graphs can be saved to file and loaded from file. These files are stored in a human readable format, but will not be discussed in detail here. In each case, the structures are stored one node per file, with the filename constructed by concatenating the base filename, the node identification number (a unique integer) and one of ".kb", ".si" or ".vi" depending on which structure the node belongs to.

Appendix C

Related Papers Published During PhD Period

During the period of my PhD I was the principal author of the following papers related to image interpretation and machine learning:

Craig Dillon and Terry Caelli, *Infering Shape from Multiple Views using Focus and Correspondence Measures*, TR-91/28 University of Melbourne, November 1991.

Craig Dillon and Terry Caelli, *Interpolating Depth in Passive Sensing*, Digital Image Computing: Techniques and Applications, pp.451-458, December 1991.

Craig Dillon and Terry Caelli, *Generating Complete Depth Maps in Passive Vision Systems*, Proceedings 11th IAPR, pp. A:562-566, Den Hague, The Netherlands, August 1992.

Craig Dillon and Terry Caelli, *Robust Incremental Learning in Pattern Recognition*, First Asian Conference on Computer Vision, pp. 652-655, Osaka, Japan, November 1993.

Craig Dillon and Terry Caelli, *Shape from Virtual Aperture Focus*, Australian and New Zealand Intelligent Information Systems Conference, pp. 382-386, Perth, December 1993.

Craig Dillon and Terry Caelli, *Cite: A Scene Understanding and Object Recognition System*, Western Australia Computer Science Symposium, Perth, September 1994.

Craig Dillon and Terry Caelli, *Cite: A Scene Understanding and Object Recognition System*, Second Asian Conference on Computer Vision, pp. I:214-218 Singapore, December 1995.