Finding Complex Targets in Complex Scenes using Machine Learning Techniques: a viable Surveillance Paradigm?

Terry Caelli[1] Walter F. Bischof[2]

[1]Department of Computer Science, Curtin University of Technology, GPO Box U1987, Perth, WA 6001, Australia, Email: tmc@cs.curtin.edu.au
[2]Department of Psychology, University of Alberta, Edmonton, Alberta, T6G 2E9, Canada, Email: wfb@psych.ualberta.ca

Abstract

In this paper, we discuss automatic rule generation techniques for learning relational properties of 2-D visual patterns and 3-D objects from training samples where the observed feature values are continuous. In particular, we explore a new conditional rule generation method that defines patterns (or objects) in terms of ordered lists of bounds on unary (single part) and binary (part relation) features. The technique, termed Conditional Rule Generation (CRG), was specifically developed to integrate the relational structures of graph representations of patterns and the generalization characteristics of Evidenced-based Systems (EBS). CRG takes into account the label-compatibilities that should occur between unary and binary rules in their very generation, a condition that is, generally, not guaranteed in well-known Rule Generation and Machine Learning techniques as they have been applied to problems in Computer Vision. We show how this technique applies to the recognition of complex targets and of objects in scenes, and we show the extent to which the learned rules can identify patterns and objects that have undergone non-rigid distortions.

1 Introduction

To develop systems which can detect relatively complex patterns or objects in complex scenes requires efficient and robust techniques for describing patterns and searching for them in such data structures. Machine learning, as it applies to the detection, recognition and surveillance of scenes, provides methods for solving such problems. In particular, in this paper we address the issue of just how ML is used in the following sub-system domains of:

- Feature Selection: The automatic selection and/or ordering of encoded features that can optimize the recognition processes.
- Generalization: The automatic generation of "structural descriptions" of targets that can cover a range of training pattern examples, as well as distorted and unseen examples.
- Efficiency: The optimization of search and matching procedures.

These goals can be attained, with differing degrees of success, using a wide variety of representations, learning and matching technologies.

The type of representation most frequently used in vision has been the *relational structure* (RS) where patterns are encoded as parts (graph vertices) and part relations (graph edges), both being described by a set of attributes or features. Such graph representations are limited in the sense that generalization in terms of either new views or non-rigid transformations of objects are difficult to represent. Further, pattern recognition typically involves graph matching, with a computational complexity that exponentiates with the number of parts [1, 2]. Little attention has been paid to the design of optimal search procedures that use conjoint feature states (i.e. conjunctions of particular sets of feature values) to define important characterizations of patterns, and they are less than ideal for the recognition of objects embedded in scenes. Typically, they use prior knowledge to prune the search space, as has been explored by a number of authors (for example, [3, 4]).

In contrast to the RS representation and associated constraint-based graph matching (tree search) methods, evidenced-based systems (EBS) provide a different approach to the recognition problem. Like RS, EBS works within the Supervised Learning (Learning from Example) paradigm and require subprocesses for encoding, segmentation and part/relational feature extraction. Patterns and objects are encoded by rules of the form:

if {**condition**} then {**evidence weights for each class**}

where the rule condition is usually defined in terms of bounds on feature values, and where rules instantiated by data activate weighted evidence for different pattern classes. Such rules can be defined over pattern features of arbitrary arities and the main problem in EBS has been to determine the feature bounds and evidence weights. That is, EBS typically involves partitioning feature spaces into regions associated with different pattern classes, and the problem has been to determine classification rules that attempt to minimize misclassification while, at the same time, maximizing rule generalization. Since these regions are not necessarily class disjoint, evidence weights are typically used to index the degrees to which samples within the region correspond to different classes. "Generalization" is then defined by the associated volumes of the regions that define the rules in feature space.

Evidence weights are typically derived from the relative frequencies of different classes per region [5] or, more recently, by minimum entropy and associative neural network techniques [6]. In either case, the label-compatibilities of data parts and their relations were only encoded through the simultaneous activation of both unary and binary rules.

Although such systems allow generalizations from samples, they only attain implicit learning of the RS, in so far as unary rules (rules related to part features) and binary rules (rules related to part relational features) are both activated to evidence patterns or objects. EBS do not explicitly consider the compatibility between unary and binary rules as they reference specific pattern parts and their relations. Indeed, patterns are uniquely defined by the enumeration of specific *labeled* unary and binary feature states of the form $U_i - B_{ij} - U_j$. The two patterns shown in Figure 1 have isomorphic unary and binary feature states but are not identical. This shows that the existence of such correspondences does not guarantee identity in shape unless the unary and binary feature labels are compatible. Even given this, determining the uniqueness of a pattern may involve checking for attribute and label consistencies of higher order than, say, the consistencies of isolated parts or part-relation pairs. Rules satisfying the "label compatibility" property of rules must evidence specific objects or patterns uniquely, i.e. lists of unary and binary feature states must evidence *specific joint occurrences of parts and relations.* The problem then is how rules having this property can be generated automatically.

As already stated, the simplest representation for visual patterns that takes into account the label-compatibility of unary and binary features, is a graph. Graph

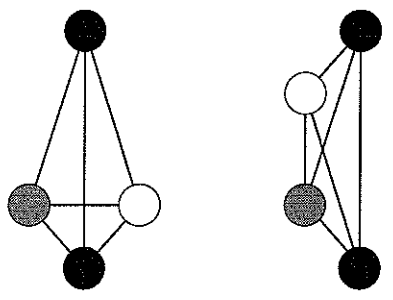

Figure 1: Two patterns with isomorphic unary (U = vertex color and orientation) and binary (B = line length and orientation) feature states but differing in their label-compatibilities: the sequences of $U_i - B_{ij} - U_j...$ differ between the two patterns).

matching techniques are used to solve the recognition problem where a sample pattern structure (for example, new data for classification) is matched to a model structure by searching for a label assignment that maximizes some objective similarity function [2]. Pattern classes are represented by sets of instances and classification is thus achieved by searching through all model graphs to determine the one producing the best match. This representation and graph matching approach, in the form of interpretation trees and feature indexing, has been the preferred architecture for object recognition [4, 7].

Different approaches to improving the efficiency of the matching processes have been proposed, such as constraint-based decision trees [3], "pre-compiled" tree generation [8], heuristic search techniques [9], dynamic programming [10], relaxation labeling [11] or hierarchical model fitting [12]. However, the problem of learning and constructing union and discrimination trees for structural descriptions has been addressed only sporadically in the literature, such as in [13] within the framework of inductive learning of symbolic structural descriptions or in [14] within the framework of probabilistic inductive prediction of sequential patterns.

In summary, graph matching methods solve the label-compatibility problem but do not allow for efficient representation of pattern classes via union and discrimination trees. Further, such representations and algorithms do not consider a fundamental issue in pattern recognition, *generalization*, i.e. the ability for the system to recognize equivalences between patterns that are not identical. Also, they do not fully exploit learning to determine the optimal search path amongst unary and binary feature states to evaluate the existence of specific patterns. For example, in 3D object recognition, it is often necessary to classify objects as belonging to a specific object type even though individual samples of the class may be non-rigid transformations of other members of the same class - as in different types of coffee mugs, etc. At the same time, we wish to automatically generate descriptions of 3D objects that not only enable such generalizations but also do so with respect to the description length of the rules (the length of strings of unary-binary-unary-... feature bounds). Evidence-based systems provide for generalization, but do not adequately address the label-compatibility problem.

In the following Sections we focus on the analysis of a new technique for the learning of structural relations, **Conditional Rule Generation** (CRG). It generates a tree of hierarchically organized rules for classifying structural pattern descriptions that aims at "best" generalizations of the rule bounds with respect to rule length (the number of U-B-U, etc., conditional feature lists). The aim of this paper is to show how the technique can be used to solve problems involving the recognition of 2D patterns and 3D objects in complex visual scenes.

2 The Conditional Rule Generation Method

In CRG, rules are defined as clusters in Conditional Feature Spaces which correspond to either unary or binary features of the training data. The clusters are generated to satisfy two conditions: one, they should maximize the covering of samples from one class and, two, they should minimize the inclusion of samples from other classes. In our approach, such rules are generated through controlled decision tree expansion and cluster refinement as described below.

2.1 Cluster Tree Generation

Each pattern (a 2D sample pattern or a view of a 3D object) is composed of a number of parts (pattern components) where, in turn, each part $p_r, r = 1, ..., N$ is described by a set of unary features $\vec{u}(p_r)$, and pairs of parts (p_r, p_s) belonging to the same sample (but not necessarily all possible pairs) are described by a set of binary features $\vec{b}(p_r, p_s)$. Below, $S(p_r)$ denotes the sample (in 3D object recognition, a "view") a part p_r belongs to, $C(p_r)$ denotes the class (3D object recognition - object) $S(p_r)$ belongs to, and H_i refers to the information, or cluster entropy statistic:

$$H_i = -\sum_k q_{ik} \ln q_{ik} \tag{1}$$

where q_{ik} defines the probability of elements of cluster i belonging to class k. We first construct the initial unary feature space for all parts over all samples and classes $U = \{\vec{u}(p_r), r = 1, .., N\}$ and partition this feature space into clusters U_i. In our approach, the initial clustering procedure is not critical, as will be discussed further below. Clusters that are unique with respect to class membership (with entropy $H_i = 0$) provide a simple classification rule for some patterns (e.g. U_3 in Figure 2). However, each non-unique (unresolved) cluster U_i is further analyzed with respect to binary features by constructing the (conditional) binary feature space $UB_i = \{\vec{b}(p_r, p_s) \mid \vec{u}(p_r) \in U_i \text{ and } S(p_r) = S(p_s)\}$. This feature space is clustered with respect to binary features into clusters UB_{ij}. Again, clusters that are unique with respect to class membership provide classification rules for some objects (e.g. UB_{11} in Figure 2). Each non-unique cluster UB_{ij} is then analyzed with respect to unary features of the second part and the resulting feature space $UBU_{ij} = \{\vec{u}(p_s) \mid \vec{b}(p_r, p_s) \in UB_{ij}\}$ is clustered into clusters UBU_{ijk}. Again, unique clusters provide class classification rules for some objects (e.g. UBU_{121} in Figure 2), the other clusters have to be further analyzed, either by repeated conditional clustering involving additional parts at levels *UBUB*, *UBUBU*, etc. or through cluster refinement, as described below.

Each element of a cluster at some point in the cluster tree corresponds to a sequence $U_i - B_{ij} - U_j - B_{jk}...$ of unary and binary features associated with a non-cyclic sequence (path) of pattern parts. In the current implementation, we analyze all path permutations in order to guarantee classification of arbitrary partial patterns, even though this leads to the generation of redundant set of rules. Elsewhere, we have studied ways of reducing this redundancy through the use of feature ordering [15].

In the current implementation of CRG, we have used a simple splitting-based clustering method to enable the generation of *disjoint* rules and to simplify the clustering procedure. Cluster trees are generated in a depth-first manner up to a maximum level of expansion. Clusters that remain unresolved at that level are split in a way described in the following Section.

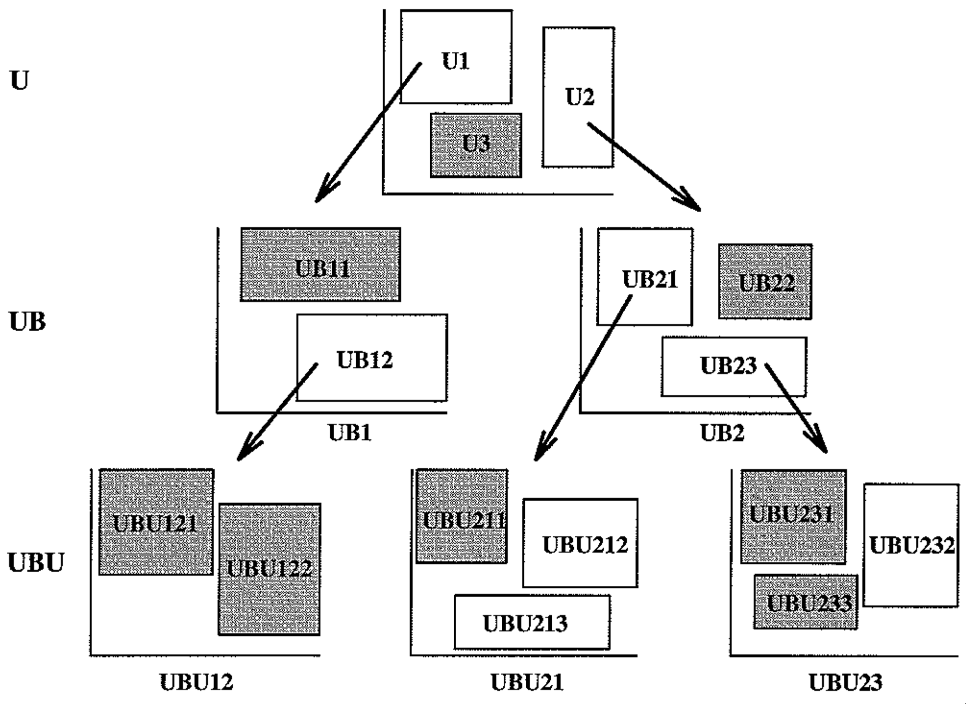

Figure 2: Cluster Tree generated by the Conditional Rule Generation Procedure (CRG). The unresolved unary clusters (U_1 and U_2) - with element from more than one class - are expanded to the binary feature spaces UB_1 and UB_2, from where clustering and expansion continues until either all rules are resolved or the predetermined maximum rule length is reached, in which case rule splitting occurs.

2.2 Cluster Refinement

All non-unique (unresolved) clusters remaining at a given level of the cluster-tree generation (e.g. clusters UBU_{212}, UBU_{213} and UBU_{232} in Figure 2) have to be analyzed further to construct unique decision rules. One way of doing this is to simply expand the cluster tree, analyzing unary and binary attributes of additional parts to generate rules of the $\{UBUB...\}$ form. However, this may never give completely "resolved" branches in the cluster tree. Alternatively, the derived clusters in the tree can be refined or broken into smaller clusters, using more discriminating feature bounds, as described below. Both approaches have their respective disadvantages. Cluster refinement leads to an increasingly complex feature-space partitioning and thus may reduce the generality of classification rules. Cluster-tree expansion, on the other hand, successively reduces the possibility of classifying pattern fragments, or, in 3D object recognition, classifying objects from partial views. In the end, a compromise has to be established between both approaches.

In cluster refinement, two issues must be addressed, the refinement me-thod and the level at which cluster refinement should be performed. Consider the cluster tree shown in Figure 2 with non-unique clusters UBU_{212}, UBU_{213} and UBU_{232}. One way to refine clusters (for example, cluster UBU_{232}) is to re-cluster the associated feature space (UBU_{23}) into a larger number of clusters. However, classification rules associated with other clusters (UBU_{231} and UBU_{233}) are lost and have to be recomputed. Alternatively, given that each cluster is bounded by a hyper-rectangle in feature space, refinement of a cluster can be achieved by splitting this rectangle along some optimal boundary. This ensures that other sibling clusters remain unaffected. With respect to the level at which cluster refinement is performed, instead of splitting an unresolved leaf cluster (UBU_{232}) one could split any cluster in the chain of parent clusters (UB_{23} or U_2).

Consider splitting the elements of an unresolved cluster C along a (unary or binary) feature dimension F. The elements of C are first sorted by their feature value $f(c)$, and then all possible cut points T midway between successive feature values in the sorted sequence are evaluated. For each cut point T, the elements of C are partitioned into two sets, $P_1 = \{c \mid f(c) \leq T\}$ with n_1 elements and $P_2 = \{c \mid f(c) > T\}$ with n_2 elements. We define the partition entropy $H_P(T)$ as

$$H_P(T) = n_1 H(P_1) + n_2 H(P_2). \tag{2}$$

the cut point T_F that minimizes $H_P(T_F)$ is considered the best point for splitting cluster C along feature dimension F (see also [16]). The best split of cluster C is considered the one along the feature dimension F that minimizes T_F. As noted above, rather than splitting an unresolved leaf cluster C_L, one can split any cluster C_i in the parent chain of C_L. For each cluster C_i, the optimal split T_F is computed, and the cluster C_i that minimizes T_F is considered the optimal level for refining the cluster tree. Clusters above C_L may contain elements of classes other than those that are unresolved in C_L. Hence, in computing H_P for those clusters, we consider only elements of classes that are unresolved in C_L.

Two further properties of the splitting procedure are important, since they affect the type of rules generated by CRG. First, if a nonterminal cluster of the cluster tree is split, the feature spaces conditional upon that cluster are recomputed since the elements of the feature space have changed. Second, in the case of a tie, i.e. if two or more clusters have the same minimal partition entropy $H_P(T)$, the cluster higher in the cluster tree is split. Together, this leads to CRG having a clear preference for shallow cluster trees and for short rules, which, in turn, leads to efficient rule evaluation.

The rules generated by CRG are sufficient for classifying new pattern or pattern fragments, provided that they are sufficiently similar to patterns presented during training and provided that the patterns contain enough parts to instantiate rules. However, cluster trees and associated classification rules can also be used for partial rule instantiation. A rule of length m (for example, a *UBUBU*-rule) is said to be partially instantiated by any shorter ($l < m$) sequence of unary and binary features (for example, a *UBU*-sequence). From the cluster tree shown in Figure 2, it is clear that a partial instantiation of rules (for example, to the *UB*-level) can lead to unique classification of certain pattern fragments (for example, those matched by the U_3 or UB_{11} rules, but it may also *reduce* classification uncertainty associated with other nodes in the cluster tree (for example, UB_{23}). From the empirical class frequencies of all training patterns associated with a node of the cluster tree (for example, UB_{23}), one can derive an expected classification vector, or *evidence* vector. The evidence vector is used to predict the classification vector of any part, or sequence of parts, that instantiates the associated rule.

In summary, CRG has been specifically developed to enable the learning of patterns defined by parts and their relations. The technique determines the type of inductive learning (attribute generalizations) that can be performed and the associated minimum length descriptors of shapes for recognition. Finally, since the method precompiles patterns as relational trees, the technique is ideally suited for the learning of patterns with variable complexity and their detection in scenes.

3 Detecting 2D Patterns in Scenes

In this Section, we illustrate learning of 2D patterns using the CRG method, the recognition of these patterns embedded in more complex scenes using the rules

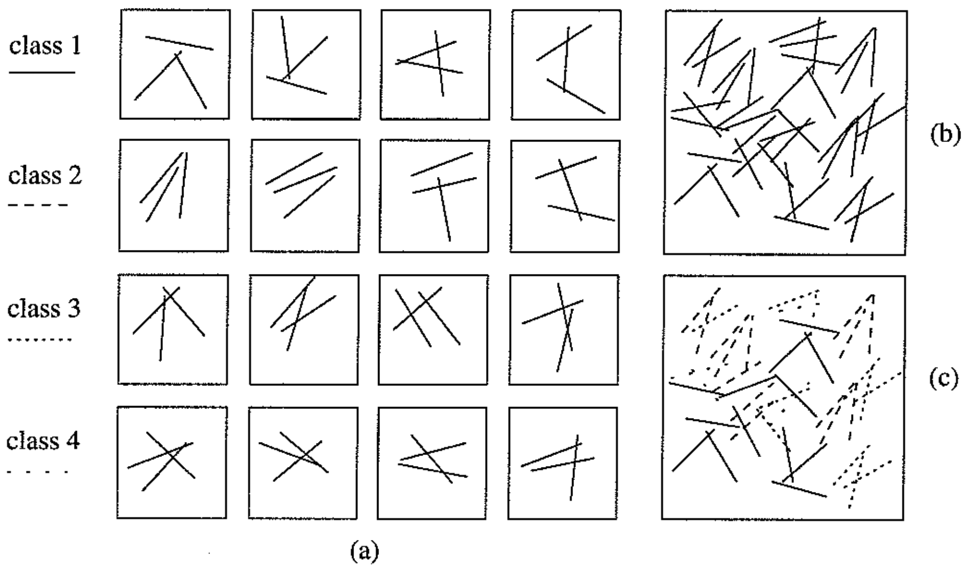

Figure 3: (a) Four classes of patterns with four training patterns (views) each. Each pattern is composed of three lines. Lines are described by the unary features "line length" and "orientation", and pairs of lines are described by the binary features "distance of line centers" and "intersection angle". (b) Montage of (slightly distorted) line triples. (c) In the adjacency graph for the montage, dots indicate the position of the line center and adjacent lines (with a center distance below a given limit) are connected. (d) Result of the pattern classification using the rules generated by CRG. Class labels for each line are shown on the right.

generated by CRG. The example, **line triples**, consists of four classes of patterns with four training examples each (see Figure 3a). Each pattern is described by the unary features "length" and "orientation", and the binary features "distance of line centers" and "intersection angle". The line patterns are simplified versions of patterns found in geomagnetic data that are used to infer the presence of certain metals or minerals.

CRG was run with maximum rule length set to $maxlevel = 5$ (i.e. rules up to the form of *UBUBU*are being generated), and it produced 35 rules, 3 *U*-rules, 18 *UB*-rules, 2 *UBU*-rules, and 12 *UBUB*-rules.

At recognition time, a montage of patterns was presented (see Figure 3b), and the patterns were identified and classified as described below, producing the classification result shown in Figure 3d. Pattern identification and classification was achieved using the following steps:

1) Unary features are extracted for all scene parts (lines), and binary features are extracted for all adjacent scene parts, i.e. pairs whose center distance does not exceed a given limit. The adjacency graph is shown in Figure 3c, where dots indicate the position of the line centers, and adjacent pattern parts (lines) are connected.

2) Given the adjacency graph, all non-cyclic paths up to a certain length l are extracted, where $l \leq maxlevel$. These paths, termed *chains*, constitute the basic units for pattern classification. A chain is denoted by $S =< p_i, p_j, ..., p_n >$ where each p_i denotes a pattern part. For some chains, all parts belong to a single learned pattern, but other chains are likely to cross the "boundary" between different patterns.

3) Each chain $S =< p_i, p_j, ..., p_n >$ is now classified using the classification rules produced by CRG. Depending on the unary and binary feature states, a chain may or may not instantiate one (or more) classification rules. In the former case, rule instantiation may be partial (with a non-unique evidence vector $\vec{E}(S)$), or complete

(with $H[\vec{E}(S)] = 0$). As discussed above, the evidence vector for each rule instantiation is derived from the empirical class frequencies of the training examples.

4) The evidence vectors of all chains $< p_{i_1}, p_{j_1}, ..., p_n >$, $< p_{i_2}, p_{j_2}, ..., p_n >$, etc., terminating in p_n determine the classification of part p_n. Some of these evidence vectors may be mutually incompatible and others may be non-unique (through partial rule instantiation). Here, we have studied two ways of combining the evidence vectors, a winner-take-all solution and a relaxation labeling solution.

Implementation of the **winner-take-all** (WTA) solution is straightforward. The evidence vectors of all chains terminating in p_n are averaged to give $\vec{E}_{av}(p_n)$, and the most likely class label is enacted. However, the WTA solution does not take into account that, for a chain $S =< p_i, p_j, ..., p_n >$, the average evidence vectors $\vec{E}_{av}(p_i)$, $\vec{E}_{av}(p_j)$, ...,$\vec{E}_{av}(p_n)$ may be very different and possibly incompatible. If they are very different, it is plausible to assume that the chain S is "crossing" boundaries between different patterns/objects. In this case, the chain and its evidence vectors should be disregarded for the identification and classification of scene parts.

This is achieved in the **relaxation labeling** (RL) solution, where evidence vectors are weighted according to intra-chain compatibility. Specifically, the RL solution is given by

$$\vec{E}^{t+1}(p_i) = \Phi \left[\sum_{S=<p_i \dots p_n>} \vec{E}^t(p_i) C(p_i, p_n) \right] \tag{3}$$

where $\vec{E}^t(p_i)$ corresponds to the evidence vector of p_i at iteration t, with $\vec{E}^0(p_i) = \vec{E}_{av}(p_i)$. $C(p_i, p_n)$ corresponds to the compatibility between parts p_i and p_n, and Φ is the logistic function

$$\Phi(z) = (1 + \exp[-20(z - 0.5)])^{-1}. \tag{4}$$

Further, we have encoded the compatibility function in terms of the scalar product between the evidence vectors of parts p_i and p_n,

$$C(p_i, p_n) = \vec{E}(p_i) \cdot \vec{E}(p_n). \tag{5}$$

For identical evidence vectors $\vec{E}(p_i)$ and $\vec{E}(p_n)$, $C(p_i, p_n) = 1$, and for incompatible evidence vectors, for example $\vec{E}(p_i) = [1, 0, 0]$ and $\vec{E}(p_n) = [0, 1, 0]$, $C(p_i, p_n) = 0$.

Compatibility of evidence vectors is a weak constraint for updating the evidence vectors of each part and it may even have an adverse effect if the adjacency graph is complete. Much stronger constraints can be derived from, for example, the label-compatibilities between pattern parts, or from pose information in the case of 3D object recognition. The usefulness of such information is, however, pattern dependent and considered beyond the scope of the present paper. In any case, for the simple patterns shown in Figure 3, and the low connectivity of the adjacency graphs of the montages, the relaxation method outlined here proved to be sufficient to obtain perfect part labeling. The results obtained using this technique are shown in Figure 3d.

4 3D Object Recognition using Range Data

4.1 Encoding of Object Surfaces

In the previous Section, we have illustrated the CRG method with a recognition problem involving 2D line patterns. For 3D recognition systems, the input can consist of intensity (brightness and/or color) data generated by a video camera, or of range (depth) data. The latter can be sensed by active vision (laser range finders or strip lighting devices) or can, for example, consist of sparse depth maps produced by Shape-from-X methods.

We deal with range data, and for the purpose of this paper, we do not deal with this initial sensing problem and simply assume that we already have view-dependent range (depth) maps of 3D objects. However, as in the 2D case, we deal with the recognition of isolated objects and objects in scenes. One of the main reasons for using such view-dependent data formats is that the computations of surface curvatures, or pixel labels in general, are restricted to what is visible. That is, there exists full view-independent surface information that is not visible: for example, the "inside regions" of some concave objects. The additional benefit of computing curvatures from such a data format (Monge patch data of the form $(x, y, z(x, y))$ is that more standard signal processing techniques can be used to regularize the evaluation of derivatives, etc. (see [17] for more details). What is important, however, is that we have computed object unary and binary part features with respect to the full 3D properties of the range data. That is, questions as to the benefits and deficits of view-dependent versus view-independent representations involves evaluations of both the data format and the types of features to be computed.

Full view-independent representations involve complete 3D descriptions of surface patches and the fact that these patch features are evaluated from view-dependent aspects is actually not the essential issue involved. For example, computing surface features that are invariant to rigid motions is as important to a "view-independent" representation as that of using full 3D CAD models. That is, for recognition purposes, it is the invariance of the representation that determines the degree of invariance in the models as much as the types of data inputs used. For these reasons, we have adhered to the view-dependent format. Further, the issue of the minimum number of views required to obtain correct identification of objects invariant to view is not so much a problem of the data formats but a problem of the types of object classes involved. For example, we only need one view of an ant and one of an elephant for fully invariant and correct 2-object classification performance!

Over the past decade, a variety of techniques have been developed for the registration of surface "shape" that produce representations which are invariant to rigid motions - a condition of central importance to robust Object Recognition Systems (ORS). Principal curvatures, Mean (H) and Gaussian (K) curvatures satisfy these conditions [18] though there are many different methods available for computing them. H and K are defined by:

$$H = \frac{1}{2}\frac{f_{xx} + f_{yy} + f_{xx}f_y^2 + f_{yy}f_x^2 - 2f_xf_yf_{xy}}{(1 + f_x^2 + f_y^2)^{\frac{3}{2}}} \tag{6}$$

and

$$K = \frac{f_{xx}f_{yy} - f_{xy}^2}{(1 + f_x^2 + f_y^2)^2} \tag{7}$$

for the Monge patch (view-dependent depth map) case where f_{uv} refers to partial differentiation of f with respect to u ($u = x$) and v ($v = y$) and $f(x,y)$ to the view-dependent range image.

Such computations require initial surface smoothing which is usually accomplished by fitting quadratic surfaces [19] or by low-pass filtering (surface blurring), after which partial derivatives are computed. Using this latter form of smoothing we have also used Fourier methods to compute the derivatives. That is, from the Differentiation Theorem [20] the partial derivatives of the function $f(x, y)$ (representing, in this case the Monge patch surface model $(x, y, z = f(x, y))$) is determined (for each variable denoted by x) by:

$$\frac{\partial^n f}{\partial x^n} = F^{-1}((iu)^n F) \tag{8}$$

where F corresponds to the Fourier transform of f and F^{-1} to the inverse Fourier transform. That is, to partially differentiate an image $f(x,y)$ with respect to x, its Fourier transform is multiplied by the real (quadratic, second order of differentiation) or imaginary (linear, first order) ramp function $(iu)^n$ - resulting in even and odd bandpass filters. The benefits of such methods lie in the degree of "support" for computing $f_x, f_y, f_{xy}, f_{xx}, f_{xy}$ - the components of H and K. Furthermore, one of the main sources of "noise" in computing H and K lies in the division of images having different differential (bandpass) information - particularly in the regions of curvature zero-crossings. Our solution has been to compute zero-crossings, or segmentation, directly from the determinant of the Hessian ("shape" operator), the numerator of K):

$$S(x,y) = f_{xx}f_{yy} - f_{xy}^2 \tag{9}$$

segmenting the surface into convex, concave and planar regions. We then compute the complete H and K values within the resultant regions (see the following Section) using the low-pass filtering in conjunction with the spectral method for the computation of derivatives (see (8) above). The net result is to produce estimates of H and K with respect to a "scale" defined by the low-pass filter.

4.2 Segmentation

The issue of segmentation for ORS's, and for range data specifically, has received a good deal of attention in recent years. Common to most approaches is the development of surface part clustering in terms of similarities in surface point position, normals, or curvature information or surface curve fitting parameters. Segmentation, in these low-level terms does not guarantee the derivation of "parts" that are consistent with, for example, "model parts" defined by other processes, and some attempts have been made to split and merge such initially segmented regions, consistent with known patch feature bounds of the object parts in the database [7].

An alternative way of guaranteeing compatibility between model and test data parts is to use a segmentation procedure that is guaranteed to *apply equally* to both domains and uses features that are invariant to the parameterization of the surface. Fortunately, Mean (H) and Gaussian (K) curvatures satisfy these conditions. We have chosen to use zero-crossings of the determinant of the Hessian (see (9) above) as our segmentation procedure - which determines convex, concave and planar regions in a way which minimizes noise amplification that typically occurs when full H and/or K zero-crossings are evaluated. Such a segmentation procedure applies equally to models and data and is invariant to rigid motions. As mentioned above, we still use full H and K values to characterize each such region and so the initial segmentation is simply an adaptive data reduction method to package surface parts in ways that can be compared across data and models.

The major problem with using zero-crossings lies in determining what constitutes "zero". The problem of thresholds for zero-crossings has recently been discussed [21]. Here, we have used a straight forward training approach where the threshold was determined from the maximum non-zero value of the Hessian's response (9) to the known planar background, assuming that scene objects are in front of a planar background [22].

4.3 Feature Extraction

In ORS, the purpose of segmentation is to enable an efficient data structure for the definition of models by the properties of surface patches and their relational features. Such features need to optimize two somewhat contradictory goals: invariance and uniqueness. The former refers to the need to represent models in ways which are

Predicate	*Type*	*Computation*
Unary	Size	U.D.1 Area
	Span	U.D.2 3D Spanning distance (Max)
		U.B.1 Perimeter
		U.B.2 mean Curvature
		U.B.3 mean torsion
Binary	B-type	B.B.1 length of jumps
		B.B.2 length of creases
	Jumpgap	B.D.1 bounding distance
		B.D.2 Centroid distance
		B.D.3 Maxdistance
	B-angle	B.A.1 differences in normal angles
		B.A.2 average bounding angle between surfaces
	N-angle	B.A.3 normal angle differences

Table 1: Typical Unary and Binary Surface Features

invariant to rigid motions, pose, etc., while the latter refers to the development of representations which uniquely define the model. For example, H and K are invariant to rigid motions but are only unique up to the general type of surface and do not uniquely define it. Such uniqueness comes from the Gauss-Weingarten equations with the Mainardi-Codazzi compatibility equations defining the constraints on the differential (tensor) operators [18].

Model surface features are usually of two generic forms [22]. Unary features refer typically to (local) surface patch properties (such as curvatures), global patch properties (such as areas), or to to properties of patch boundaries (such as perimeter). Binary features typically capture part relationships such as distances, angles, and also include boundary relationships (see below). Typical examples of all feature types are shown in Table 1 (center column) and those used in this implementation are shown in Table 1 (right column). The right-hand column groups features into different types, unary curvatures (U.C), unary distance (U.D), unary boundary (U.B) and binary boundary (B.B), binary distance (B.D) and binary angle (B.A).

We have employed statistics of the pixel ("local") Mean and Gaussian curvatures of each patch. These features define surface shape characteristics that are invariant to rigid motions. Such measures eliminate the need for less quantitative features, such as "sense" which defines only the type of surface shape. Such "local" unary features are view-independent and they enable the identification of a part when only a section of it is visible - given that the section is representative of the part shape (i.e., if it is an unbiased sample). "Global" unary features are less invariant since they are computed over a full patch and so are subject to, for example, self-occlusion for different views. We have, however, included the areas, perimeters and spanning distances as already used in current implementations - though area is directly related to the average of the Gaussian curvature. We have defined part boundary (edge) properties by their associated curvature and torsion statistics. In particular, the torsion statistic defines the degree to which the bounding contours deviate from planarity and, from the Serret-Frenet equations, these features uniquely define a contour in 3D - invariant to rigid motions. Here we have computed boundary contour curvatures(κ) and torsion(τ) statistics by their finite different forms. Curvature is defined by

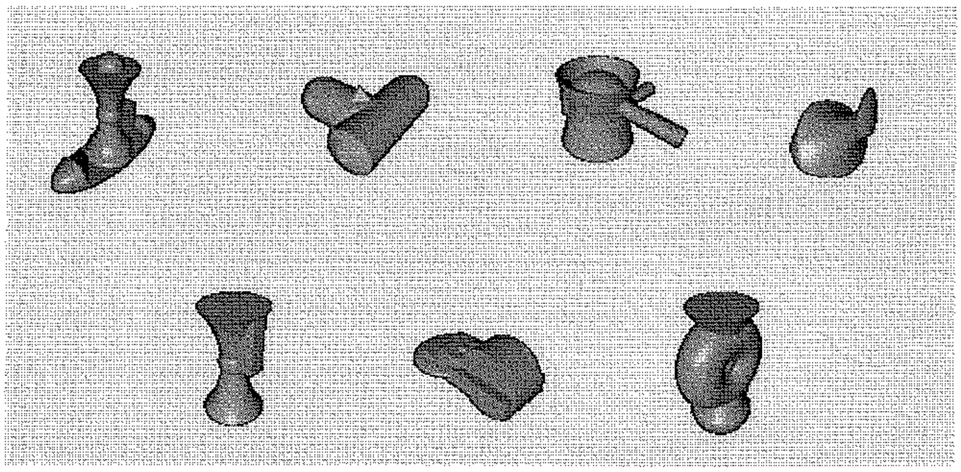

Figure 4: Rendered view of each of the seven objects used in the 3D object recognition experiment.

$$\kappa(s) = \frac{1}{\mid t_s(s) \mid} \tag{10}$$

where

$$t_s(s) = (x_s(s+1) - x_s(s), y_s(s+1) - y_s(s), z_s(s+1) - z_s(s))$$

and

$$X_s(s) = (x(s+1) - x(s), y(s+1) - y(s), z(s+1) - z(s))$$

for s being the parameter of the curve (contour) equation

$$X(s) \equiv (x(s), y(s), z(s)).$$

Torsion is defined by

$$\tau(s) = -b_s(s) \cdot n(s) \tag{11}$$

where

$$b(s) = \tau(s) \times n(s)$$

with $\times$ corresponding to vector (cross) product, and

$$n(s) = k(s) / \mid b(s) \mid$$

corresponding to the normal to the curve at s.

We have used the binary features total lengths of jumps and creases between shared contours, consistent with recent object recognition systems (for example, [5]). Such continuous versions of these binary features are more suitable for a feature space (vector space) representation and for the rule generation (clustering) procedure proposed in the current paper.

4.4 Learning Structural Descriptions of Objects

In the 3D object recognition example, seven synthetic objects were learned at training time. Each object was presented in isolation and from 20 different views (equally spaced over three Euler angles). Examples views of each object are shown in Figure 4.

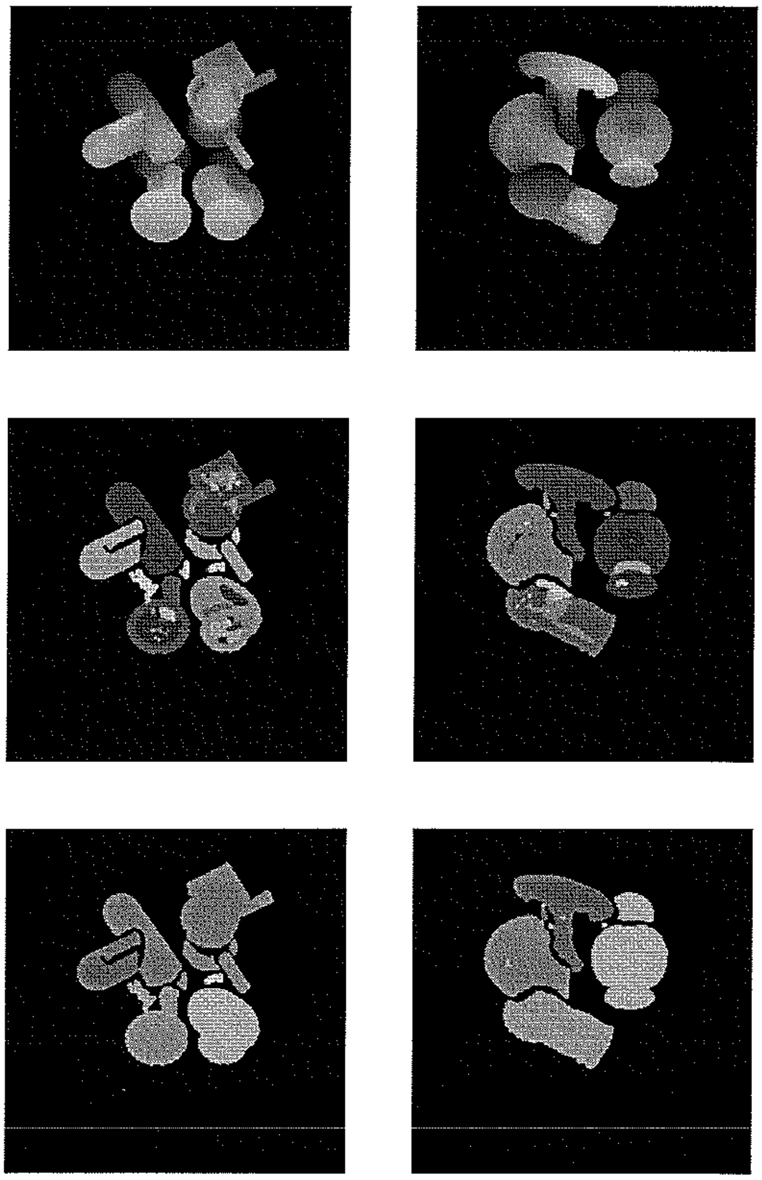

Figure 5: Two different montages of synthetic objects (left and right panel). (Top) Range images of two scenes used to test object identification and classification. (Middle) Segmented depth map regions (defined by different grey levels) from the zero-crossings of Gaussian curvature. (Bottom) Region classification for the two montages. Different grey levels define different class labels.

Analysis of the depth maps for each object and view proceeded as described in the Sections 4.1 - 4.3, resulting, for each view, in a set of depth map regions that were described by the unary and binary features shown in Table 1.

These rules were then used to classify montages of objects such as shown in Figure 5. Here, the top row shows the depth maps of two montages, the middle row shows the segmented depth map regions, and the bottom row shows the region classifications. For the montage on the left, object overlap is relatively small, and for the montage on the right it is substantial. Chain analysis and part classification proceeded as described in Section 3, both for the WTA-scheme and the RL scheme.

	Number of parts	WTA scheme	RL scheme
left scene	82	76	78
right scene	63	53	56

Table 2: Number of correct region classifications for two scenes in Figure 5, using the WTA-scheme and the RL-scheme.

In the RL scheme, relaxation was run for 20 iterations, followed by a WTA iteration on the final evidence vectors to produce a unique classification for each region. A summary of the correct region classification, for both schemes and for both the left and right montage in Figure 5, is given in Table 2. From the results in this Table, as well as from the classification map in Figure 5, it is clear the the rules produced by CRG are capable of classifying correctly a majority of object regions.

5 3D Object Recognition using Intensity Data

The **blocks** example presented in this Section consists of various configurations of colored toy blocks. The configurations are learned in isolation (see Figure 6) and have to be identified in more complex arrangements (see Figure 7). The training set consisted of 5 classes of block configurations, each with three training examples, and the test arrangements consisted of up to 20 blocks.

Images of the training and test scenes were captured with a color camera. Preprocessing was fairly simple, consisting of a *segmentation* stage and a *feature extraction* stage. Segmentation was achieved using a form of K-means clustering (minimizing within-cluster variance in feature space) on position (x, y) and color (r, g, b) attributes [23]. For the resulting clusters, small clusters were merged with larger neighbor clusters in order to eliminate spurious image regions. Given the rich image information, it is not surprising that the resulting image regions correspond fairly well to the individual blocks.

In the feature extraction stage the following unary features were extracted for each image region: size (in pixels), compactness $(perimeter^2/area)$, and the normalized color signals $R/(R+G+B)$, $G/(R+G+B)$, and $B/(R+G+B)$. For pairs of image regions the following binary features were computed: absolute distance of region centers, minimum distance between the regions, distance of region centers normalized by the sum of the region areas, and length of shared boundaries normalized by total boundary length.

For the training data, CRG analyzed 276 different paths of pattern parts and produced 32 rules: 9 *U*-rules, 4 *UB*-rules, 12 *UBU*-rules, 3 *UBUB*-rules, and 4 *UBUBU*-rules. From the distribution of rule types, it is evident that CRG used predominantly unary features for classification. Given the fact that CRG has a strong tendency to produce shallow cluster trees and short rules (see Section 2.2), and given the fact that the unary features are quite diagnostic (see Figure 6), this result is not surprising. However, each unary and binary feature was used in at least some of the classification rules.

Classification performance was tested with several complex configurations of block patterns, two of which are shown in Figure 7, together with the classification results. Classification proceeded as described in Section 3, using the chain analysis and relaxation labeling solution. For both scenes, all parts (11 in Figure 7a, 17 in Figure 7b) were classified correctly with the exception of a single part from the class-4 configuration (see Figures 7c and 7d).

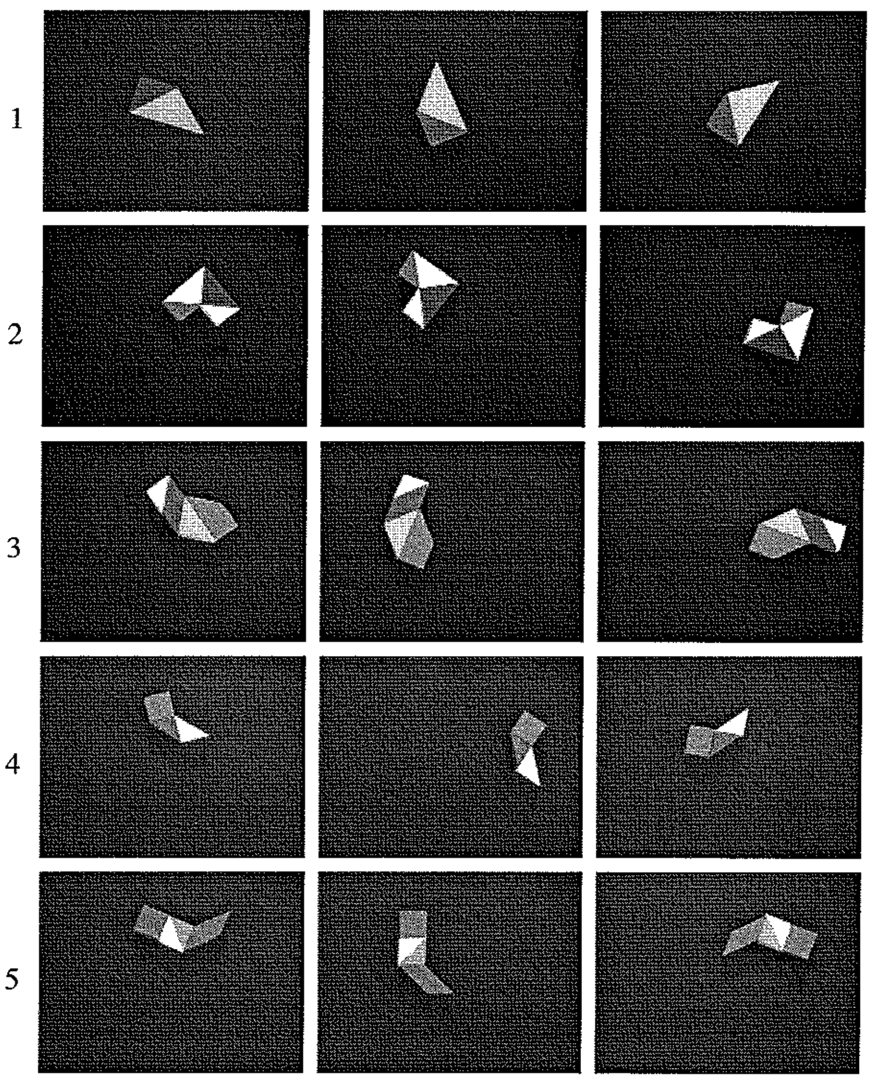

Figure 6: Images of five classes of toy block configurations with three views each. The image parts are described by the unary features size, eccentricity and the three normalized color coordinates. Pairs of image parts are described by the binary features of midpoint distance, area-normalized midpoint distance, minimum distance and normalized shared boundary length.

For comparison purposes, we have analyzed the block example using classical decision trees [24]. In the first analysis, each image part P of the training and test images was described by 13 features. These features consisted of the five unary features of P (see above), the four binary features (see above) of the relation between P and its closest neighbor, and another four binary features of the relation between P and its second-closest neighbor. For the class-1 cases which consisted of two parts only, the feature values for the second binary relation were set to "unknown". A decision tree was generated using C4.5 with default parameters [24], and the resulting tree was used to classify all parts of the test scenes in Figure 7. In each of the two scenes, 3 parts were misclassified. The good performance obtained with C4.5 is consistent with the observation that the use of higher-order relational information does not seem to be crucial for successful classification of this data set.

In this first analysis, features of all UBB-triples (unary features and binary

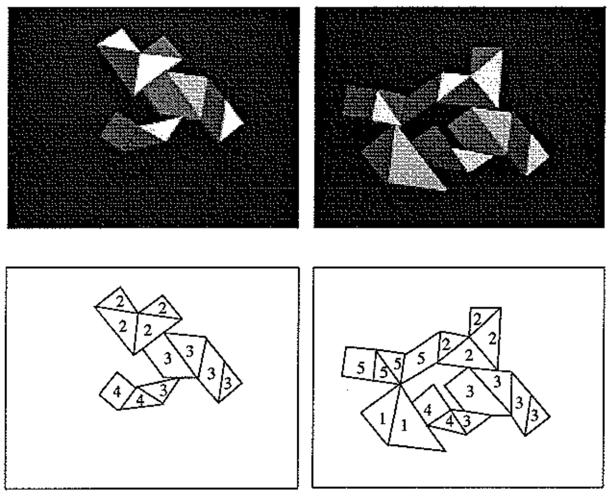
Figure 7: Two block scenes and their classifications. (a) Block scene consisting of 11 blocks corresponding to examples of classes 2, 3, and 4. (b) Block scene consisting of 17 blocks corresponding to examples of all classes. (c) Classification result for block scene in (a) with region labels corresponding to classes. (d) Classification result for block scene in (b) with region labels corresponding to classes.

features of relations with two other parts) were used for classification. A second analysis, using *UBU*-triples (with 14 features: the same five unary features of all pairs of parts, as well as the same four binary features of their relation) was performed, but the results cannot be interpreted as easily. For the scene in Figure 7a, 33 out of 110 *UBU*-triples or 30% were misclassified, and for the scene in Figure 7b 103 out of 272 *UBU*-triples or 37.8% were misclassified. One reason for the error rate being so high is the fact that no analysis corresponding to the chain analysis described in Section 3 was performed with the C4.5 results. However, the error rates seem to be too high to be corrected using the relaxation scheme proposed there.

A general point is, however, more important. The CRG method generates rules of (minimal) variable length *optimized for a given training set*, whereas the decision tree (C4.5) *fixes* the dimensionality of the feature space and rule length. The choice of *UBB*-triples for the block example lead to a C4.5 performance that was essentially the same as that of CRG, but for the *UBU*-triples C4.5 performance was much worse. This choice has to be done *a priori* whereas it is adjusted dynamically in the CRG method.

6 Discussion

CRG develops structural descriptions of patterns in the form of decision trees on attribute bounds of ordered predicates (see Figure 2). It is thus useful to compare it with other techniques from Machine Learning which attain similar ends symbolically.

CRG shares with ID3 / C4.5 [25, 24], and related techniques, similar methods for the search and expansion of decision trees. However, these latter techniques were not designed to generate rules satisfying label compatibility between unary and binary predicates. CRG, on the other hand, is explicitly designed to develop rules for unique identification of classes with respect to their "structural" (i.e. linked unary and binary feature) representation. The application of C4.5 to the block example in the previous Section was therefore somewhat misleading, in the sense that label-compatible data were generated beforehand.

In decision trees, features or attributes are analyzed within a single feature space, independent of their relationships or arities, and no preferential order is imposed on the features. In contrast, the CRG method generates conditional features spaces as required, and it defines a preferential ordering on attributes in the sense that, for example, a split of a U-feature is preferred over a split of UBU-features. This preferential order leads to the generation of shallow cluster trees and short rules, as discussed in the previous Sections.

Decision trees operate on a fixed path length (for example, the UBB- or UBU-triples in the block example) and thus *force*, a priori, the choice of relational structures to be analyzed. CRG, on the other hand, has variable length path expansion determined by the number of parts and their relations that are required to uniquely define patterns. Consequently, CRG is superior to classic decision trees when classification relies on relational information and does so to different degrees for different examples or classes. Under these circumstances one would be forced to use high-dimensional features spaces with classical decision trees, whereas CRG would generate minimal depth trees. Furthermore, generating minimum depth trees is of crucial importance since the number of paths grows exponentially with path length.

In summary on can say the classical decision trees are *attribute-indexed* in the sense that various levels in the tree define different attributes and the nodes define different attribute states. To this decision tree structure, CRG adds another layer, a *part-indexed* tree of features spaces, each with its own attribute-indexed decision tree. With this tree of decision trees, CRG imposes both a limit on the number of attributes that are being considered, and an ordering on the evaluation of attributes.

CRG uses linearly separable attribute bounds for rules or generalizations. Since CRG is part-indexed and not explicitly attribute-indexed, this is not required but has been used in this implementation for comparison purposes. Finally, the computational complexity of CRG is, in principle, identical to decision trees insofar as the attribute testing and splitting procedures are similar. However, the unique *relational aspects* of CRG may or may not result in more efficient learning, depending on the type of learning context.

Recently, Quinlan [26] and Muggleton and Buntine [27] have investigated general methods for learning *symbolic* relational structures in the form of Horn clauses in the following sense. In FOIL, [26] considers the problem of learning, from positive examples (closed world) or positive and negative examples, conjunctions of literals that satisfy

$$C \leftarrow L_1, ..., L_m$$

where C would correspond, in our case, to a class label. FOIL solves such problems by expanding the literals - adding predicates and their variables - to the right-hand-side to maximize the covering of positive instances and to minimize inclusion of negative ones. In this framework, then, CRG is also concerned with generating similar class descriptions of the specific forms:

$$\begin{array}{lcl} C_1^1 & \leftarrow & U^1(X), B^1(X,Y), U^2(Y), B^2(Y,Z), U^3(Z), ... \\ ... & & \\ C_1^{n_1} & \leftarrow & U^1(X), B^1(X,Y), U^2(Y), B^2(Y,Z), U^3(Z), ... \\ ... & & \\ C_m^1 & \leftarrow & U^1(X), B^1(X,Y), U^2(Y), B^2(Y,Z), U^3(Z), ... \\ ... & & \\ C_m^{n_m} & \leftarrow & U^1(X), B^1(X,Y), U^2(Y), B^2(Y,Z), U^3(Z), ... \end{array}$$

However, CRG differs significantly from FOIL in the following ways:
1) the choice of unary U-rules and binary B-rules as bounded attribute (feature) states, is determined within continuous unary and binary feature spaces;
2) the ordering of literals must be *satisfied* in the rule generation;

3) the search technique uses backtracking and recursive splitting and
4) the resultant rules are not only Horn clauses but each literal *indexes* bounded regions in the associated feature space (as shown in Figure 2).

The CRG method is an example of the general solution to complex pattern recognition problems involving the generation of rules, as bounded predicate Horn Clauses, which are linked together in ways that determine "structure" uniquely enough to identify classes but enable generalization to tolerate distortions. Both aims, uniqueness and generalization, are not explicitly guaranteed in other methods, such as neural networks or decision trees. Further, uniqueness and generalization constitute the equivalent of a "cost" function in CRG, and the search technique has been developed to satisfy these constraints.

Finally, CRG raises the question as to what really is a "structural description" of a pattern. CRG simply generates conditional rules that combine an attempt to generalize the pattern definitions in terms of feature bounds and to restrict the description lengths as much as possible. For complex and highly variable training patterns, CRG can generate a large number of rules which can be thought of as a set of *equivalent descriptions* of the pattern structure. It is possible to determine the more frequently occurring paths and associated feature bounds from the cluster tree, if the notion of "commonness" is deemed necessary for a structural description. However, this may not really be a meaningful definition of structure. Rather than producing a singular rule structure, a "structural description" is defined by a *set of rules* that CRG generates from a set of training patterns.

CRG offers a way for automatically generating structural descriptions which enable rapid tree-based search techniques in complex scene data. For this reason it provides a most useful approach to problems in target detection, surveillance and security applications where not all objects in the scene are required to be identified but those which are also require robust description and rapid detection.

Acknowledgments

TMC was supported by a grant from the Australian Research Committee. WFB was supported by grant OGP38251 from the Canadian Natural Sciences and Engineering Council.

References

[1] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM Journal of Computing*, vol. 6, pp. 537–546, 1977.

[2] D. Ballard and C. Brown, *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[3] W. E. L. Grimson, *Object Recognition by Computer.* Cambridge, MA: MIT Press, 1990.

[4] P. Flynn and A. K. Jain, "3D object recognition using invariant feature indexing of interpretation tables," *Computer Vision, Graphics, and Image Processing*, vol. 55, pp. 119–129, 1992.

[5] A. K. Jain and D. Hoffman, "Evidence-based recognition of objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, pp. 783–802, 1988.

[6] T. Caelli and A. Pennington, "An improved rule generation method for evidence-based classification systems," *Pattern Recognition*, vol. 26, pp. 733–740, 1993.

[7] P. Flynn and A. K. Jain, "Three-dimensional object recognition," in *Handbook of Pattern Recognition and Image Processing, Volume 2: Computer Vision* (T. Y. Young, ed.), New York, NY: Academic Press, 1993.

[8] K. Ikeuchi and T. Kanade, "Automatic generation of object recognition programs," *Proceeding of the IEEE*, vol. 76, pp. 1016–1035, 1988.

[9] R. C. Bolles and P. Horaud, "A three-dimensional part orientation system," *The International Journal of Robotics Research*, vol. 5, pp. 3–26, 1986.

[10] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on Computing*, vol. 22, pp. 67–92, 1973.

[11] R. Mohan and R. Nevatia, "Using perceptual organization to extract 3-d structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1121–1139, 1989.

[12] D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," *Artificial Intelligence*, vol. 31, pp. 355–395, 1987.

[13] R. Michalski and R. E. Stepp, "Automated construction of classifications: Conceptual clustering versus numerical taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, pp. 396–409, 1983.

[14] K. C. C. Chan, A. K. Wong, and D. K. Y. Chiu, "Learning sequential patterns for probabilistic inductive prediction," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, pp. 1532–1547, 1994.

[15] W. F. Bischof and T. Caelli, "Learning structural descriptions of patterns: A new technique for conditional clustering and rule generation," *Pattern Recognition*, vol. 27, pp. 1231–1248, 1994.

[16] U. Fayyad and K. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, pp. 87–102, 1992.

[17] E. Barth, T. Caelli, and C. Zetsche, "Image encoding, labelling and reconstruction from differential geometry," *Computer Vision, Graphics and Image Processing*, vol. 55, pp. 428–446, 1993.

[18] M. D. Carmo, *Differential geometry of Curves and Surfaces.* Englewood Cliffs, NJ: Prentice Hall, 1976.

[19] P. Besl and R. Jain, "Invariant surface characteristics for 3D object recognition in range images," *Computer Vision, Graphics and Image Processing*, vol. 33, pp. 33–80, 1986.

[20] A. Rosenfeld and A. Kak, *Digital Picture Processing.* New York, NY: Academic Press, 1982.

[21] N. Yokoya and M. Levine, "Range image segmentation based on differential geometry: A hybrid approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 643–649, 1989.

[22] T. Caelli and A. Dreier, "Variations on the evidenced-based object recognition theme," *Pattern Recognition*, vol. 27, pp. 185–204, 1994.

[23] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, NJ: Prentice Hall, 1988.

[24] J. R. Quinlan, *C4.5 Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1993.

[25] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[26] J. R. Quinlan, "Learning logical definitions from relations," *Machine Learning*, vol. 5, pp. 239–266, 1990.

[27] S. Muggleton and W. Buntine, "Machine invention of first-order predicates by inverting resolution," in *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–352, 1988.

[28] A. Pearce, T. Caelli, and W. F. Bischof, "Rulegraphs for graph matching in pattern recognition," *Pattern Recognition*, vol. 27, pp. 1231–1248, 1994.