Structural Descriptions and Inexact Matching

LINDA G. SHAPIRO, SENIOR MEMBER, IEEE, AND ROBERT M. HARALICK, SENIOR MEMBER, IEEE

Abstract—**In this paper we formally define the structural description of an object and the concepts of exact and inexact matching of two structural descriptions. We discuss the problems associated with a brute-force backtracking tree search for inexact matching and develop several different algorithms to make the tree search more efficient. We develop the formula for the expected number of nodes in the tree for backtracking alone and with a forward checking algorithm. Finally, we present experimental results showing that forward checking is the most efficient of the algorithms tested.**

Index Terms—**Backtracking, forward checking, inexact matching, look-ahead matching, relational homomorphism, relaxation, structural description, tree search.**

I. Introduction

A STRUCTURAL description of an object consists of the descriptions of its parts and their interrelationships. For example, a simple chair is made up of six parts: a back, a seat, and four legs. The back, seat, and legs can sometimes be described as rectangular parallelepipeds with various constraints on their lengths, widths, and depths. The interrelationships between the parts specify how they fit together. For instance, the top of the seat and the front of the back may be at right angles to each other.

The parts of an object can be primitive (nondecomposable) or they may be further broken down into subparts. When the parts of an object are not primitives, the structural description of the object consists of one level of descriptions for each level of subparts. Such a multilevel description is called a *hierarchic* description and is useful for complex objects with many repetitions of parts and subparts.

In this paper we will be concerned only with single-level structural descriptions consisting of a set of primitive parts and their interrelationships. We will formally define the structural description of an object and the concept of a match between two structural descriptions. We will extend the concept of a match to an inexact match and describe and compare several algorithms for inexact matching using a tree search with backtracking alone, with an operation called forward checking, and with an operation called looking ahead. All the ideas in this paper can be further extended to multilevel descriptions using hierarchic structures (see Shapiro and Haralick [14]) and hierarchic relaxation (see Davis [3] and Zucker and Mohammed [18]).

Manuscript received January 2, 1980; revised August 13, 1980. This work was supported by the National Science Foundation under Grants MCS-7923828 and MCS-7919741.

L. G. Shapiro is with the Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061.

R. M. Haralick is with the Departments of Electrical Engineering and Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061.

II. Structural Descriptions and Exact Matching

A *structural description* D of an object is a pair $D = (P, R)$. $P = \{P1, \cdots, Pn\}$ is a set of *primitives*, one for each of the n primitive parts of the object. Each primitive Pi is a binary relation $Pi \subseteq A \times V$ where A is a set of possible *attributes* and V is a set of possible *values*. $R = \{PR1, \cdots, PRK\}$ is a set of *named N-ary relations* over P. For each $k = 1, \cdots, K$. PRk is a pair (NRk, Rk) where NRk is a name for relation Rk, and for some positive integer Mk, $Rk \subseteq P^{Mk}$. Thus, set P represents the parts of an object, and set R represents the interrelationships among the parts. Note that the elements of any relation Rk may include as components primitives, attributes, values, and any symbols necessary to specify the given relationship.

One way that structural descriptions are used is to define *prototype* objects (see Barrow *et al.* [2] and Haralick and Kartus [5]). The structural descriptions of prototype objects are called *stored models* and are used as part of the knowledge base of a recognition system. Such a system inputs *candidate* objects, computes their structural descriptions, and tries to identify each candidate with a stored model. Thus, instead of asking whether two structural descriptions match each other, we will only ask whether a candidate structural description matches a prototype structural description. This one-way matching will be defined in terms of exact matching in this section and in terms of inexact matching in Section III.

Note that the dictionary definition of the verb "match" is "to correspond; to be of corresponding size, shape, color, pattern, etc." [1]. When we speak of two objects matching, we often assume that matching is a symmetric process; A matches B if and only if B matches A. The matching defined in this paper is *not* necessarily a symmetric process. The next few paragraphs describe some figures illustrating which kinds of matching are not necessarily symmetric and which kind is symmetric.

In exact matching, a candidate primitive Cj *matches* a prototype primitive Pi if the binary relation Pi is a subset of the binary relation Cj. Thus, every attribute-value pair (a, v) in the primitive Pi is also an element of the primitive Cj. To define the matching of a candidate relation to a prototype relation, we need the concept of composing a relation with a mapping and the concept of a relational homomorphism.

Let $R \subseteq P^N$ be an N-ary relation over a set P, and h be a

0162-8828/81/0900-0504$00.75

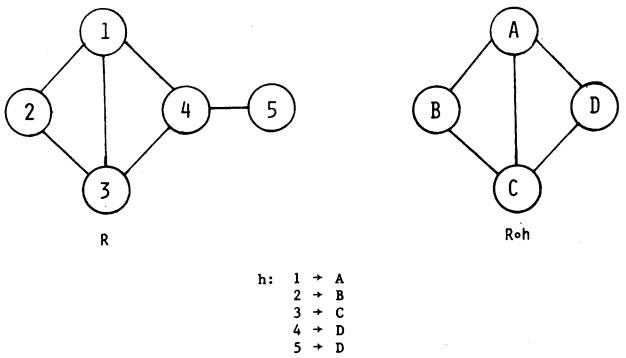

Fig. 1. The composition of binary relation R with mapping h.

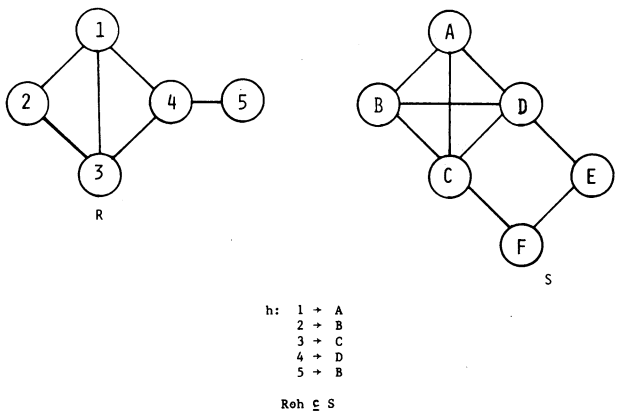

Fig. 2. A relational homomorphism h from binary relation R to binary relation S.

function h: $P \rightarrow Q$ mapping elements of P into a set Q. We define the *composition* $R \circ h$ of R with h by

$$R \circ h = \{(q1, \cdots, qN) \in Q \mid \text{ there exists } (p1, \cdots, pN) \in R \text{ with } h(pi) = qi,\ i = 1, \cdots, N\}.$$

Fig. 1 illustrates the composition of a binary relation with a mapping.

Let $S \subseteq Q^N$ be a second N-ary relation. A *relational homomorphism* from R to S is a mapping h: $P \rightarrow Q$ that satisfies $R \circ h \subseteq S$. That is, when a relational homomorphism is applied to each component of an N-tuple of R, the result is an N-tuple of S. Fig. 2 illustrates the concept of a relational homomorphism.

A relational homomorphism maps the primitives of P to a subset of the primitives of Q having all the same interrelationships that the original primitives of P had. If P is a much smaller set than Q, then finding a one-one relational homomorphism is equivalent to finding a copy of a small object as part of a larger object. Finding a chair in an office scene is an example of such a task. If P and Q are about the same size, then finding a relational homomorphism is equivalent to determining that the two objects are similar. A *relational monomorphism* is a relational homomorphism that is one-one. Such a function maps each primitive in P to a unique primitive in Q. A monomorphism indicates a stronger match than a homomorphism. Fig. 3 illustrates a relational monomorphism.

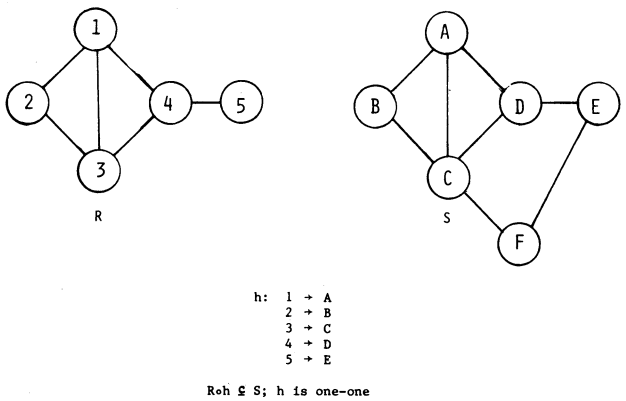

Fig. 3. A relational monomorphism h from binary relation R to binary relation S. There is a copy of R in S.

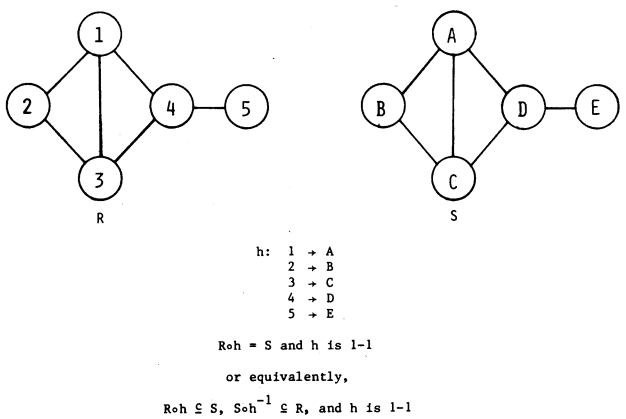

Fig. 4. A relational isomorphism h from binary relation R to binary relation S.

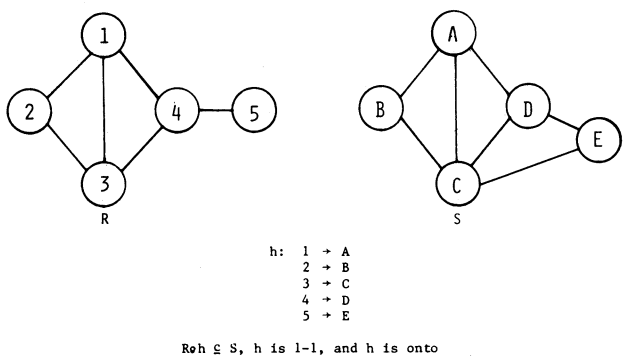

Fig. 5. A relational monomorphism from binary relation R onto binary relation S. This mapping h is not a relational isomorphism since h^{-1} is not a relational monomorphism from S to R.

Finally, a *relational isomorphism* h from an N-ary relation R to an N-ary relation S is a one-one relational homomorphism from R to S, and h^{-1} is a relational homomorphism from S to R. In this case, P and Q have the same number of elements, each primitive in P maps to a unique primitive in Q, and every primitive in Q is mapped to by some primitive of P. Also, every tuple in R has a corresponding tuple in S, and vice versa. An isomorphism is the strongest kind of match: a symmetric match. Fig. 4 illustrates a relational isomorphism,

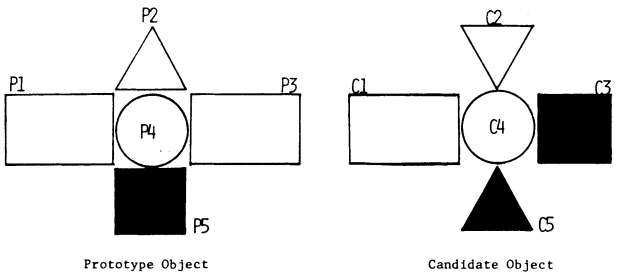

Fig. 6. A prototype object and a candidate object, both made up of primitive parts.

and Fig. 5 shows the difference between a relational isomorphism and a relational monomorphism.

In this paper, we will only require relational homomorphisms for matching, but the reader should realize that algorithms for the monomorphism and isomorphism are essentially identical to the homomorphism algorithms. The only difference is that the incorporation of the stronger constraints will tend to make the algorithms execute quicker.

Now we are ready to define the meaning of an exact match from one structural description to another. First, there must be a function h which gives the correspondence from the primitives of the first description to the primitives of the second description. Second, h must be a relational homomorphism from each relation of the first description to the relation with the same name of the second description. More precisely, let $Dp = (P, R)$ be a prototype structural description and $Dc = (Q, S)$ be a candidate structural description. Let $P = \{P1, \cdots, Pn\}$, $Q = \{Q1, \cdots, Qm\}$, $R = \{(NR1, R1), \cdots, (NRk, Rk)\}$, and $S = \{(NS1, S1), \cdots, (NSk, Sk)\}$. We say that *Dc matches Dp* if there is a mapping h: $P \rightarrow Q$ satisfying

1) $h(Pi) = Qj$ implies $Pi \subseteq Qj$, and

2) $NRi = NSj$ implies $Ri \circ h \subseteq Sj$.

That is, if a relation Ri in Dp has the same name as a relation Sj in Dc, then h, which makes the correspondence from the primitives of the prototype to the primitives of the candidate, must be a relational homomorphism from Ri to Sj.

For example, consider the prototype object and candidate object shown in Fig. 6. Given below are a structural description Dp for the prototype object and a structural description Dc for the candidate object. The parts of the prototype are the primitives $P1$, $P2$, $P3$, $P4$, and $P5$, and the parts of the candidate are the primitives $C1$, $C2$, $C3$, $C4$, and $C5$. The primitives have possible attributes {shape, color} and possible values {rectangular, triangular, circular} for shape and {black, white} for color. Note that more attributes have been measured for the candidate object than for the prototype object. This is because, at the time of measurement, it is not clear what prototype the candidate will match, and the attributes required for several different prototypes may have to be measured. The relations named Left are sets of pairs of the form (x, y) where x is adjacent to and directly left of y. The relations named Above are sets of pairs of the form (x, y) where x is adjacent to and directly above y.

Prototype Description Dp

$Dp = \{P, RP\}$

$P = \{P1, P2, P3, P4, P5\}$

$RP = \{(\text{Left}, \text{Left\_}P), (\text{Above}, \text{Above\_}P)\}$

$\text{Left\_}P = \{(P1, P4), (P4, P3)\}$

$\text{Above\_}P = \{(P2, P4), (P4, P5)\}$

$P1 = \{(\text{shape}, \text{rectangular}), (\text{color}, \text{white})\}$

$P2 = \{(\text{shape}, \text{triangular})\}$

$P3 = \{(\text{shape}, \text{rectangular})\}$

$P4 = \{(\text{shape}, \text{circular})\}$

$P5 = \{(\text{color}, \text{black})\}$

Candidate Description Dc

$Dc = \{C, RC\}$

$C = \{C1, C2, C3, C4, C5\}$

$RC = \{(\text{Left}, \text{Left\_}C), (\text{Above}, \text{Above\_}C)\}$

$\text{Left\_}C = \{(C1, C4), (C4, C3\}$

$\text{Above\_}C = \{(C2, C4), (C4, C5)\}$

$C1 = \{(\text{shape}, \text{rectangular}), (\text{color}, \text{white})\}$

$C2 = \{(\text{shape}, \text{triangular}), (\text{color}, \text{white})\}$

$C3 = \{(\text{shape}, \text{rectangular}), (\text{color}, \text{black})\}$

$C4 = \{(\text{shape}, \text{circular})\}$

$C5 = \{(\text{shape}, \text{triangular}), (\text{color}, \text{black})\}$

The candidate description Dc matches the prototype description Dp via the mapping h: $P \rightarrow C$ given by $h(Pi) = Ci$, $i = 1, \cdots, 5$. Note that the conditions of a match are satisfied even though $C5$ has a different shape than $P5$, and $C3$ has a different color than $P3$. This is because the prototype primitive $P5$ only specifies a color attribute, and the prototype primitive $P3$ only specifies a shape attribute. Note also that $\text{Left\_}P \circ h = \text{Left\_}C$ and $\text{Above\_}P \circ h = \text{Above\_}C$, instead of just satisfying the subset condition. In this case, the candidate is a *homomorphic image* of the prototype.

III. Weighted Prototype Structural Descriptions and Inexact Matching

In a world where there is no observation noise and no random alterations of the entities for any entity class, exact matching of structural descriptions is an appropriate procedure. Unfortunately, in the real world, random structural alterations of entities occur if for no other reason than the fact that observation or measurement of structural relationships has some associated random noise component. Thus, we cannot expect two entities of the same class to have exactly matching structural descriptions.

This naturally leads to the concept of inexact matching. Here we seek matches which are not necessarily perfect, only good enough. The model which is behind the inexact matching assumes that the ideal structural description for an entity is randomly altered. Associated with each possible altered structural description is the probability that such a structural description will result from the random alteration process. We expect that structural descriptions in which there are only a few alterations will have higher probability of occuring than those with many alterations. We might also know that certain structural alterations are less likely to happen than others.

As soon as we admit that the inexactness occurs because of random alteration, we must become sensitive to the fact that the inexact matches we might find might be entirely due to a chance match with an altered structural description for an entity of an entirely different entity class. This kind of event is much more likely to happen with inexact matching than with exact matching.

One way of handling this situation is to compute our confidence in the inexact match, where confidence is measured not on the basis of the inexactness of the match, but by the likelihood ratio whose numerator is the probability that the alteration determined by the inexact match would occur for the structural description of an entity in the class, and whose denominator is the probability that the computed inexact match would arise from just a chance inexact match to a completely random structural description. Thus, the probability model naturally sets up the information required to measure our confidence in the inexact match.

We leave the detailed discussion of the associated probability models to another paper. In this paper we concentrate on giving a precise meaning to the inexactness of an inexact match, keeping in the back of our minds that associated with every value of inexactness will be two probabilities: the probability that the computed inexactness arises from an alteration of the structural description for an entity in the given entity class, and the probability that it arises from inexactly matching a random structural description.

In inexact matching, the parts of the candidate object may not be exactly the same as the parts of the prototype object—in fact, some of them may be badly distorted or missing altogether. Similarly, some of the interrelationships present in the prototype may not hold in the candidate. The problem of distorted parts has been addressed by Tsai and Fu [15]. Since our main concern in this paper is with relationships, we will handle the part matching problem with a simple distance measure. That is, for each attribute a there is a threshold ta by which the value of a in a candidate primitive can differ from the value of a in the corresponding prototype primitive. Thus, a candidate primitive *Cj inexactly matches* a prototype primitive *Pi* if for every pair (a, v) in the prototype primitive *Pi*, there is a pair (a, v') in the candidate primitive *Cj* with $|v - v'| \leqslant ta$. The distance measure is not necessarily numeric and must be defined for each application. Similarly, the thresholds depend on the application and the data.

In handling missing parts and missing relationships, we want to take into account the fact that some parts are more important than others and some relationships are more important than others. We represent this fact by assigning a weight to each part and each N-tuple in the model. This extends our definition of the prototype as follows.

A *weighted prototype structural description* D is a 4-tuple $D = (P, wp, R, WR)$ where $P = \{P1, \cdots, Pn\}$ is a set of primitives as before, and wp is a *primitive-weighting function*, wp: $P \rightarrow [0, 1]$ that assigns a weight to each primitive in P and satisfies $\Sigma_i wp(Pi) = 1$. $R = \{(NR1, R1), \cdots, (NRK, RK)\}$ is again a set of named N-ary relations over P. $WR = \{w1, \cdots, wK\}$ is a set of *N-tuple-weighting functions.* For each $k = 1, \cdots, K$, wk assigns weights to the Mk-tuples of relation Rk. Thus, each wk is a function wk: $Rk \rightarrow [0, 1]$ satisfying $\Sigma_{r \in Rk} wk(r) = 1$. Note that requiring the sum of the weights to be one was an arbitrary choice. Any number will do as long as it is used consistently in all models.

ϵ-Homomorphisms

Since the prototype relations are now weighted, the relational homomorphisms must take these weights into account. Suppose R is an N-ary relation over a set P, w: $R \rightarrow [0, 1]$ is a weighting function for R, and S is an N-ary relation over set Q. Let h be a mapping h: $P \rightarrow Q$ from set P to set Q. An N-tuple r of R is *satisfied* by h with respect to S if $h(r)$ is an element of S. An *ϵ-homomorphism* from R to S with respect to w is a mapping h: $P \rightarrow Q$ such that

$$\sum_{\substack{r \in R \\ h(r) \notin S}} w(r) \leqslant \epsilon.$$

That is, the sum of the weights on those N-tuples that are not satisfied by h with respect to S is less than the threshold ϵ.

The inexact matching problem may now be stated as follows. Let Dp be a weighted prototype structural description, and let Dc be a candidate structural description. Suppose $Dp = (P, wp, RP, WRP)$ where $P = \{P1, \cdots, Pn\}$, $RP = \{(NR1, R1), \cdots, (NRk, Rk)\}$, and $WRP = \{w1, \cdots, wk\}$. Suppose $Dc = (C, RC)$ where $C = \{C1, \cdots, Cm\}$ and $RC = \{(NS1, S1), \cdots, (NSK, SK)\}$. Let A be the set of attributes in P and C, and let V be the set of values for the attributes. Then *Dc inexactly matches Dp* with respect to the attribute-value thresholds $T = \{ta | a \in A\}$, the missing parts threshold tm, and the relation thresholds $E = \{\epsilon i | PRi \in RP\}$ if there is a mapping h: $P \rightarrow C \cup \{\text{null}\}$ that satisfies the following.

1) If $h(Pi) = Cj \in C$, then Cj inexactly matches Pi with respect to T.

2) $$\sum_{\substack{Pi \in P \\ h(Pi) = \text{null}}} wp(Pi) \leqslant tm.$$

3) If $NRi = NSj$, then h is an ϵi-homomorphism with respect to wi from Ri to Sj.

In searching for a match between a prototype object and a candidate object, we are looking for a mapping from the primitives of the prototype to the primitives of the candidate. The mapping must satisfy: 1) that each candidate primitive inexactly matches its corresponding prototype primitive according to a threshold associated with the prototype primitive; 2) that the sum of the weights of those prototype primitives that do not map to a candidate primitive must not exceed another threshold; and 3) that it is an ϵ-homomorphism from each prototype relation to a candidate relation, where the threshold ϵ is associated with the prototype relation.

One idea that we have not mentioned is the concept of a *best match*. A best match is a mapping that somehow minimizes the error incurred. Since for n primitives and k relations in a structural description, there are $n + k + 1$ error measurements involved in an inexact match, the definition of a best match is not immediately obvious. A mapping might incur n errors on one relation and satisfy no N-tuples of a second relation. Or, it might do well in primitive matching and relation matching, but only involve ten percent of the prototype primitives.

The definition of a best match depends on the priorities required for the matching task to be performed. For that reason we will not attempt to define a best match in this paper. However, the reader should note that once the concept of a best match has been defined, there are standard ways of modifying the tree search described in Section IV so that the best match will be found.

IV. Matching Structural Descriptions

The relational homomorphism problem (for 0-homomorphisms or exact matches) has been shown to be a special case of a more general problem called the *consistent labeling problem* (Haralick and Shapiro [8]). The consistent labeling problem is defined as follows.

Let U be a set of objects called *units* and L be a set of objects called *labels*. Let $T \subseteq U^N$ be a *unit constraint relation*. That is, if an N-tuple $(u1, \cdots, uN)$ is an element of T, then the label of one unit ui and the N-tuple is constrained by the labels of the other units in the N-tuple. Let $R \subseteq (U \times L)^N$ be a *unit-label constraint relation*. That is, if an N-tuple $[(u1, l1), \cdots, (uN, lN)]$ [written as $(u1, l1, \cdots, uN, lN)$] is an element of R, then unit $u1$ may have label $l1$, unit $u2$ may have label $l2, \cdots$, and unit uN may have label lN, all simultaneously. The consistent labeling problem is to find a mapping f: $U \rightarrow L$ satisfying if $(u1, \cdots, uN)$ is in T, then $(u1, f(u1), \cdots, uN, f(uN))$ is in R. The 4-tuple (U, L, T, R) is called a *compatibility model*, and f is called a *consistent labeling*.

The relational homomorphism problem fits into this model as follows. Let $Rp \subseteq P^N$ be a relation that is part of the prototype and $Rc \subseteq C^N$ be the corresponding relation in the candidate. If we take $U = P$, $L = C$, $T = Rp$, and $R = Rp \times Rc$, then f is a relational homomorphism from Rp to Rc if and only if f is a consistent labeling with respect to the compatibility model (U, L, T, R). This was proved in Haralick and Shapiro [8].

The general consistent labeling problem and thus the relational homomorphism problem can be solved by a tree search incorporating a look-ahead, forward checking, and/or relaxation operator. In this section, we make the extension to ϵ-consistent labelings and define some look-ahead operators to aid in the problem of finding them. The problem of finding ϵ-homomorphisms will then be solved by finding ϵ-consistent labelings.

Look-Ahead for Inexact Matching

Let (U, L, T, R) be a compatibility model. We will assume that if an N-tuple $(u1, \cdots, uN)$ is an element of T, then T does not contain any permutations of $(u1, \cdots, uN)$. Also, we expect no two components of any N-tuple in T to have the same value.

Let Ew: $T \times L^N \rightarrow [0, 1]$ be a nonnegative function. $Ew(u1, \cdots, uN, l1, \cdots, lN)$ is the error that occurs when the N-tuple $(l1, \cdots, lN)$ of labels is applied to units $(u1, \cdots, uN)$.

The *inexact consistent labeling problem* is to find all mappings h: $U \rightarrow L$ so that the sum of the errors incurred by h on all N-tuples of units that constrain one another is less than a given $\epsilon 0$. That is, we must find all h satisfying

$$\sum_{(u1, \cdots, uN) \in T} Ew(u1, \cdots, uN, h(u1), \cdots, h(uN)) \leqslant \epsilon 0.$$

Note that when $Ew(u1, \cdots, uN, l1, \cdots, lN)$ is defined to be $w(u1, \cdots, uN)$ when $((u1, l1), \cdots, (uN, lN))$ is not an element of R and 0 otherwise (where w is the weighting function discussed in Section III), then the inexact consistent labeling problem is equivalent to the problem of finding ϵ-homomorphisms.

The labeling problem is combinatorial in nature and can be solved by a brute force backtracking tree search. As discussed by Mackworth [10], the backtracking strategy suffers from thrashing behavior. That is, the search fails at several different places in the tree, all for the same reasons. If the reason for failure could be remembered or anticipated, then the tree search could be made more efficient.

In order to illustrate the concepts to be developed in this section, we will use the following continuing example. Suppose $U = \{1, 2, 3, 4\}$, $T = \{(1,2)(1,3)(1,4)(2,3)(2,4)(3,4)\}$, $L = \{A, B, C, D\}$, $N = 2$, $\epsilon = .2$, and $R =$

$$\{(1,B,2,A)(1,B,2,B)(1,C,2,B)(1,C,2,C)(1,D,2,B)$$
$$(1,D,2,C)(1,D,2,D)(1,A,3,B)(1,A,3,D)$$
$$(1,B,3,B)(1,B,3,C)(1,B,3,D)(1,D,3,D)$$
$$(1,A,4,A)(1,B,4,B)(1,C,4,A)(1,C,4,B)$$
$$(1,C,4,C)(1,D,4,B)(2,A,3,C)(2,B,3,A)$$
$$(2,B,3,D)(2,C,3,D)(2,D,3,A)(2,D,3,C)$$
$$(2,A,4,D)(2,B,4,C)(2,C,4,D)(2,D,4,A)$$
$$(2,D,4,B)(3,A,4,D)(3,B,4,A)(3,B,4,D)$$
$$(3,C,4,A)(3,D,4,C)(3,D,4,D)\}.$$

Also assume that $w(u1, u2) = \frac{1}{6}$ for every pair $(u1, u2)$ of units and that $Ew(u1, u2, l1, l2) = \frac{1}{6}$ when $((u1, l1), (u2, l2))$ is not in R and 0 otherwise. The brute-force backtracking tree search generates a tree of 44 nodes. A 22 node portion of the tree for unit 1 having labels A and B is as follows:

```
1A
    2A
    2B
        3D
    2C
        3D
    2D
1B
    2A
        3B
        3C
        3D
    2B
        3A
        3B
        3C
        3D
            4C
    2C
        3D
    2D
        3C.
```

Notice that although label 1 for unit A is impossible in any consistent labeling [since there is no $(1, A, 2, X)$ for any label X in R], the brute force search spends some time thrashing on the $(1, A)$ subtree.

To understand this thrashing behavior better, consider why the tree search could fail without or expecting it to fail. We might not expect it to fail because of our shortsightedness: we have taken into account the error incurred against all past units (those units which have already been assigned labels) but have not taken into account the minimum error that the current labeling must incur against future units or the minimum error that future units have with future units.

To take these errors into account we must divide T into various pieces based upon the set Up of past units which have been assigned labels and the set Uf of future units which have not been assigned labels. T intersect Up^N is the set of all N-tuples composed of units which have already been assigned labels and which, therefore, have an exact error of

$$\sum_{\substack{(u1, \cdots, uN) \in \\ T \text{ intersect } Up^N}} Ew(u1, \cdots, uN, h(u1), \cdots, h(uN)).$$

Suppose in our example that the tree search has assigned label A to unit 1 and label B to unit 2. Then $Up = \{1, 2\}$, and T intersect $Up^2 = \{(1, 2)\}$. Since $(1, A, 2, B)$ is not in R, the exact error accumulated so far is $\frac{1}{6}$.

T intersect Uf^N is the set of all N-tuples composed of units which have not already been assigned labels. Hence, the partial labeling h which is only defined over Up cannot influence or force any errors in T intersect Uf^N. We may take the smallest possible future error due to N-tuples of units in T intersect Uf^N as zero, or if we like a better lower bound, we can use

$$\sum_{\substack{(u1, \cdots, uN) \in \\ T \text{ intersect } Uf^N}} \min_{(l1, \cdots, lN)} Ew(u1, \cdots, uN, l1, \cdots, lN).$$

Back in our example, when unit 1 has label A and unit 2 has label B, then $Uf = \{3, 4\}$, and T intersect $Uf^2 = \{(3, 4)\}$. Then the above expression reduces to the minimum over $l3$ and $l4$ of $Ew(3, 4, l3, l4)$. Since $(3, A, 4, D)$ is in R, the future error is, this time, zero.

T has N-tuples other than those in T intersect Up^N and T intersect Uf^N. For example, there are those N-tuples having $(N - 1)$ units from Up and one unit from Uf. This subset of T will have an associated minimum error that strongly depends on the partial labeling h. We can give an explicit expression for this minimum error if we define the subset $T(u, i; Up)$ of T by

$$T(u, i; Up) = \{(u1, \cdots, uN) \in T \mid ui = u \text{ and } n \neq i \text{ implies } un \in Up\}.$$

Obviously,

$$\bigcup_{u \in Uf} \bigcup_{i=1}^{N} T(u, i; Up)$$

is the set of all N-tuples in T having $(n - 1)$ components being units in Up and one component being some future unit in Uf. Also notice that since no two components of an N-tuple in T can have the same value, $T(u, i; Up)$ intersect $T(u, j; Up)$ is the empty set when $i \neq j$. Hence, for a given future unit u and label l, the quantity

$$\begin{aligned} &\text{epf}(u, l; Up, h) \\ &\quad = \sum_{i=1}^{N} \sum_{(u1, \cdots, uN) \in T(u, i; Up)} Ew(u1, \cdots, u(i-1), \\ &\qquad u, u(i+1), \cdots, uN, h(u1), \cdots, \\ &\qquad h(u(i-1)), l, h(u(i+1)), \cdots, h(uN)) \end{aligned}$$

is the error that the current labeling h on past units in Up causes on future unit u with label l. Should this error be greater than the error budget for future units, label l can be excluded from further consideration.

The smallest error that future unit u can incur given h is $\min_{l \in L} \text{epf}(u, l; Up, h)$. The smallest error that the future units individually incur given the partial labeling h is

$$\sum_{u \in Uf} \min_{l \in L} \text{epf}(u, l; Up, h).$$

Should this error exceed the error budget for future units, then the tree search must either try the next label on the current unit or backtrack. In our example, when unit 1 has label A

and unit 2 has label B, then for future unit 3, $\text{epf}(3, A; \{1,2\}, \{(1,A),(2,B)\}) = Ew(1,3,A,A) + Ew(2,3,B,A)$. The N-tuple $(1,A,3,A)$ is not in R; the N-tuple $(2,B,3,A)$ is in R. The error incurred here is thus $\frac{1}{6} + 0 = \frac{1}{6}$. Similarly, $\text{epf}(3, B; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{6}$, $\text{epf}(3, C; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{3}$, and $\text{epf}(3, D; \{1,2\}, \{(1,A),(2,B)\}) = 0$. Also, $\text{epf}(4, A; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{6}$, $\text{epf}(4, B; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{3}$, $\text{epf}(4, C; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{6}$, and $\text{epf}(4, D; \{1,2\}, \{(1,A),(2,B)\}) = \frac{1}{3}$.

The minimum over $l \in L$ of $\text{epf}(3, l; \{1,2\}, \{(1,A),(2,B)\})$ is 0, and the minimum over $l \in L$ of $\text{epf}(4, l, \{1,2\}, \{(1,A),(2,B)\})$ is $\frac{1}{6}$. Thus, the smallest error that the future units 3 and 4 can incur given the partial labeling $\{(1,A),(2,B)\}$ is $0 + \frac{1}{6} = \frac{1}{6}$. Since we already had an exact error of $\frac{1}{6}$, we now can forsee a total error of at least $\frac{1}{3}$ which is larger than $\epsilon = 0.2$.

There are yet other subsets of N-tuples in T which we have not accounted for and for which the labeling h forces some error. The next one we might consider is that set of N-tuples from T having $(N-2)$ of its components being units in Up and two of its components being units in Uf. To help us give an explicit expression for this error, we define the subset $T(u, i, v, j; Up)$ of T by

$$T(u,i,v,j;Up) = \{(u1,\cdots,uN) \in T \mid ui = u, \quad uj = v, \quad \text{and} \quad n \neq i,j \text{ implies } un \in Up\}.$$

Then considering the unit set to be ordered by the greater than relation $>$,

$$\bigcup_{u \in Uf} \bigcup_{\substack{v \in Uf \\ v > u}} \bigcup_{i=1}^{N} \bigcup_{j=1}^{N} T(u,i,v,j: Up)$$

is precisely the set of all N-tuples in T having $(N-2)$ components being in Up and two components being in Uf. These sets are all mutually exclusive when $u \neq v$. Hence, for a given pair of future unit-label pairs (u, l) and (v, m) the quantity

$$\text{eff}(u,l,v,m;Up,h) \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{(u1,\cdots,uN) \in T(u,i,v,j;Up)} Ew(u1,\cdots, u,\cdots,v,\cdots,uN,h(u1),\cdots,l,\cdots,m,\cdots,h(uN))$$

is the error that the current labeling h on Up causes on the future unit-label pairs (u, l) and (v, m). One lower bound of the error that future units (a pair at a time) incur on each other is

$$\sum_{u \in Uf} \min_{l \in L} \sum_{\substack{v \in Uf \\ v > u}} \min_{m \in L} \text{eff}(u,l,v,m;Up,h).$$

Also the minimal total error any particular future unit-label pair (u, l) incurs against the other future units can be obtained as

$$\sum_{\substack{v \in Uf \\ v \neq u}} \min_{m \in L} \text{eff}(u,l,v,m;Up,h).$$

From this we obtain another lower bound of the error that future units (a pair at a time) incur on one another as

$$\frac{1}{2} \sum_{u \in Uf} \min_{l \in L} \sum_{\substack{v \in Uf \\ v \neq u}} \min_{m \in L} \text{eff}(u,l,v,m;Up,h).$$

Each of these error bounds, as discussed in the next subsection, can be used in the context of the standard backtracking tree search to make it smarter by precomputing, remembering, or anticipating some of the causes for future failures, thereby avoiding them and making the tree search more efficient. Haralick and Elliott [9] discuss the specialization of these ideas to exact relational homomorphism problems arising from binary constraint satisfaction problems.

V. Tree Searching

In this section we discuss some different algorithms for tree searching that find inexact matches by determining ϵ-homomorphisms.

A. Backtracking

In the standard backtracking approach, each partial labeling h defined on the set of past units Up incurrs an error $ep(Up, h)$, where

$$ep(Up,h) = \sum_{\substack{(u1,\cdots,uN) \in \\ T \text{ intersect } Up^N}} Ew(u1,\cdots,uN,h(u1),\cdots,h(uN)).$$

If at any time in the tree search the error incurred by this partial labeling exceeds the error budget, the tree search must either try the next label for the current unit or if there is no next label, it must backtrack.

B. Forward Checking

Forward checking proceeds in a manner similar to backtracking. But it recognizes that in addition to the error $ep(Up, h)$ which the partial labeling h incurs against the past units Up, the partial labeling h commits the past units with their assigned labels from h to have a minimum error with the future units Uf. By doing some forward checking, letting the past units with their assigned labels broadcast to each future unit-label pair this incurred error, it becomes easy to keep track of a lower bound for the error the past units must have with the future units. Recall that $\text{epf}(u, l; Up, h)$ is the total error accumulated by future unit-label pair (u, l) from all the past units in Up with their assigned labels from h. If for any label l for current unit u, the quantity

$$ep(Up,h) + \text{epf}(u,l;Up,h) + \sum_{\substack{v \in Uf \\ v \neq u}} \min_{m \in L} \text{epf}(v,m;Up,h)$$

exceeds the error budget, then forward checking fails and we must either try the next label for the current unit or backtrack.

In our continuing example, the portion of the tree searched for unit 1 having labels A and B with forward checking is

1A
1B
2A
2B
3D
4C
2D.

Notice that the 22 nodes processed with the brute force backtracking search has been reduced to 7 nodes.

C. Looking Ahead by One

Looking ahead by one proceeds in a manner similar to forward checking. But it recognizes that in addition to the minimum error that a partial labeling creates by past units against past units and past units against future units, there is some minimum error of future units against future units. We called $\text{eff}(u, l, v, m; Up, h)$ the error that future unit-label pair (u, l) has with future unit label pair (v, m) taking into account that past units in Up must have the labels assigned to them by h. Then the minimum error that a future unit-label pair (u, l) incurs with the future units (taken one at a time) is

$$\sum_{\substack{v \in Uf \\ v \neq u}} \min_{m \in L} \text{eff}(u, l, v, m; Up, h).$$

If for any unit-label pair (u, l) the quantity

$$ep(Up, h) + \sum_{v \in Uf} \min_{m \in L} \text{epf}(v, m; Up, h) + \sum_{\substack{v \in Uf \\ v \neq u}} \min_{m \in L} \text{eff}(u, l, v, m, Up, h)$$

exceeds the error budget, then the pair (u, l) can be dropped from consideration as a possible participant in the extension of labeling h. This idea may be applied iteratively, whereupon it becomes a weighted discrete relaxation operator, the natural generalization of the discrete relaxation operator originally defined by Ullman [16], independently rediscovered by Waltz [17], and also discussed in Rosenfeld *et al.* [12], Haralick *et al.* [6], Haralick [7], Haralick and Shapiro [8], and Gaschnig [4].

After the current unit has had a label instantiated for it and has become a past unit, another lower bound can be computed for the future error. We have already observed that the smallest error future units can have with future units taken one at a time given the partial labeling h on Up is

$$\sum_{u \in Uf} \min_{l \in L} \sum_{\substack{v \in Uf \\ v > u}} \min_{m \in L} \text{eff}(u, l, v, m; Up, h).$$

Hence, looking ahead by one also uses the quantity

$$ep(Up, h) + \sum_{u \in Uf} \min_{l \in L} \text{epf}(u, l; Up, h) + \sum_{u \in Uf} \min_{l \in L} \sum_{\substack{v \in Uf \\ v > u}} \min_{m \in L} \text{eff}(u, l, v, m; Up, h)$$

in the error budget check. If this quantity exceeds the error budget, looking ahead by one fails and we must either try the next label for the current unit or backtrack.

In our example, the portion of the tree searched for unit 1 having labels A and B with look-ahead by one is

1B
2B
3D
4C.

Thus, look-ahead by one, in this small problem, eliminated backtracking and thrashing entirely.

D. Looking Ahead by Two

Looking ahead by two does the same sort of thing done by looking ahead by one, but, in addition, it takes into account the minimum error incurred by a pair of future unit-label pairs as they look ahead to other future units. Recall that the error incurred by future unit-label pair (u, l) against future unit-label pair (x, q) is computed by $\text{eff}(u, l, x, q; Up, h)$. For a given pair $((u, l), (v, m))$ of future unit-label pairs, the best label q that another future unit x can have is one which minimizes $\text{eff}(u, l, x, q; Up, h) + \text{eff}(v, m, x, q; Up, h)$. The smallest error that $((u, l), (v, m))$ can incur on all future units including itself is then

$$\text{eff}(u, l, v, m; Up, h) + \sum_{\substack{x \in Uf \\ x \neq u, v}} \min_{q \in L} [\text{eff}(u, l, x, q; Up, h) + \text{eff}(v, m, x, q; Up, h)].$$

Whenever the above quantity plus the quantity

$$ep(Up, h) + \sum_{x \in Uf} \min_{q \in L} \text{epf}(x, q, Up, h)$$

exceeds the error budget, looking ahead by two may throw out the pair of unit-label pairs $((u, l), (v, m))$. Thus, looking ahead by two needs a two-dimensional unit label table. Note that before looking ahead by two performs its look-ahead by two, it performs the look-ahead by one. If that succeeds, it then performs the look-ahead by two. Applied iteratively, this becomes a weighted discrete relaxation operator, the natural generalization of the operator used by Montanari [11] and the Φ_2 operator of Haralick *et al.* [6].

In this section we have described several look-ahead or relaxation operators to be used in conjunction with a tree search to find ϵ-homomorphisms. We have defined the operators based on only a single relation. These operators can and should be used cooperatively when more than one relation is involved. For further information on cooperative calculation, see Zucker [18] and Davis [3].

VI. Performance Evaluation

Our past experiments in inexact matching have been in shape matching (Shapiro [13]). In these experiments, a SNOBOL4 program found homomorphisms from a pair of ternary relations representing a prototype shape to a second pair of ternary relations representing a candidate shape. In order to more thoroughly test our inexact matching algorithms, we have developed a statistical model that allows us to generate random binary relation consistent labeling problems and a set of criteria on which to compare the performance of the algorithms in finding ϵ-consistent labelings. Haralick and Elliott [9] used a similar model to explore the behavior of various algorithms for finding exact or zero-consistent labelings.

In this section we will define the criteria used and discuss the generation of the random problems. We will also use an appropriate random model to develop the expected number of nodes per level in the tree search. Finally, we will describe the experimental results.

A. Criteria for Evaluating Search Algorithms

In Section IV we discussed the brute force tree search with backtracking that finds ϵ-consistent labelings. The algorithms for the tree search with forward checking and with look-ahead by one have been implemented and are given in Appendix I. The following terminology refers to these algorithms.

A *consistency check* for binary relations is the operation that determines if a pair $((u1, l1), (u2, l2))$ is an element of the unit-label constraint relation. All three of the algorithms require consistency checks. A *back check* is a consistency check performed in the context of straight backtracking. Back checks are the only consistency checks performed by the straight backtracking algorithm. No back checks are performed by the forward checking or look-ahead by one algorithms. A *look-ahead* is a consistency check performed in the context of forward checking or look-ahead by one. Look-aheads are executed in routine UPDATE for forward checking and in routine PSI for look-ahead by one. The straight backtracking tree search, of course, does no looking ahead.

The forward checking and look-ahead by one algorithms use a table to keep track of accumulated error. The table, referred to as ULTAB in the algorithms, is actually a stack of tables, one for each level in the tree. At the current level ULTAB(U, L) gives the error accumulated by the forward checking and look-ahead by one operations for unit U and label L. Labels whose accumulated error for a given unit is too high are no longer eligible labels for that unit. A *lookup* is a table lookup performed in the context of forward checking or look-ahead by one. Lookups are counted both when adding information and retrieving information from ULTAB. Finally, the term *node* refers to a node of the tree in the tree search and represents the operation of assigning a particular label to a unit. The criteria measured by the program include number of consistency checks, number of back checks, number of look-aheads, number of lookups, and number of nodes in the tree. These quantities can be measured for the entire tree and for each level in the tree. We also recorded the time to perform a tree search although this is machine and language dependent. The time is, of course, highly correlated with total number of consistency checks.

B. Generation of Random Consistent Labeling Problems

We generated random consistent labeling problems to use in thoroughly testing the search algorithms. In this section we define the statistical model for generating consistent labeling problems.

Let N be the number of units, L be the number of labels, and ϵ be the inexact matching threshold of the problems to be generated. We will assume that all pairs of units $(u1, u2)$ with $u1 \neq u2$ constrain one another, and that if $((u1, l1), (u2, l2))$ is an element of the unit-label constraint relation, so is $((u2, l2), (u1, l1))$. Thus, the unit constraint relation has, effectively, $N(N-1)/2$ unit pairs. We assign each such unit pair an equal weight of $2/(N(N-1))$. That is, for each unit pair $(u1, u2)$, $w(u1, u2) = 2/(N(N-1))$, and $Ew(u1, u2, l1, l2) = w(u1, u2)$ if $((u1, l1), (u2, l2)) \notin R$ and 0 otherwise.

The generation of the unit-label constraint relation R is based on the assumption that the probability that a given consistency check succeeds is independent of the pair of units or labels involved and independent of whatever labels may already have been assigned to past units in the tree search. That is,

1) $P(((u, l), (u', l')) \in R)$
$= P(((v, m), (v', m')) \in R)$, and

2) $P(((u(K+1), l(K+1)), (u, l)) \in R \mid l1, \cdots, lK$ are consistent labels of $u1, \cdots, uk)$
$= P(((u(K+1), l(K+1)), (u, l)) \in R)$
for every unit u and label l.

Given that every possible element of R is equally probable, we can use a random number generator to determine which pairs of the form $((u, l), (u', l'))$, $1 \leq u, u' \leq N$, $1 \leq l, l' \leq L$, are elements of R and which are not. Let the parameter p specify what percentage of these possible pairs are actually elements of R. For example, if $N = 8$, $L = 8$, $\epsilon = 0.08$, and $p = 0.4$, then there are $8 * 7/2 = 28$ pairs in the unit constraint relation, the weight of each pair is 1/28 or 0.0357, and the generated unit-label constraint relation will contain 40 percent of all possible elements, randomly chosen. Since ϵ is 0.08, an ϵ-consistent labeling may have zero errors (the sum of the weights of the unsatisfied unit constraints = 0.0), one error (the sum of the weights of the unsatisfied unit constraints = 0.0357), or two errors (the sum of the weights of the unsatisfied unit constraints = 0.0714). However, if there are three errors, then the sum of the weights of the unsatisfied unit constraints is 0.1061 which is greater than $\epsilon = 0.08$. Thus, a labeling with three errors is not a 0.08 consistent labeling.

C. Expected Number of Nodes in the Tree for Backtracking and for Forward Checking

1) Backtracking: At level K, K units have been assigned labels. With L possible labels per unit there are L^K different functions that assign labels to the K units. We need to deter-

mine the probability that any of these L^K labelings is successful through level K. Successful means that there are no more than M consistency tests that fail for the labeling.

Since at level K, a labeling must have passed $K(K-1)/2$ consistency tests, the maximum number of consistency tests that a labeling could fail and yet still succeed as a labeling is $\min\{M, K(K-1)/2\}$. Now, for any number of failures m, $0 \leq m \leq \min\{M, K(K-1)/2\}$, the probability that m tests have failed out of the $K(K-1)/2$ performed is

$$\binom{K(K-1)/2}{m} p^{K(K-1)/2-m}(1-p)^m.$$

The probability that $\min\{M, K(K-1)/2\}$ or fewer tests have failed is

$$\sum_{m=0}^{\min\{M, K(K-1)/2\}} \binom{K(K-1)/2}{m} p^{K(K-1)/2-m}(1-p)^m.$$

Hence, the expected number of labelings from the L^K possible labelings that will have succeeded is

$$L^K \sum_{m=0}^{\min\{M, K(K-1)/2\}} \binom{K(K-1)/2}{m} p^{K(K-1)/2-m}(1-p)^m.$$

2) Forward Checking: For a labeling to have succeeded through level K in forward checking, the labeling must have succeeded in the sense of backtracking. It also must succeed in the following sense. Let Nf be the number of failures committed by the labels of the past units with themselves. Thus, $Nf = \#\{(u, v) | u, v \in Up$ and $(u, f(u), v, f(v))$ is not in $R\}$. Let Mf be the sum of the smallest number of failures some label for each future unit has with the labels of the past units. Then Nf plus Mf must be within the failure tolerance. To compute this probability, we distribute the total number M of allowed failures in all possible ways among the past and future units.

We allow m failures for the past units and $F(k)$ failures for the kth future unit, $k = K+1, \cdots, N$. As before the probability of exactly m failures in the past units is

$$\binom{K(K-1)/2}{m} p^{K(K-1)/2-m}(1-p)^m.$$

The probability of $F(k)$ or more failures in K consistency tests for a label of the kth future unit is

$$\sum_{n=F(k)}^{K} \binom{K}{n} (1-p)^n p^{K-n}.$$

The probability that all L labels fail $F(K+k)$ or more consistency tests out of K tests for each label is

$$\left[\sum_{n=F(k)}^{K} \binom{K}{n} (1-p)^n p^{K-n}\right]^L.$$

Then the probability that the smallest number of tests failed by some label is exactly $F(k)$ is

$$\left[\sum_{n=F(k)}^{K} \binom{K}{n} (1-p)^n p^{K-n}\right]^L - \left[\sum_{n=F(k+1)}^{K} \binom{K}{n} (1-p)^n p^{K-n}\right]^L.$$

Also, the probability $Q(F(K+1), \cdots, F(N))$ that future units $K+1$ through N have their best labels fail exactly $F(K+1), \cdots, F(N)$ times is $Q(F(K+1), \cdots, F(N))$ equals

$$\prod_{k=K+1}^{N} \left(\left[\sum_{n=F(k)}^{K} \binom{K}{n} (1-p)^n p^{K-n}\right]^L - \left[\sum_{n=F(k+1)}^{K} \binom{K}{n} (1-p)^n p^{K-n}\right]^L\right).$$

Therefore, the expected number of labeling to succeed through level K is

$$L^K \sum_{m=0}^{\min\{M, K(K-1/2\}} \binom{K(K-1)/2}{m} \cdot p^{K(K-1)/2-m}(1-p)^m \sum_{F(k+1)=0}^{\min\{K, M-m\}} \cdots \sum_{F(N)=0}^{\min\{K, M-m-F(K+1)-\cdots-F(N-1)\}} \cdot Q(F(K+1), \cdots, F(N)).$$

3) Experimental Results Comparing the Three Search Methods: In comparing backtracking alone, backtracking plus forward checking, and backtracking plus look-ahead by one, we looked at the number of consistency checks, the number of nodes, and the execution time for a tree search. In general, we found that backtracking plus forward checking had the least number of consistency checks and the least time, backtracking plus look ahead by one was next, and backtracking alone had the highest number of consistency checks and the most time. Fig. 7 shows the total number of consistency checks as a function of number of units for the three different search algorithms with $p = 0.5$ and $\epsilon = 0.1$. Fig. 8 shows the time in milliseconds on an IBM 370/158 of number of units. The times are, of course, dependent on the machine, the language, and the compiler. In this set of experiments, the number of labels was the same as the number of units and each data point shows the average result of five trials.

With respect to the size of the portion of the tree actually searched, we found that backtracking alone searched the most nodes, backtracking with forward checking was next, and backtracking with look-ahead by one searched the fewest nodes. Fig. 9 shows the number of nodes searched as a function of level in the tree for problems with eight units, eight labels, $p = 0.5$, and $\epsilon = 0.1$. Thus, the forward checking and looking ahead by one beat the straight backtracking in number of consistency checks, time, and number of nodes. The looking ahead by one beat the forward checking in number of nodes searched, but used many more consistency checks (and therefore time) to do so. This would indicate that, as was the

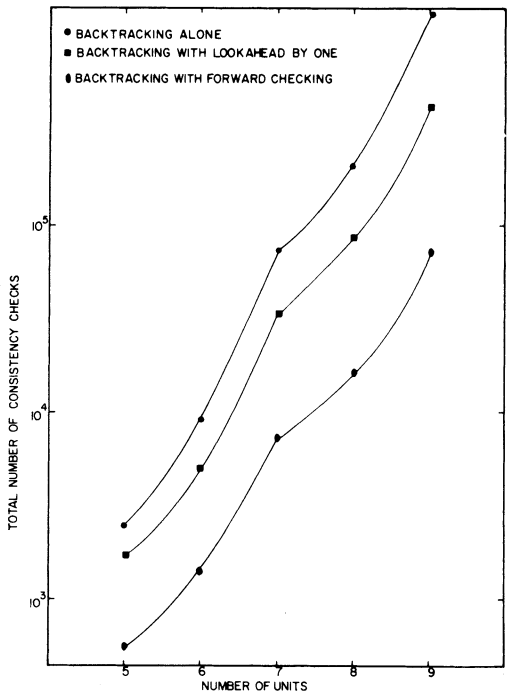

Fig. 7. The number of consistency checks as a function of number units for $p = 0.5$, $\epsilon = 0.1$, and three different search methods.

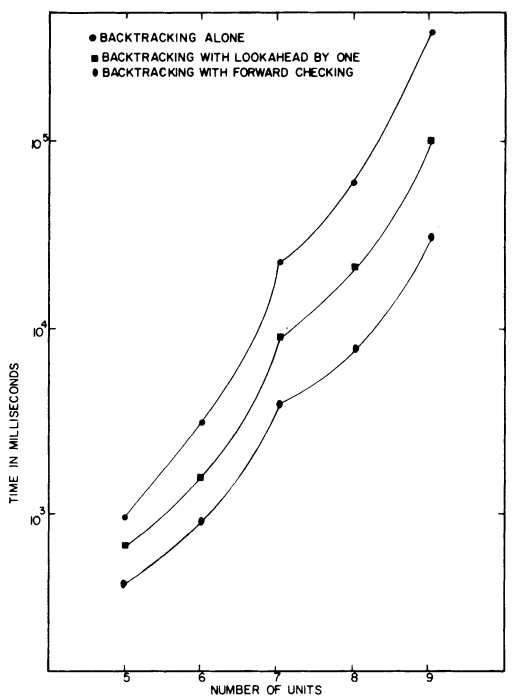

Fig. 8. The number of milliseconds of CPU time on an IBM 370/158 as a function of number units for $p = 0.5$, $\epsilon = 0.1$, and three different search methods.

case for exact matching (Haralick and Elliott [9]), in inexact matching, forward checking is the most efficient of the three methods of search.

4) Optimizing Tree Search Order: In [9] Haralick and Elliot showed that, for exact matching, the number of consistency checks could be minimized by ordering the tree search so that the units most likely to fail are done first. A unit is

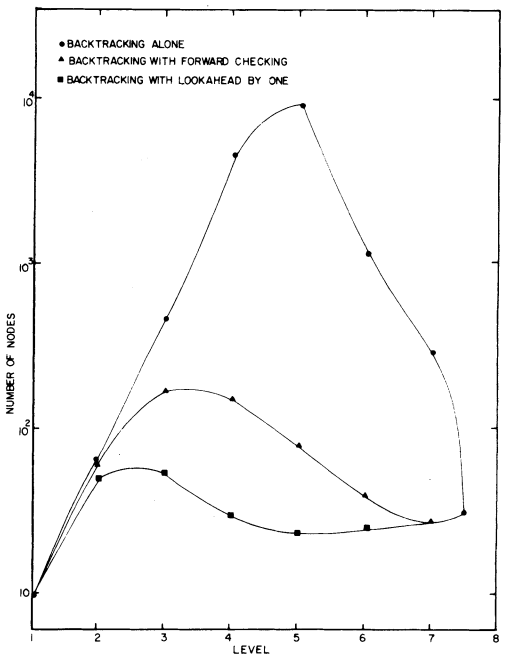

Fig. 9. The number of nodes processed as a function of level in the tree for $N = 8$, $L = 8$, $p = 0.5$, $\epsilon = 0.1$, and three different search methods.

likely to fail if most of the labels have been ruled out for it. The forward checking routine keeps track for each future unit of the number of labels that have not yet been ruled out for that unit. At each level, the tree search procedure decides which unit to try at that level by choosing the unit with the smallest number of labels left. As in the exact matching experiments, we found that ordering the search in this manner did cut the number of consistency checks by a small amount. Fig. 10 shows the comparison of number of consistency checks as a function of number of units for $p = 0.5$, $\epsilon = 0.1$, backtracking with forward checking, and backtracking with forward checking plus ordering by worst unit first.

5) Counting Table Lookups and Look Aheads: The forward checking and the look ahead by one operators use a table to keep track of the status of each possible label for each future unit. The table, ULTAB, works as follows. ULTAB(i,j) holds the error so far accumulated for future unit i and label j. Initially, ULTAB(i,j) is set to 0.0 for all units i and labels j. When a label l is assigned to a unit u, this assignment affects all the future units that do not yet have labels. For each future unit i and label j where $((u,l),(i,j))$ is not an element of the unit-label constraint relation, ULTAB(i,j) is incremented by the weight of (u,i) [in our program $2/(N(N-1))$]. Whenever the error of the labeling so far plus the minimum possible error for all future units plus ULTAB(i,j) becomes greater than the error threshold ϵ, then label j is no longer a possible label for unit i. In this case, NUM(i), also considered a part of the table, is decremented by one to indicate one less possible label for unit i. Finally, the minimum error of all labels for unit i is stored in MINERR(i) and also considered part of the table.

Whenever the table is accessed, either for a store or a fetch, the number of lookups is incremented by one. In forward

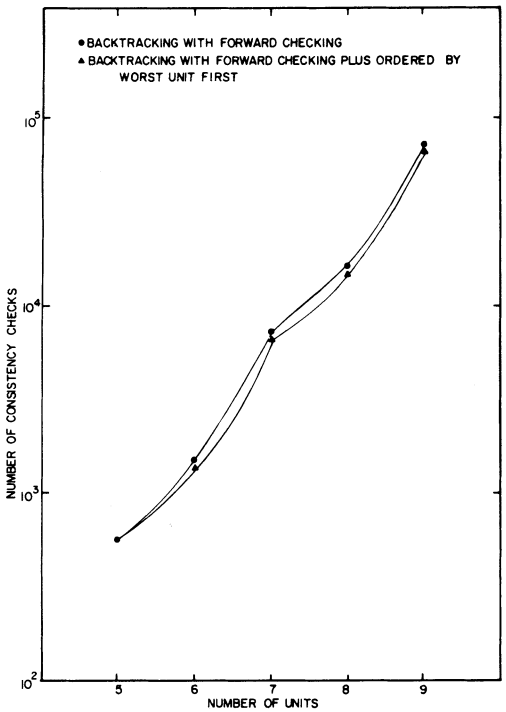

Fig. 10. Ordering the tree search by worst unit first reduces the total number of consistency checks ($p = 0.5$, $\epsilon = 0.1$).

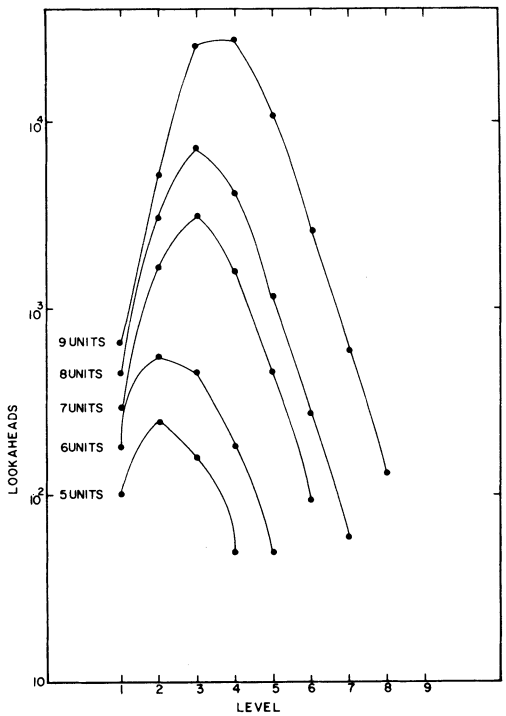

Fig. 12. Look aheads as a function of level for $p = 0.5$, $\epsilon = 0.1$, and backtracking.

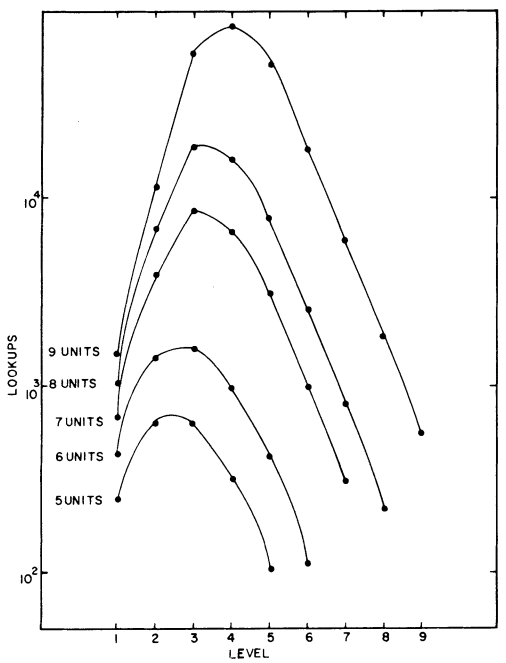

Fig. 11. Lookups as a function of level for $p = 0.5$, $\epsilon = 0.1$, and backtracking with forward checking.

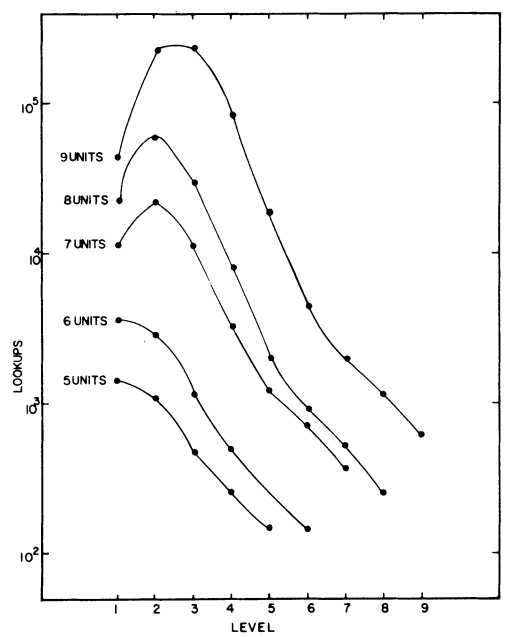

Fig. 13. Lookups as a function of level for $p = 0.5$, $\epsilon = 0.1$, and backtracking with look ahead by one.

checking, for a node at level K where there are $N - K$ future units, the program counts $1 + 2 * (N - K) * (L + 1)$ lookups per completely processed node. For the same node, the program counts $(N - K) * L$ look aheads (consistency checks during forward checking). For look ahead by one, the program performs an additional $(N - K) * L * (2 + (N - K - 1) * L)$ lookups and $(N - K) * L * (N - K - 1) * L$ look aheads. Fig. 11 illustrates lookups as a function of level in the tree, and Fig. 12 illustrates look aheads as a function of level for 5, 6, 7, 8, and 9 units, $p = 0.5$, $\epsilon = 0.1$, and forward checking. Fig. 13 illustrates lookups as a function of level in the tree, and Fig. 14 illustrates look aheads as a function of level in the tree for 5, 6, 7, 8, and 9 units, $p = 0.5$, $\epsilon = 0.1$, and look ahead by one.

6) The Size of the Problem as a Function of Error: Because forward checking proved to be the most efficient search method, we ran a separate series of experiments for forward check-

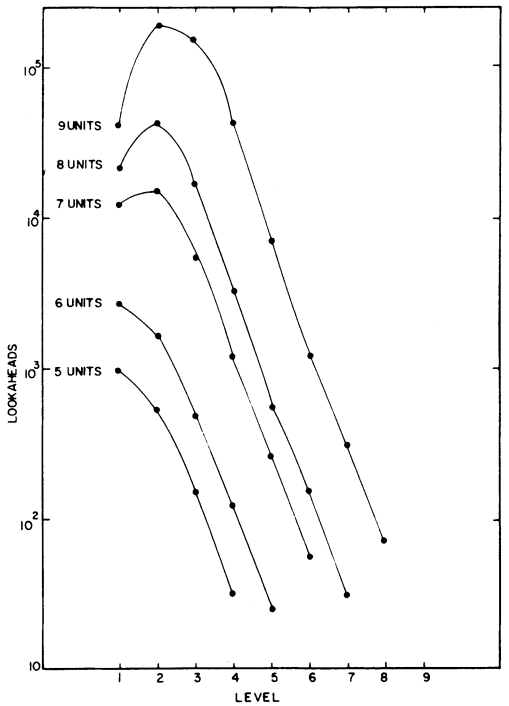

Fig. 14. Look aheads as a function of level for $p = 0.5$, $\epsilon = 0.1$, and backtracking with look ahead by one.

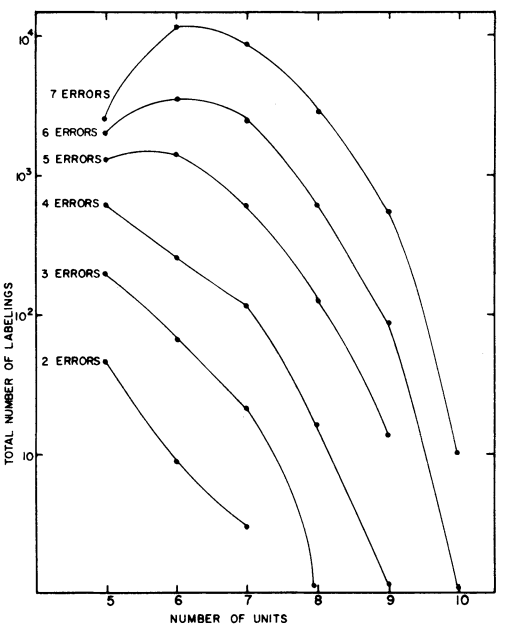

Fig. 15. Total number of labelings as a function of number of units for $p = 0.4$, forward checking, and ϵ varied to give 2, 3, 4, 5, 6, and 7 errors.

ing only. In these experiments we varied the error allowance while holding everything else constant. In order to better compare these results for 5, 6, 7, 8, 9, and 10 units, we varied ϵ in terms of the number of pairs of the unit constraint relation that could fail to be satisfied. We counted one error for each pair of the unit constraint relation that was not satisfied in a given labeling. We varied ϵ so that for each number of units, we could study number of labelings with 1, 2, 3, 4, 5, 6, and 7 errors. Fig. 15 illustrates the total number of labelings for 5, 6, 7, 8, 9, and 10 units each with 2, 3, 4, 5, 6, and 7 errors for $p = 0.4$ and backtracking with forward checking. We start at two errors because, for $p = 0.4$, there are no labelings in some cases with 0 and 1 errors. These results indicate that the inexact consistent labeling problem involves much more work as ϵ gets larger.

VII. Conclusion

We have defined the concept of an inexact match of a candidate structural description to a prototype description and have shown that inexact matching is a special case of the inexact consistent labeling problem. We have discussed the problems involved in finding ϵ-consistent labelings and have described and analyzed four methods: tree search with backtracking alone, tree search with backtracking and forward checking, tree search with backtracking and look-ahead by one, and tree search with backtracking and look-ahead by two. We have given high-level algorithms for the first three methods.

In order to test the algorithms, we have developed a statistical model that allows us to generate random binary relation ϵ-consistent labeling problems. Our experiments indicate that with respect to the number of consistency checks and time, forward checking was best, looking ahead by one next, and straight backtracking worst. Looking ahead by one searches less nodes of the tree than forward checking, but uses many more operations to do so. The extra operations include consistency checks and table lookups. We have shown how these vary as a function of level in the tree. The number of table lookups is greater than, but appears to be proportional to, the number of consistency checks.

We have developed formulas for the expected tree size for forward checking versus straight backtracking and found that the experimental results correspond closely to the theoretical results. Finally, we have studied the size of the problem as a function of the amount of error allowed. Our results show that the inexact consistent labeling problem, and therefore inexact matching, is a much harder problem than the exact version.

Appendix I

Simplified Backtracking Tree Search with Forward Checking

```
comment ULTAB and MINERR are actually stacks, one table per level;
CONTROL := forward;
while CONTROL = forward or some units have been assigned labels
do
   begin
      if all units have a label then CONTROL := back;
```

```
        if CONTROL = back then back up one level;
        U := next unit to try;
        CONTROL := back;
        while there are labels to try for unit U do
           begin
              L := next label for U;
              PERR := error of partial labeling so far;
              BERR := FORER(ULTAB, U, L)
              FERR := FUTMIN(future units)
              if PERR + BERR + FERR ≤ ε then
                 begin
                    ERRF := UPDATE(ULTAB, U, L, PERR + BERR);
                    if UPDATE fails then (try) next (label);
                    CONTROL := forward;
                    add (U, L) to the partial labeling;
                    if all units have labels then print the labeling;
                    move forward one level;
                    exit
                 end
           end
     end

  procedure FUTMIN(future units);
  FUTMIN := 0;
  for each future unit UF do
     FUTMIN := FUTMIN + MINERR(UF)
  end FUTMIN

  procedure UPDATE(ULTAB, U, L, PASTERR);
  UPDATE := 0;
  for each future unit UF do
     begin
        SMALLERR := 99999.;
        for each label LF that is still eligible for UF do
           begin
              if (U, L, UF, LF) is in the unit-label constraint relation
              then ERR := 0
              else ERR := WEIGHT(U, UF);
              ULTAB(UF, LF) := ULTAB(UF, LF) + ERR;
              if ULTAB(UF, LF) < SMALLERR
              then SMALLERR := ULTAB(UF, LF)
           end;
        UPDATE := UPDATE + SMALLERR
        if (UPDATE + PASTERR > ε) then fail return;
        MINERR(UF) := SMALLERR
     end
  end UPDATE
```

SIMPLIFIED BACKTRACKING TREE SEARCH WITH LOOK-AHEAD BY ONE

```
  comment ULTAB and MINERR are actually stacks, one table per level;
  CONTROL := forward;
  while CONTROL = forward or some units have been assigned labels
  do
     begin
        if all units have a label then CONTROL := back;
        if CONTROL = back then back up one level;
        U := next unit to try;
        CONTROL := back;
        while there are labels to try for unit U do
```

```
            begin
               L := next label for U;
               PERR := error of partial labeling so far;
               BERR := FORER(ULTAB, U. L);
               FERR := FUTMIN(future units);
               if PERR + BERR + FERR ≤ ε then
                  begin
                     ERRF := UPDATE(ULTAB, U, L, PERR + BERR)
                     if UPDATE fails then (try) next (label);
                     ERR1 := PSI(ULTAB, U, L, PERR + BERR, ERRF)
                     if PSI fails then (try) next (label);
                     CONTROL := forward;
                     add (U, L) to the partial labeling;
                     if all units have labels then print the labeling;
                     move forward one level;
                     exit
                  end
            end
      end

procedure PSI(ULTAB, U, L, PASTERR, FUTERR);
PSI := 0;
for each future unit UF do
   begin
      UFSMALLERR := 99999.;
      for each label LF that is still eligible for UF do
         begin
            SUMV := 0;
            for each future unit VF ≠ UF do
               begin
                  VFSMALLERR := 99999.;
                  for each label MF that is still eligible for VF do
                     begin
                        if (UF, LF, VF, MF) is in the unit-label constraint relation
                        then ERR := 0
                        else ERR := WEIGHT(UF, VF);
                        if ERR + ULTAB(UF, LF) + ULTAB(VF, MF) +
                           PASTER + (FUTERR - MINERR(UF) -
                           MINERR(VF)) > ε then (try) next (label);
                        if ERR < VFSMALLERR
                        then VFSMALLERR := ERR
                     end
                  SUMV := SUMV + VFSMALLER;
                  if SUMV + PASTERR + FUTERR > ε
                  then begin
                           ULTAB(UF, LF) := infinity;
                           exit
                     end
               end;
            if SUMV < UFSMALLER then UFSMALLERR := SUMV
         end;
      PSI := PSI + UFSMALLERR/2;
      if PSI + PASTERR + FUTERR > ε
      then fail return
   end;
end PSI
```

REFERENCES

[1] C. L. Barnhart, Ed., *The American College Dictionary.* New York: Random House, 1958.

[2] H. G. Barrow, A. P. Ambler, and R. M. Burstall, "Some techniques for recognizing structure in pictures," in *Frontiers of Pattern Recognition*, S. Watanabe, Ed. New York: Academic, 1972, pp. 1–29.

[3] L. S. Davis, "Hierarchical relaxation for shape analysis," in *Proc. IEEE Conf. Pattern Recog. Image Processing*, 1978, pp. 275–279.

[4] J. Gasching, "A general backtrack algorithm that eliminates most redundant tests," in *Proc. 5th Int. Joint Conf. Artificial Intell.*, 1972, p. 457.

[5] R. M. Haralick and J. Kartus, "Arrangements, homomorphisms, and discrete relaxation," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-8, pp. 600–612, Aug. 1978.

[6] R. M. Haralick, L. S. Davis, A. Rosenfeld, and D. Milgram, "Reduction operations for constraint satisfaction," *Informat. Sci.*, vol. 14, pp. 199–219, 1978.

[7] R. M. Haralick, "Scene analysis, homomorphisms, and arrangements," in *Computer Vision Systems*, A. Hanson, and E. Riseman, Eds. New York, Academic, 1978.

[8] R. M. Haralick and L. G. Shapiro, "The consistent labeling problem: Part 1," *IEEE Trans. Pattern. Anal. Machine Intell.*, vol. PAMI-1, Apr. 1979, pp. 173–184.

[9] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," in *Proc. 6th Int. Joint Conf. Artificial Intell.*, 1979.

[10] A. Mackworth, "Consistency in networks of relations," *Artificial Intell.*, vol. 8, pp. 99–118, 1977.

[11] U. Montanari, "Networks of constraints: Fundamental properties and applications to picture processing," *Informat. Sci.*, vol. 7, pp. 95–132, 1974.

[12] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-6, pp. 420–433, June 1976.

[13] L. G. Shapiro, "A structural model of shape," *IEEE Trans. Pattern Anal. Machine Intell.*, 1980.

[14] L. G. Shapiro and R. M. Haralick, "A general spatial data structure," in *Proc. IEEE Conf. Pattern Recog. Image Process.*, Chicago, IL, May 31–June 2, 1978, pp. 238–289.

[15] W. H. Tsai and K. S. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," School Elec. Eng., Purdue Univ., 1979.

[16] J. R. Ullman, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, pp. 31–42, Jan. 1976.

[17] D. L. Waltz, "Generating semantic descriptions from drawings of scenes with shadows," MIT Tech. Rep. A1271, Nov. 1972.

[18] S. W. Zucker and J. L. Mohammed, "A hierarchical relaxation system for line labeling and grouping," in *Proc. IEEE Conf. Pattern Recog. Image Process.*, 1978, pp. 410–415.

Linda G. Shapiro (S'74–M'74–SM'81) was born in Chicago, IL, in 1949. She received the B.S. degree in mathematics from the University of Illinois at Urbana-Champaign in 1970, and the M.S. and Ph.D. degrees in computer science from the University of Iowa, Iowa City, in 1972 and 1974, respectively.

She was an Assistant Professor of Computer Science at Kansas State University, Manhattan, from 1974 to 1978. She is currently a Professor at Virginia Polytechnic Institute and State University, Blacksburg. Her research interests include scene analysis, pattern recognition, spatial information systems, computer graphics, and data structures. She has completed an undergraduate textbook on data structures with R. Baron.

Dr. Shapiro is a member of the Association for Computing Machinery, the Pattern Recognition Society, and the American Association for Artificial Intelligence. She is currently Co-Editor of the Newsletter for the IEEE Technical Committee on Machine Intelligence and Pattern Analysis.

Robert M. Haralick (S'62–S'67–M'69–SM'76) received the Ph.D. degree from the University of Kansas, Lawrence, KS, in 1969.

He is presently a Professor in the Department of Electrical Engineering and in the Department of Computer Science at Virginia Polytechnic Institute and State University, Blacksburg, VA. He has done research in pattern recognition, multiimage processing, remote sensing, texture analysis, image data compression, clustering, artificial intelligence, and general system theory. Dr. Haralick is an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, *Pattern Recognition*, and *Computer Graphics and Image Processing*. He is on the Editorial Board of the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE and is the Vice-Chairman of IEEE Computer Society Technical Committee on Machine Intelligence and Pattern Analysis.

Dr. Haralick is a member of the Association for Computing Machinery, the Pattern Recognition Society, and the Society for General System Research.