ALGORITHMS FOR INEXACT MATCHING

Linda G. Shapiro

Department of Computer Science
Virginia Polytechnic Institute
and State University
Blacksburg, VA 24061

Robert M. Haralick

Department of Electrical Engineering
Department of Computer Science
Virginia Polytechnic Institute
and State University
Blacksburg, VA 24061

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 analyze the kinds of errors that can occur to make the tree search fail. We develop the formulas that can be used in lookahead operators to rule out a node near the top of the tree because too much error will have accumulated by the time the search reaches the bottom. Using these formulas, we describe several different algorithms to make the tree search more efficient. We present experimental results showing that forward checking is the most efficient of the algorithms tested.

Key Words: structural description, matching, inexact matching, relational homomorphism, tree search, backtracking, forward checking, lookahead, relaxation

This research was funded by the National Science Foundation under grants MCS-7923827 and MCS-7919741

I. Structural Descriptions and Exact Matching

A structural description D of an object is a pair D = (P,R). P = {P1,...,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 $\subset$ A x V where A is a set of possible attributes and V is a set of possible values. R = {PR1,...,PRK} is set of named N-ary relations over P. For each k = 1,...,K, PRk is a pair (NRk,Rk) where NRk is a name for relation Rk, and for some positive integer Mk, Rk $\subset$ P**MK. Thus the set P represents the parts of an object, and the set R represents the interrelationships among the parts.

One way that structural descriptions are used is to define prototype objects. 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 II.

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 $\subset$ P**N be an N-ary relation over a set P and h be a function h:P-->Q mapping elements of P into a set Q. We define the composition R~h of R with h by

$$R\sim h = \{(q1,\ldots,qn) \in Q \mid \text{there exists } (p1,\ldots,pn) \in R \text{ with } h(pi) = qi,\ i=1,\ldots,N\}$$

Let S $\subset$ Q**N be a second N-ary relation. A relational homomorphism from R to S is a mapping h:P-->Q that satisfies R~h $\subset$ 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.

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 rela-

tional homomorphism is equivalent to determining that the two objects are similar.

Let Dp = (P,R) be a prototype structural description and Dc = (Q,S) be a candidate structural description. Let P = {P1,...,Pn}, Q = {Q1,...,Qm}, R = {(NR1,R1),...,(NRk,RK)}, and S = {(NS1,S1),...,(NSk,Sk)}. We say that Dc matches Dp if there is a mapping h : P --> Q satisfying

1) h(Pi) = Qj implies Pi $\subseteq$ Qj, and
2) NRi = NSj implies Ri~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 homomorophism from Ri to Sj.

II. Weighted Prototype Structural Descriptions

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 [6]. 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' | $\leq$ ta.

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,...,Pn} is a set of primitives as before, and wp is a primitive-weighting function, wp:P-->[0,1] that assigns a weight to each primitive in P and satisfies $\sum$ wp(Pi) = 1. R = {(NR1,R1),...,(NRK,RK)} is again a set of named N-ary relations over P. WR = {w1,...,wK} is a set of N-tuple-weighting functions. For each k = 1,..., K, wk assigns weights to the Mk-tuples of relation Rk. Thus each wk is a function wk:Rk-->[0,1] satisfying $\sum_{r \in Rk}$ wk(r) = 1.

III. 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-->[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-->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-->Q such that:

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

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 structual description. Suppose Dp = (P,wp,RP,WRP) where P = {P1,...,Pn}, RP = {(NR1,R1),...,(NRk,Rk)}, and WRP = {w1,...,wk}. Suppose Dc = (C, RC) where C = {C1,...,Cm} and RC = {(NS1,S1),...,(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 = {ϵi | PRi $\in$ RP} if there is a mapping h:P-->C U {null} that satisfies

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

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

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

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, 1979 [2]). 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 $\subset$ U**N be a unit constraint relation. That is, if an N-tuple (ul,...,uN) is an element of T, then the label of one unit ui in the N-tuple is constrained by the labels of the other units in the N-tuple. Let R $\subset$ (U X L)**N be a unit-label constraint relation. That is, if an N-tuple ((ul,ll),...,(uN,lN)) [written as (ul,ll,...,uN,lN)] is an element of R, then unit ul may have label ll, unit u2 may have label l2, . . . , and unit uN may have label lN, all simultaneously. The consistent labeling problem is to find a mapping f : U --> L satisfying that if (ul,...,uN) is in T, then (ul,f(ul),...,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 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 begin the extension to ϵ-consistent labelings.

Lookahead for Inexact Matching

Let (U,L,T,R) be a compatibility model. Let Ew : T X L**N --> [0,1] be a non-negative function. Ew(ul,...,uN, ll,...,lN) is the error that occurs when the N-tuple (ll,...,lN) of labels is applied to units (ul,...,uN). The inexact consistent labeling problem is to find all mappings h : U --> 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 ϵ0. That is, we must find all h satisfying

$$\sum_{(ul,\ldots,uN)\ \in\ T} Ew(ul,\ldots,uN,h(ul),\ldots,h(uN)) \ \leq\ \epsilon 0.$$

Note that when Ew(ul,...,uN,ll,...,lN) is defined to be w(ul,...,uN) when ((ul ,ll),...,(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. The backtracking strategy suffers from thrashing behavior. That is, the search fails at several different places in the tree, all for the same reason. If the reason for failure could be remembered or anticipated, then the tree search could be made more efficient.

To understand this thrashing behavior better, consider why the tree seach could fail without our 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 unis 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

$$\sum_{(ul,\ldots,uN)\ \in\ T \text{ intersect } Up**N} Ew(ul,\ldots,uN,h(ul),\ldots,h(uN)).$$

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

$$\sum_{(ul,\ldots,uN)\ \in\ T \text{ intersect } Uf**N} \ \min_{(ll,\ldots,lN)} Ew(ul,\ldots,uN,ll,\ldots,lN).$$

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

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

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

$$epf(u,l;Up,h) = \sum_{i=1}^{N} \sum_{(ul,\ldots,uN) \in T(u,i;Up)} w(ul,\ldots,u(i-1),u,u(i+1),\ldots,uN,h(ul),\ldots,h(u(i-1)),\ l,h(u(i+1)),\ldots,h(uN))$$

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 epf(u,l;Up,h). The smallest error that the future units individually incur given the partial labeling h is

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

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.

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

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

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

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

is the error that the current labeling h on Up causes on the future unit-label pairs (u,l) and (v,m).

V. Tree Searching Algorithms

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

$$ep(Up,h) = \sum_{(ul,\ldots,uN) \in T \text{ intersect } Up^{**}N} Ew(ul,\ldots,uN,h(ul),\ldots,h(uN)).$$

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

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 the minimum error the past units must have with the future units. Recall that ef(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. Forward checking uses

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

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

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 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

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

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

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

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 relation operator originally defined by Ullman [7], independently rediscovered by Waltz [8], and also discussed in Rosenfeld, Hummel, and Zucker [4], Haralick and Shapiro [2], and Gaschnig [1].

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

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

Hence looking ahead by one uses the quantity

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

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.

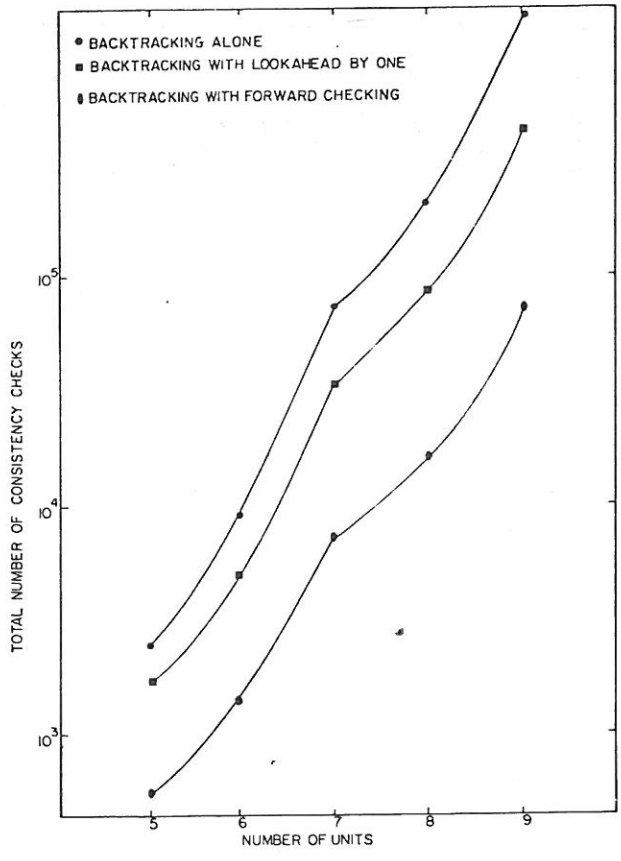
VI. Results

In order to 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. For a description of this model, see Haralick and Elliott [3] who explored the behavior of various algorithms for finding exact or zero-consistent labelings.

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. A back check is a consistency check performed in the context of straight backtracking. A lookahead is a consistency check performed in the context of forward checking or lookahead by one. A lookup is a table lookup performed in the context of forward checking or lookahead by one. 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 lookaheads, 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.

In comparing backtracking alone, backtracking plus forward checking, and backtracking plus lookahead 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 lookahead by one was next, and backtracking alone had the highest number of consistency checks and the most time. Shown below is the total number of consistency checks as a function of number of units for the three different search algorithms with p = .5 and ϵ = .1.



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 lookahead by one searched the fewest nodes. 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 case for exact matching (Haralick and Elliott [3]), in inexact matching, forward checking is the most efficient of the three methods of search.

Due to lack of space, we have omitted the rest of the results from this paper. For more results, see Shapiro and Haralick [5].

REFERENCES

1. Gaschnig, J., "A General Backtrack Algorithm that Eliminates Most Redundant Tests", Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1972, p. 457.

2. Haralick, R.M. and L.G. Shapiro, "The Consistent Labeling Problem: Part 1", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, April 1979, pp. 173-184.

3. Haralick, R.M. and G.L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Proceedings of the 6th International Joint Conference on Artificial Intelligence, 1979.

4. Rosenfeld, A., R.A. Hummel, and S.W. Zucker, "Scene Labeling by Relaxation Operations", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, June 1976, pp. 420-433.

5. Shapiro, L.G., and R.M. Haralick, "Structural Descriptions and Inexact Matching", to appear in IEEE Transactions on Pattern Analysis and Machine Intelligence, 1981.

6. Tsai, W.H. and K.S. Fu, Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis, School of Electrical Engineering, Purdue University, 1979.

7. Ullman, J.R., "An Algorithm for Subgraph Homomorphisms", Journal of the ACM, Vol. 23, Jan. 1976, pp.31-42.

8. Waltz, D.L., Generating Semantic Descriptions from Drawings of Scenes with Shadows, MIT Tech. Rep. A1271, Nov. 1972.