Relational matching

Linda G. Shapiro and Robert M. Haralick

The problem of computer vision is to give a computer a representation of a picture of a scene and have it figure out what objects are in the scene and in what spatial relationships. This involves low-level vision (image processing), midlevel vision (feature extraction and measurement), and high-level vision (interpretation). One important part of high-level vision is relational matching, the process of matching two relational descriptions of objects, often for the purpose of object identification. In this paper, we describe several kinds of relational matching, give sequential algorithms for solving relational matching problems, and briefly discuss parallel algorithm possibilities.

I. Introduction

Computer vision is a multidisciplinary research area populated by engineers, computer scientists, mathematicians, psychologists, physicists, and others. The general problem of computer vision is to give a computer a representation of a picture of a scene and have it figure out what objects are in the scene and in what spatial relationships they lie. For humans this is a trivial problem; small children can do it. For computers, the problem is far from solved.

The representation of a picture of a scene given as data to a computer is called an image. An image consists of one or more matrices of numbers. Each matrix is called a band, and each number is associated with one pixel (picture element). A gray tone image consists of one band in which each pixel value represents the gray tone intensity of a portion of the picture. A color image consists of three bands for red, green, and blue intensity information. Gray tone and color images often come from TV cameras. Other important classes of image include radar images, FLIR images, and range images.

Traditionally, computer vision can be partitioned into three levels: low-level vision; midlevel vision; and high-level vision. A low-level vision process will input one or more images, perform some image processing operation, and output one or more images. The most common kind of operator in low-level vision is called a neighborhood operator. Such an operator examines the values of small neighborhood of pixels around a given pixel and produces a resultant value that is a function of all the pixel values in the neighborhood. The resultant value becomes the new value of the given pixel in an output image. For example, the simplest edge operators are neighborhood operators whose final result is a binary image which has a value of 1 at each pixel where the edge operator thinks an edge is present in the original image and value 0 elsewhere. Another low-level operator is one that segments an image into regions of interest. Such a process could use the binary edge image produced by an edge operator as additional input. The output of a segmentation operator is a symbolic image, in which each pixel of a region contains the unique numeric label of that region.

A midlevel vision process will input one or more images, perform some analysis, and output data structures that describe the properties of, or relationships between, features extracted from the images. Midlevel processes may in addition produce more output images and may input data structures from other midlevel processes. Given an original image and an edge image obtained from applying a low-level edge operator, one useful midlevel process might attempt to link the edge pixels into features called arcs. On the image, each arc is a connected chain of pixels. The data structure produced is called an arc property list. It is a table accessible by arc number and containing the values of the properties of each arc. Typical arc properties include coordinates of the head and tail, direction of the best linear fit, length, and average contrast across the arc. A second data structure produced contains the chain encodings of the arcs.

Given an original image and a symbolic image obtained from applying a segmentation operator, another midlevel process determines a set of properties of each region. It produces a data structure called a region property list which is a table accessible by region number and containing the values of the properties of each region. Typical region properties include mean, maximum, and minimum gray tones, gray tone variance, coordinates of the centroid, elongation, circularity, and moments. Here a second data structure could contain the chain encodings of the boundaries of the regions.

The authors are with University of Washington, Department of Electrical Engineering, Seattle, Washington 98195.

Received 11 June 1986.
0003-6935/87/101845-07$02.00/0.

A third midlevel operator inputs an image and the symbolic image representing its segmentation and produces a data structure called a region adjacency graph. This structure is accessible by region label and contains for each region a list of all adjacent region labels. It is useful in further analysis or for producing coarser and coarser segmentations. Other structures produced by midlevel operators contain relationships among arcs and relationships between arcs and regions.

High-level vision processes input the data structures produced by midlevel vision processes and output an interpretation of the scene. High-level vision processes perform matching and reasoning tasks and coordinate the use of low-level and midlevel processes. One important task that high-level vision performs is to identify objects in the scene from their projections on the image and to interpret the meaning of the scene as a whole. Although classification of many simple objects can be performed by statistical techniques, the recognition of complex objects having parts in various spatial relationships requires a different approach. When the objective is not only to recognize an object but also to measure some critical angles or distances on the object, again statistical techniques are not sufficient. When the scene is interpreted as a whole, the analysis depends on the interpretations of the various objects in the scene and on their spatial relationships. In all these tasks, an approach called relational matching is used to solve the problem.

In this paper, we define several kinds of relational matching and give sequential algorithms for solving relational matching problems. The relational matching problem is a computationally difficult one that clearly could benefit from a parallel approach. The purpose of this paper is to state the problem and the sequential procedures for its solution in the hope that researchers in parallel optic computation may be able to help discover a practical parallel algorithm.

II. Relational Descriptions and Mappings

How can a complex object or entity be described? The object or entity has global properties such as area, height, and width. It also has a set of parts or important features. The parts each have properties of their own, and there are spatial relationships that describe their interconnections. To define the process of relational matching, we need a unified context in which to express these properties and relationships. We call this context a relational description. A relational description is a set of relations that together describe a complex object or entity. The relation is the basic unit of a relational description. Thus we will start with relations.

A. Relations

Let O_A be an object or entity and A be the set of its parts or important features. An N-ary relation R over A is a subset of the Cartesian product $A^N = A \times \ldots \times A$ (N times). For example, suppose that O_A is a chair and its part set A consists of four legs, a back, and a seat. A list of the parts is a unary relation $R_1 \subseteq A$. A list of the pairs of parts that connect together is a binary relation $R_2 \subseteq A \times A$. Other binary relations of interest include the list $R_3 \subseteq A \times A$ of pairs of parallel parts and the list $R_4 \subseteq A \times A$ of pairs of perpendicular parts. The set of triples of the form (p_1,p_2,p_3) where parts p_1 and p_3 both connect to part p_2 is a fifth relation $R_5 \subseteq A \times A \times A$. The set $D_A = \{R_1,R_2,R_3,R_4,R_5\}$ forms a relational description of the chair. This relational description describes only spatial relationships. Before we add properties to make the descriptions more robust, we discuss a method for comparing these simple relations.

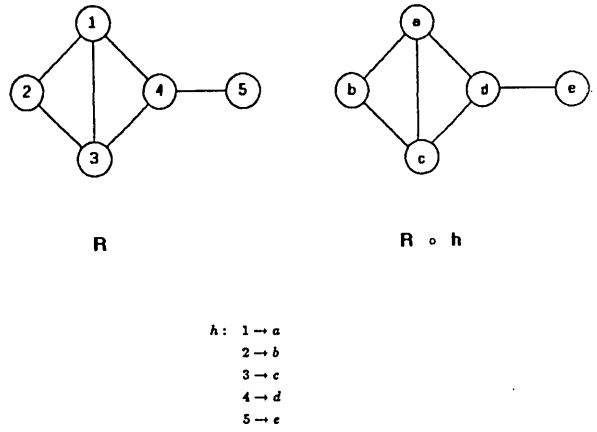


Fig. 1. Composition of binary relation R with mapping h.

B. Relational Homomorphisms

Let A be the part set of object O_A and B be the part set of object O_B. Let $R \subseteq A^N$ be an N-ary relation over part set A. Let $f: A \rightarrow B$ be a function that maps elements of set A into set B. We define the composition $R \circ f$ of R with f by

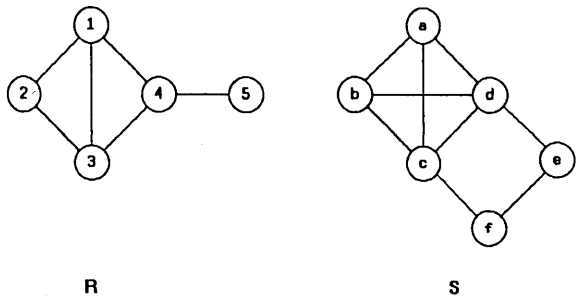
$$R \circ f = \{(b_1,\ldots,b_N) \in B \mid \text{there exists}$$

$$(a_1,\ldots,a_N) \in R \text{ with } f(a_i) = b_i, i = 1,\ldots,N)\}.$$

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

Let $S \subseteq B^N$ be a second N-ary relation. A relational homomorphism from R to S is a mapping $f: A \rightarrow B$ that satisfies $R \circ f \subseteq S$. 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. Figure 2 illustrates the concept of a relational homomorphism.

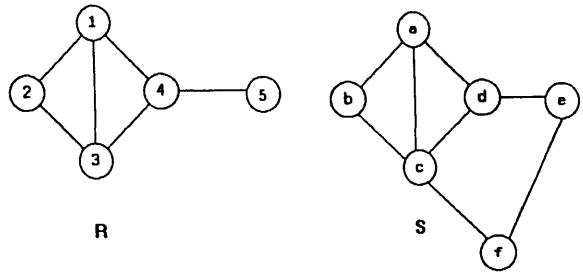
A relational homomorphism maps the primitives of A to a subset of the primitives of R having all the same interrelationships that the original primitives of A had. If A is a much smaller set than B, finding a one–one relational homomorphism is equivalent to finding a copy of a small object as part of a larger object. Finding a chair in an office scene is an example of such a task. If A and B are about the same size, 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 A to



h: $1 \rightarrow a$
$2 \rightarrow b$
$3 \rightarrow c$
$4 \rightarrow d$
$5 \rightarrow e$

$R \circ h \subseteq S$

Fig. 2. Relational homomorphism h from binary relation R to binary relation S.



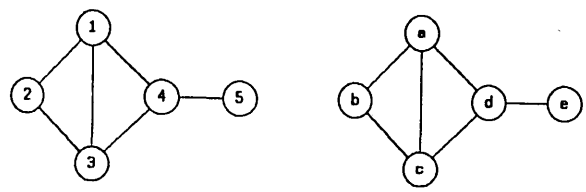
h: $1 \rightarrow a$
$2 \rightarrow b$
$3 \rightarrow c$
$4 \rightarrow d$
$5 \rightarrow e$

$R \circ h \subseteq S$; h is one – one

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

a unique primitive in B. A monomorphism indicates a stronger match than a homomorphism. Figure 3 illustrates a relational monomorphism.

Finally, a relational isomorphism f from an N-ary relation R to an N-ary relation S is a one–one, onto relational homomorphism from R to S, and f^{-1} is a relational homomorphism from S to R. In this case, A and B have the same number of elements, each primitive in A maps to a unique primitive in B, and every primitive in A is mapped to by some primitive of B. 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. Figure 4 illustrates a relational isomorphism, and Fig. 5 shows the difference between a relational isomorphism and a relational monomorphism.



h: $1 \rightarrow a$
$2 \rightarrow b$
$3 \rightarrow c$
$4 \rightarrow d$
$5 \rightarrow e$

$R \circ h = S$ and h is $1-1$

or equivalently,

$R \circ h \subseteq S$, $S \circ h^{-1} \subseteq R$, and h is $1-1$

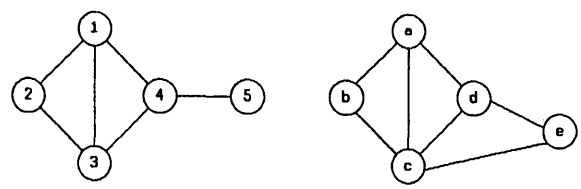
Fig. 4. Relational isomorphism h from binary relation R to binary relation S.

C. Relational Descriptions and Relational Distance

A relational description D_x is a sequence of relations $D_x = \{R_1, \ldots, R_I\}$ where for each $i = 1, \ldots, I$, there exists a positive integer n_i with $R_i \subseteq X^{n_i}$ for some set X. X is a set of the parts of the entity being described, and the relations R_i indicate various relationships among the parts. A relational description may be used to describe an object model, a group of regions on an image, a 2-D shape, a Chinese character, or anything else having structure to it. In the spirit of the relational homomorphism defined in the previous section, we wish to define a distance measure for pairs of relational descriptions.

Let $D_A = \{R_1, \ldots, R_I\}$ be a relational description with part set A. Let $D_B = \{S_1, \ldots, S_I\}$ be a second relational description with part set B. We assume that $|A| = |B|$; if this is not the case, we add enough dummy parts to the smaller set to make it the case.

Let f be any one–one onto mapping from A to B. The structural error of f for the ith pair of corresponding relations (R_i and S_i) in D_A and D_B is given by



h: $1 \rightarrow a$
$2 \rightarrow b$
$3 \rightarrow c$
$4 \rightarrow d$
$5 \rightarrow e$

$R \circ h \subseteq S$, h is $1-1$, and h is onto

Fig. 5. 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.

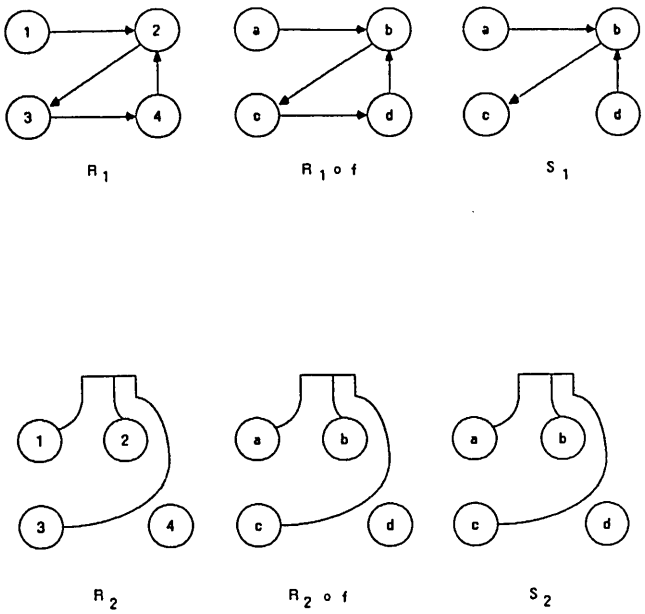


Fig. 6. Relations R_1,$R_1 \circ f$,S_1,$R_2R_2 \circ f$, and S_2. The notation indicates a hyperarc representing a triple.

$$E_s^i(f) = |R_i \circ f - S_i| + |S_i \circ f^{-1} - R_i|.$$

The structural error indicates how many tuples in R_i are not mapped by f to tuples in S_i and how many tuples in S_i are not mapped by f^{-1} to tuples in R_i.

The total error of f with respect to D_A and D_B is the sum of the structural errors for each pair of corresponding relations. That is,

$$E(f) = \sum_{i=1}^{I} E_s^i(f).$$

The total error gives a quantitative idea of the difference between the two relational descriptions D_A and D_B with respect to the mapping f.

The relational distance between D_A and D_B is then given by

$$GD(D_A,D_B) = \min E(f).$$

$$f: A \xrightarrow[\text{onto}]{1\text{-}1} B.$$

This is, the relational distance is the minimal total error obtained for any one-one onto mapping f from A to B. In Ref. 1 we proved that the relational distance is a metric over the space of relational descriptions. We call a mapping f that minimizes total error a best mapping from D_A to D_B. If there is more than one best mapping, we arbitrarily select one as the designated best mapping. More than one best mapping occurs when the relational descriptions involve certain kinds of symmetry.

1. *Examples*

Let $A = \{1,2,3,4\}$ and $B = \{a,b,c,d\}$. Let $D_A = \{R_1 \subseteq A^2, R_2 \subseteq A^3\}$, and $D_B = \{S_1 \subseteq B^2, S_2 \subseteq B^3\}$. Let $R_1 = \{(1,2)(2,3)(3,4)(4,2)\}$ and $S_1 = \{(a,b)(b,c)(d,b)\}$. Let $R_2 = \{(1,2,3)\}$ and $S_2 = \{(a,b,c)\}$. Let f be defined by $f(1) = a, f(2) = b, f(3) = c, f(4) = d$. These relations are illustrated in Fig. 6. Then we have

$$\begin{aligned}
|R_1 \circ f - S_1| &= |\{(a,b),(b,c),(c,d),(d,b)\} \\
&\quad -\{(a,b),(b,c),(d,b)\}| = 1, \\
|S_1 \circ f^{-1} - R_1| &= |\{(1,2),(2,3),(4,2)\} \\
&\quad -\{(1,2),(2,3),(3,4),(4,2)\}| = 0, \\
E_s^1(f) &= 1 + 0 = 1, \\
|R_2 \circ f - S_2| &= |\{(a,b,c)\} - \{(a,b,c)\}| = 0, \\
|S_2 \circ f^{-1} - R_2| &= |\{(1,2,3)\} - \{(1,2,3)\}| = 0, \\
E_s^2(f) &= 0 + 0 = 0, \\
E(f) &= E_s^1(f) + E_s^2(f) = 1.
\end{aligned}$$

We note that f is the best mapping and, therefore, $GD(D_A,D_B) = 1$.

For a simple but practical example consider a set of object models constructed from simple parts with two binary relations: the connection relation and parallel relation. Figure 7 illustrates a model ($M1$) and two other models ($M2$ and $M3$) that are each a relational distance of 1 from the first model. The model $M4$ shown in Fig. 8 is a variation of $M3$, but its relational distance from $M3$ is 6 due to several missing relationships induced by the additional two parts.

D. Attributed Relational Descriptions and Relational Distance

The relational descriptions defined in the previous section describe relationships among parts but not properties of parts, properties of the whole, or properties of these relationships. However, it is easy to extend both the concept of relational description and the definition of relational distance to include them. Intuitively, an m-tuple of attributes added to an n-tuple of parts produces an $n + m$-tuple that specifies a relationship plus the properties of that relationship. If $n = 1$ and $m > 0$, each tuple lists a part and its properties. If $n = 0$, $m > 0$, and the relation has only one tuple, this is a property vector describing the global properties of the object. Formally the definitions change to the following.

Let X be a set of parts of object O_X and P be a set of property values. Generally, we can assume P is the set of real numbers. An attributed relation over part set X with property value set P is a subset of $X^n \times P^m$ for some non-negative integers n and m. An attributed relational description D_X is a sequence of attributed relations $D_X = \{R_1,\ldots,R_I\}$, where for each $i = 1,\ldots,I$ there exists a non-negative integer n_i, a non-negative integer m_i (where $n_i + m_i > 0$), and a property value set P_i with $R_i \subseteq X^{n_i} \times P_i^{m_i}$. For example, a binary parts connection relation $R \subseteq X^2$ can be extended to an attributed relation $R' \subseteq X^2 \times \mathcal{R}$; $\mathcal{R}$ is the set of real numbers, and an attributed pair (x_1,x_2,a) specifies that part x_1 connects to part x_2 at angle a.

Consider an attributed relation $R \subseteq A^n \times P^m$ over some part set A and property value set P. Let $r \in R$ be an $n + m$-tuple having n parts followed by m property values. Let $S \subseteq B^n \times P^m$ be a second attributed

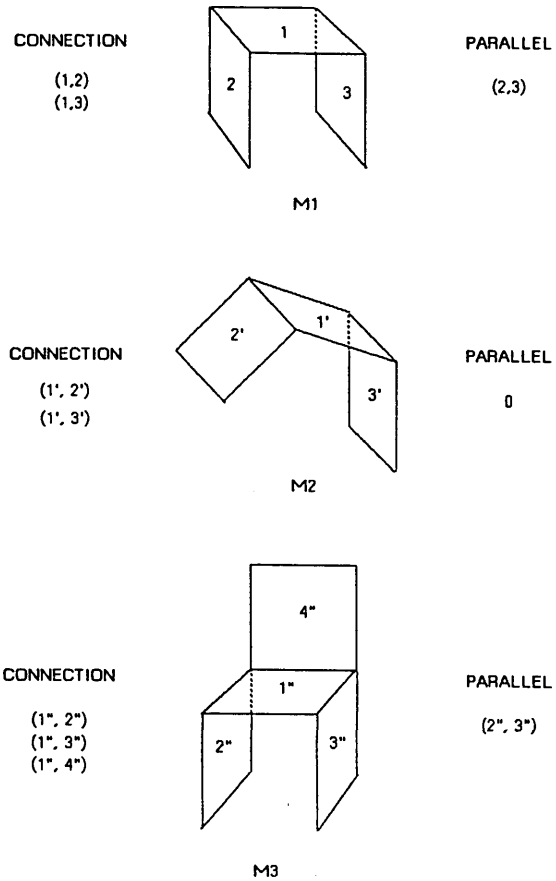


Fig. 7. Object model $M1$ and two other models, $M2$ and $M3$, that are each a relational distance of 1 from $M1$.

relation over part set B and property value set P. Let $f: A \rightarrow B$ be a one–one onto mapping from A to B. We define the composition $r \circ f$ of attributed tuple r with f by

$$r \circ f = \{(b_1,\ldots,b_n,p_1,\ldots,p_m) \in B^n \times P^m \mid \text{there exists } (a_1,\ldots,a_n,p_1,\ldots,p_m) \in R \text{ with } f(a_i) = b_i, i = 1,\ldots,n\}.$$

Assume that if $(b_1,\ldots,b_n,p_1,\ldots,p_m) \in S$ and $(b_1,\ldots,b_n,q_1,\ldots,q_m) \in S$, then $p_1 = q_1,\ldots,p_m = q_m$. That is, each n-tuple of parts has only one m-tuple of properties. The error of a tuple $t = (b_1,\ldots,b_n,p_1,\ldots,p_m)$ with respect to a relation $S \subseteq B^n \times P^m$ is given by

$$e(t,S) = \begin{cases} \text{norm-dis}((p_1,\ldots,p_m),(q_1,\ldots,q_m)) & \text{if } (\exists (q_1,\ldots,q_m) \in P^m, s.t. (b_1,\ldots,b_n,q_1,\ldots,q_m) \in S \\ 1 & \text{otherwise,} \end{cases}$$

where norm-dis returns the Euclidean distance (or any other desired distance) between two vectors, normalized by dividing by some maximum possible distance. Thus $e(t,s)$ is a quantity between 0 and 1. Now we can extend the definition of the structural error of f for the ith pair of corresponding relations (R_i and S_i) to

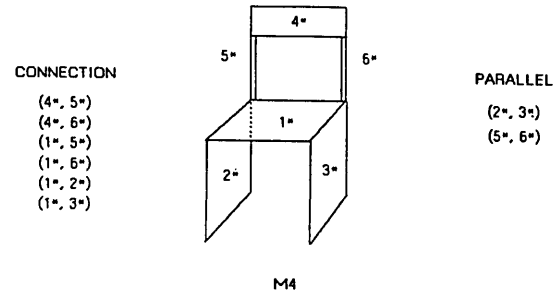


Fig. 8. Model $M4$ that differs from $M3$ by a relational distance of 6.

$$E_s^i(f) = \sum_{r \in R_i} e(r \circ f, S_i) + \sum_{s \in S_i} e(s \circ f^{-1}, R_i).$$

Total error and relational distance are defined as in Sec. II.C.

III. Algorithms for Relational Matching

In Sec. II we explored several ways of defining relational matching. One can demand that two relational descriptions be isomorphic in order to say they match or one can be more lenient and say there must be a relational homomorphism from the first to the second. Furthermore, it may be desirable to find the best match between an unknown relational description and a set of stored relational models. In this case, the stored model that has the least relational distance to the unknown description is the best match. Whether the object is to detect relational isomorphisms, monomorphisms, or homomorphisms or to compute relational distance, the only known algorithms that can solve arbitrary matching problems employ a tree search. In this section we describe the standard backtracking tree search and one of its variants, and we make some comments on parallel algorithms. For more details and other variants, see Refs. 2–7. To simplify the discussion, the algorithms presented will be to determine all relational homomorphisms from a relation R to a relation S. The algorithms for monomorphisms, isomorphisms, and relational distance are straightforward variations of the homomorphism algorithms.

A. Backtracking Tree Search

Let R be an N-ary relation over part set A and S be an N-ary relation over part set B. We will refer to the elements of set A as the units and the elements of set B as the labels. We wish to find the set of all mappings $f: A \rightarrow B$ that satisfy $R \circ f \subseteq S$. Of course, the set may be empty, in which case the algorithm should fail. The backtracking tree search begins with the first unit of A. This unit can potentially match each label in set B. Each of these potential assignments is a node at level 1

of the tree. The algorithm selects one of these nodes, makes the assignment, selects the second unit of A, and begins to construct the children of the first node, which are nodes that map the second unit of A to each possible label of B. At this level, some of the nodes may be ruled out because they violate the constraint $R \circ f \subseteq S$. The process continues to level $|A|$ of the tree. The paths from the root node to any successful nodes at level $|A|$ are the relational homomorphisms. Figure 9 illustrates a portion of the backtracking tree search for a simple digraph matching problem. The algorithm for a backtracking tree search is as follows:

```
procedure tree search(A,B,f,R,S)
a := first(A);
for each b ∈ B
 {
 f' := f ∪ {(a,b)};
 OK := true;
 for each N-tuple r in R containing component a
  and whose other components are all in domain(f)
   if r ∘ f' is not in S
   then {OK := false; break} endif;
 A' = remainder(A);
 if isempty(A')
 then output(f')
 else tree search(A',B,f',R,S);
 }
end tree search;
```

B. Backtracking with Forward Checking

The backtracking tree search has exponential time complexity. Although there are no known polynomial algorithms in the general case, there are a number of discrete relaxation algorithms that can cut down search time by reducing the size of the tree that is searched. Forward checking is one such method. It is based on the idea that once a unit-label pair (a,b) is instantiated at a node in the tree, the constraints imposed by the relations cause instantiation of some future unit-label pairs (a',b') to become impossible. Suppose that (a,b) is instantiated high in the tree and that the subtree beneath that node contains nodes with first components $a_1,a_2,\ldots,a_n,a'$. Although (a',b') is impossible for any instantiations of $(a_1,a_2,\ldots,a_n)$, it will be tried in every path that reaches its level in the tree. The principle of forward checking is to rule out (a',b') at the time that (a,b) is instantiated and keep a record of that information.

The data structure used to store the information is called a future error table (FTAB). There is one future error table for each level of recursion in the tree search. Each table is a matrix having one row for each element of A and one column for each element of B. For any uninstantiated or future unit $a' \in A$ and potential label $b' \in B$, FTAB$(a',b') = 1$ if it is still possible to instantiate (a',b') given the history of instantiations already made. FTAB$(a',b') = 0$ if (a',b') has already been ruled out due to some previous assignment. When a pair (a,b) is instantiated by the backtracking tree search, an updating procedure is called to examine all pairs (a',b') of future units and their possible labels. For each pair (a',b') that is incompatible with the assignment of (a,b) and the previous instantiations, FTAB(a',b') has become 0. If for any future unit a',FTAB(a',b') becomes 0 for all labels $b' \in B$, instantiation of (a,b) fails immediately. The backtracking tree search with forward checking is as follows:

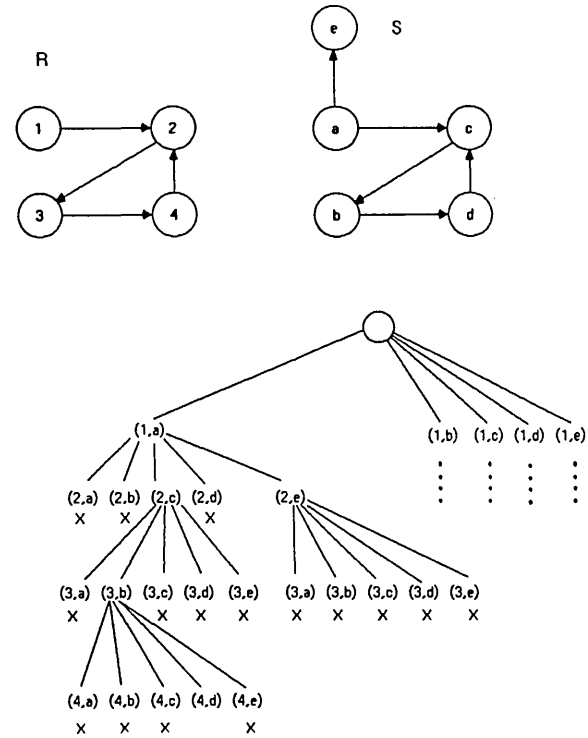


Fig. 9. Backtracking tree search to find a homomorphism from $R = \{(1,2),(2,3),(3,4),(4,2)\}$ to $S = \{(a,c),(c,b),(b,d),(d,c),(a,e)\}$. × under a node indicates failure. The only homomorphism found is with $f = \{(1,a),(2,c),(3,b),(4,d)\}$.

```
procedure forward-checking-tree search (a,b,f,FTAB,R,S)
a := first (A);
for each b ∈ B
 if (FTAB(a,b) == 1)
 then
  {
  f' := f ∪ {(a,b)};
  A' = remainder (A);
  if isempty (A')
  then output (f')
  else
   {
   NEWFTAB := copy(FTAB);
   OK := update(NEWFTAB, a,b,A',B,R,S,f');
   if (OK) forward-checking-tree search
   (A',B,f',NEWFTAB,R,S);
   }
  endif
 endif
procedure update(FTAB,a,b,future-units,B,R,S,f')
update := false;
for each a' ∈ future-units
 for each b' ∈ B with FTAB(a',b') == 1
  if compatible (a,b,a',b',R,S,f')
  then update := true
  else FTAB(a',b') := 0
  endif;
end update.
```

For binary relations R and S, the utility function compatible, which determines whether an instantiation of (a',b') is possible given instantiation (a,b), is very simple. Units a and a' only constrain one another when either (a,a') or (a',a) is in R. Thus the algorithm for function compatible for binary relations R and S is as follows:

```
procedure b-compatible(a,b,a',b',R,S,f')
if ((a,a') ∈ R and not ((b,b') ∈ S)) or
 ((a',a) ∈ R and not ((b',b) ∈ S))
 then b-compatible := false
 else b-compatible := true endif;
end b-compatible;
```

Note that for binary functions, the last argument f' to function b-compatible is not used but is included here for consistency.

For N-ary relations R and S, $N > 2$, those N-tuples of R where a and a' are among the components and all other components are already instantiated must be examined. The code for N-ary relations R and S is as follows:

```
procedure compatible(a,b,a',b',R,S,f')
f'' := f' ∪ {(a',b')};
compatible := true;
for each r ∈ R containing a and a' whose other components
 are in domain(f'')
  if r ○ f'' is not in S
  then {compatible := false; break} endif;
end compatible;
```

The binary procedure is very fast, since its time complexity is constant. The general procedure, if implemented as stated here, would have to examine each N-tuple of R. For a software implementation, it would be desirable to design the data structures for R, S, and f so that only the appropriate N-tuples of R are tested. A hardware implementation could offer even more flexibility.

C. Parallel Algorithms

To make the relational matching algorithm parallel, one needs to be able to make the backtracking tree search parallel. It is not difficult to understand how to parallelize the tree search in a computational network of parallel processors.[8] The whole tree is given to one processor within the network, and this processor begins to work on the tree search. All processors in the network which are not working on the tree search and are, therefore, idle interrupt in turn all processors to which they can directly communicate. The interrupt essentially is a message indicating idleness. Any processor which is working and receives an idleness interrupt takes the tree it is working on and splits the tree into two subtrees. It keeps one subtree, and it gives the other to the interrupting processor to work on.

As long as there is some communication path, however indirect, between every pair of processors in the network, the above approach to parallelizing the tree search guarantees that every processor gets some work to do after it becomes idle. The communication overhead of passing a subtree to another processor can be minimal if the basic data for the entire problem are broadcast to each of the processors before tree search computation begins. Thus specifying a subtree consists merely of specifying a simple list of all the instantiations already made above the subtree. For example, if the root of the subtree is at a level N from the root node of the entire tree, a list of N ordered pairs of the already instantiated units and their corresponding labels specify the subtree.

In so far as parallelizing the forward checking procedure, a parallel array processor SIMD implementation can be readily formulated.[9] The unit-label table can be represented as a bit matrix with the labels indexing the columns and the units indexing the rows. The updating procedure essentially amounts to ORing over each row and then ANDing those results. This kind of updating must be done for each unit-label pair.

IV. Summary

We have introduced the concepts of relational descriptions, relational homomorphisms and isomorphisms, and relational distance. We have generalized these concepts to attributed relational descriptions and attributed relational distance. We have given procedures for finding relational homomorphisms that operate on sequential computers and briefly discussed parallel algorithms for multiprocessor systems. It is our hope that this presentation encourages readers to work on parallel solutions to the relational matching problem using an optical approach.

References

1. L. G. Shapiro and R. M. Haralick, "A Metric for Comparing Relational Descriptions," IEEE Trans. Pattern Anal. Machine Intell. **PAMI-7,** 90 (1985).
2. R. M. Haralick and L. G. Shapiro, "The Consistent Labeling Problem—Part I," IEEE Trans. Pattern Anal. Machine Intell. **PAMI-1,** 173 (1979).
3. R. M. Haralick and L. G. Shapiro, "The Consistent Labeling Problem—Part II," IEEE Trans. Pattern Anal. Machine Intell. **PAMI-2,** 193 (1980).
4. A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene Labeling by Relaxation Operations," IEEE Trans. Syst. Man Cybern. **SMC-00** 420 (June 1976).
5. L. G. Shapiro and R. M. Haralick, "Structural Descriptions and Inexact Matching," IEEE Trans. Pattern Anal. Machine Intell. **PAMI-3,** 504 (1981).
6. L. G. Shapiro and R. M. Haralick, "Organization of Relational Models for Scene Analysis," IEEE Trans. Pattern Anal. Machine Intell. **PAMI-4,** 595 (1982).
7. L. G. Shapiro, "The Use of Numeric Relational Distance and Symbolic Differences for Organizing Models and for Matching," in *Techniques for 3D Machine Perception* (North-Holland, Amsterdam, 1985).
8. J. T. McCall, J. Tront, F. Gray, R. M. Haralick, and W. M. McCormick, "The Effects of Combinatorial Problem Parameters on the Design of Multi Parallel Architecture," IEEE Trans. Comput. **C-34,** (1985).
9. J. R. Ullman, R. M. Haralick, and L. G. Shapiro, "Computer Architectures for Solving Consistent Labeling Problems," The Comput. J. **28,** 105 (1985).