MATCHING RELATIONAL STRUCTURES USING DISCRETE RELAXATION

LINDA G. SHAPIRO
and
ROBERT M. HARALICK
Department of Electrical Engineering
University of Washington
Seattle, WA 98195
USA

A relational description of an object describes the object in terms of its properties, its parts, and the interrelations among its parts. Relational matching is the process of comparing two relational descriptions to determine the correspondence between their part sets and to decide, based on this correspondence, how similar they are. Relational matching can be used for stereo vision, for object recognition, and for organizing a database of models. In this chapter, we describe several different kinds of relational matching and give algorithms that use discrete relaxation to solve the relational matching problem. An approach to relational matching using parallel hardware is briefly discussed.

1. INTRODUCTION

High-level vision processes perform matching and reasoning tasks. 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, then 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 of these tasks, an approach called *relational matching* can be used to solve the problem. In this paper we define several kinds of relational matching, give sequential algorithms for solving relational matching problems using discrete relaxation, and briefly discuss a parallel approach to relational matching.

2. 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. In order 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, so we will start with relations.

2.1. 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.

2.2. 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 $f : A \to B$ be a function that maps elements of set A into set B. We define the *composition* $R \circ f$ of R with f by

$$\begin{aligned} R \circ f = \{(b_1, \dots, b_N) \in B \mid & \text{ there exists} \\ & (a_1, \dots, a_N) \in R \text{ with } f(a_i) = b_i, i = 1, \dots, N)\} . \end{aligned}$$

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

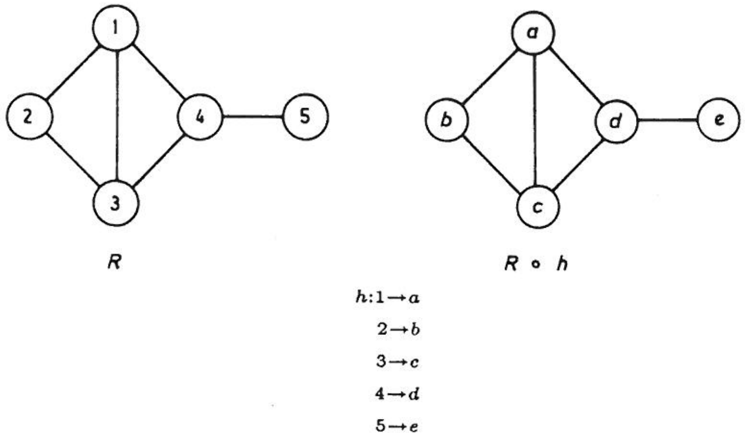
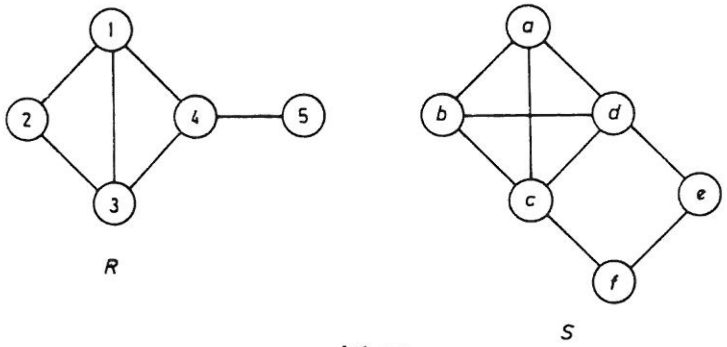


Fig. 1. The composition of binary relation R with mapping h.

Let $S \subseteq B^N$ be a second N-ary relation. A *relational homomorphism* from R to S is a mapping $f : A \to B$ 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 inter-relationships that the original

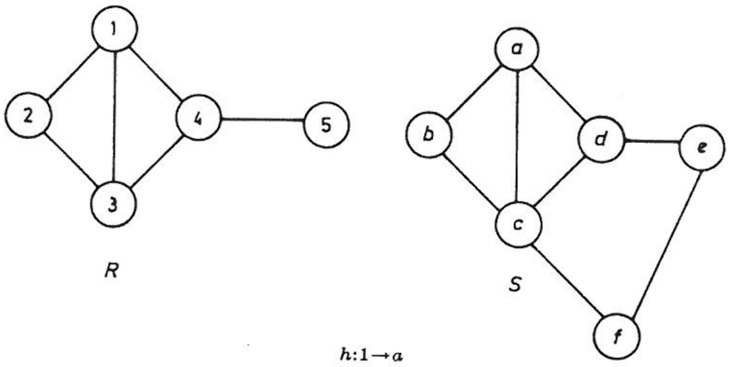


$$\begin{array}{r}
h{:}1 \rightarrow a \\
2 \rightarrow b \\
3 \rightarrow c \\
4 \rightarrow d \\
5 \rightarrow e \\
R \circ h \subseteq S
\end{array}$$

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

primitives of A had. If A is a much smaller set than B, 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 A and B 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 A to a unique primitive in B. A monomorphism indicates a stronger match than a homomorphism. Figure 3 illustrates a relational monomorphism.

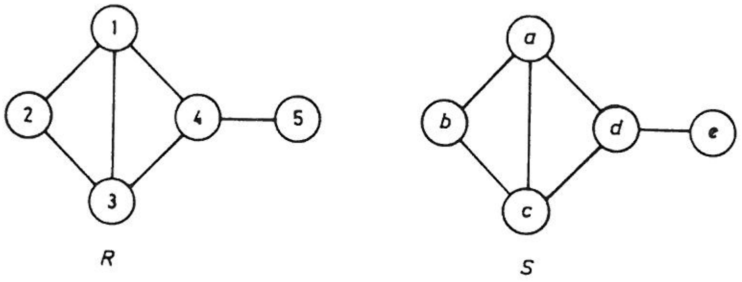
Finally, a *relational isomorphism* f from an N-ary relation R to an N-ary relation S is a one-one relational homomorphism from R to S, and f^{-1} is a relational homomorphism from S to R. In this case, A and B have the same number of elements, each primitive in A maps to a unique primitive in B, and 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.



$h{:}1 \to a$
$2 \to b$
$3 \to c$
$4 \to d$
$5 \to e$

$R \circ h \subseteq S$; h is 1-1

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



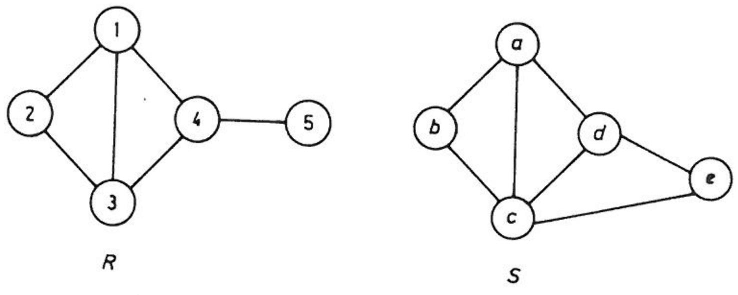
$h{:}1 \to a$
$2 \to b$
$3 \to c$
$4 \to d$
$5 \to e$

$R \circ h = S$ 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. A relational isomorphism 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 1-1, and h is onto.

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.

2.3. Relational Descriptions and Relational Distance

A *relational description* D_X is a set 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 two-dimensional shape, a Chinese character, or anything else having structure to it. In the spirit of the relational homomorphisms 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 will assume that $|A| = |B|$; if this is not the case, we will 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 i th pair of corresponding relations (R_i and S_i) in D_A and D_B is given by

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

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

$$GD(D_A, D_B) = \min_{f : \xrightarrow[\text{onto}]{1-1} B} E(f) \ .$$

That 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 will occur when the relational descriptions involve certain kinds of symmetries.

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

$$\begin{aligned} |R_1 \circ f - S_1| = |\{(a,b),(b,c),(c,d),(d,b)\} \\ - \{(a,b),(b,c),(d,b)\}| = 1 \end{aligned}$$

$$\begin{aligned} |S_1 \circ f^{-1} - R_1| = |\{(1,2),(2,3),(4,2)\} \\ - \{(1,2),(2,3),(3,4),(4,2)\}| = 0 \ , \end{aligned}$$

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

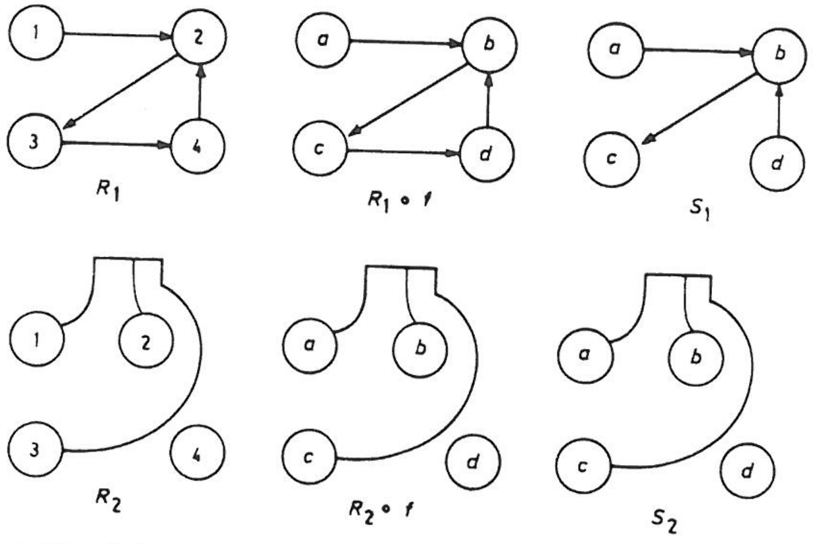


Fig. 6. The relations R_1, $R_1 \circ f$, S_1, R_2, $R_2 \circ f$ and S_2. The notation $\sqcap\!\sqcap$ indicates a hyperarc representing a triple.

$$E_s^2(f) = 0 + 0 = 0 \,,$$
$$E(f) = E_s^1(f) + E_s^2(f) = 1 \,.$$

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 the 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.

2.4. 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

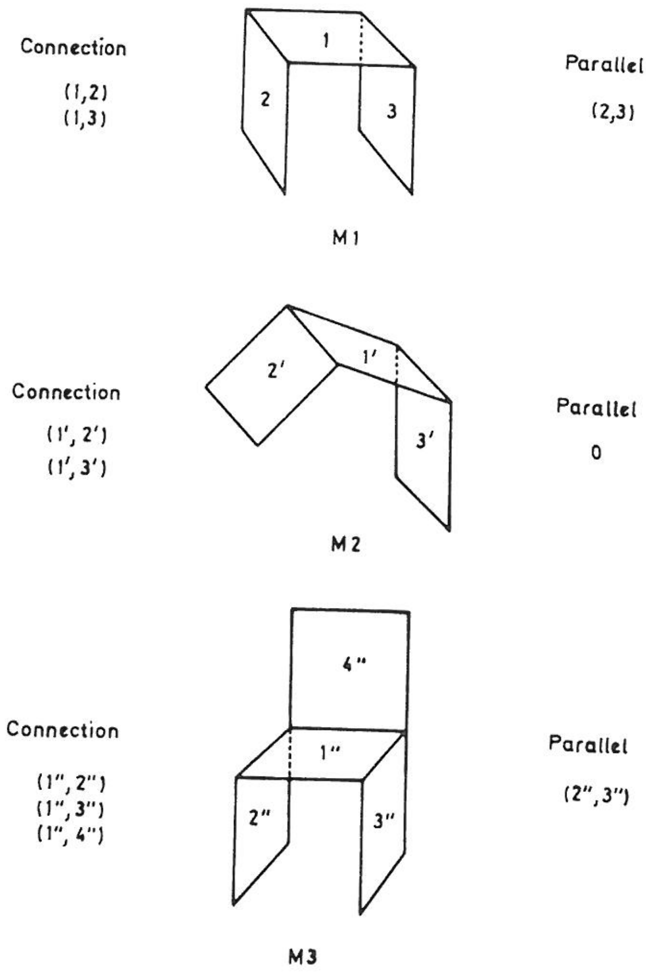


Fig. 7. An object model M1 and two other models, M2 and M3, that are each a relational distance of 1 from M.

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 that 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

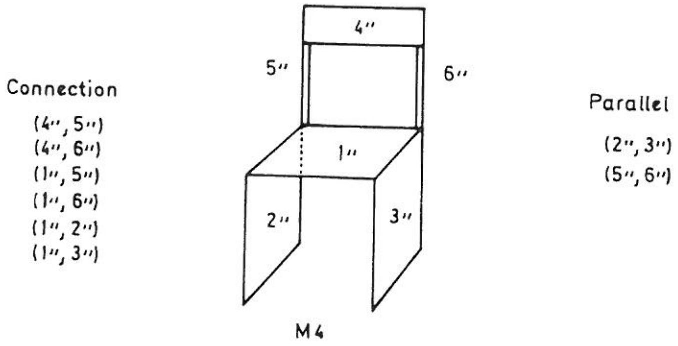


Fig. 8. A model M4 that differs from M3 by a relational distance of 6.

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 $R' \subseteq X^2 \times \mathcal{R}$, with $\mathcal{R}$ the set of real numbers, where 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 relation over part set B and property value set P. Let $f : A \to B$ by a one-one, onto mapping from A to B. We define the composition $r \circ f$ of attributed tuple r with f by

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

Assume that if $(b_1, \ldots, b_n, p_1, \ldots, p_n) \in S$ 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

$$e(t, s) = \begin{cases} \text{norm\_dis}((p_1, \ldots, p_m), (q_1, \ldots, q_m)) & \text{if } (b_1, \ldots, b_n, q_1, \ldots, q_m) \\ & \in S \\ 1 & \text{otherwise} \end{cases}$$

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

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

Total error and relational distance are defined as in Sec. 2.3.

3. ALGORITHMS FOR RELATIONAL MATCHING

In Sec. 2 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 that 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.

3.1. Backtracking Tree Search

Let R be an N-ary relation over part set A and let S be an N-ary relation over part set B. We will refer to the elements of set A as *units* and the elements of set B as *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 treesearch ($A, B, f, R, S$)
a:= first ($A$);
for each $b \in B$
      {
      $f' := f \cup \{(a, b)\}$;
      OK:= true;
      for each N-tuple $r$ in $R$ containing component a
        and whose other components are all in domain($f$)
            if $r \circ f'$ is not in $S$
            then {OK := false; break } endif;
      if OK then
            {
            $A' =$ remainder ($A$);
            if isempty($A'$)
            then output($f'$)
            else treesearch($A', B, f', R, S$);
            }
      endif
      }
end treesearch;
```

3.2. Backtracking with Forward Checking

The backtracking tree search has exponential time complexity. Although there are no known polynomial algorithms in the general case, there

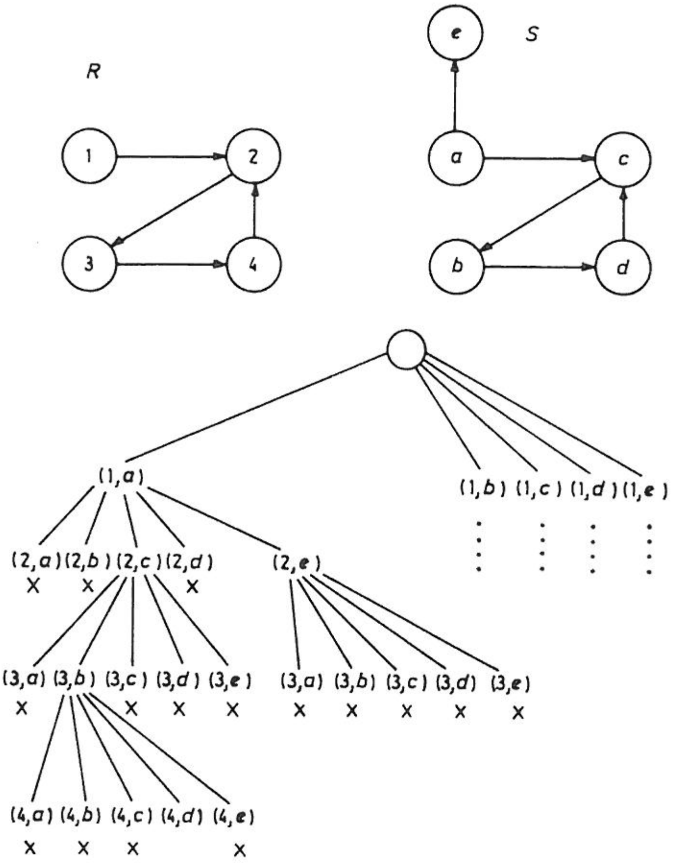

R
S
1
2
3
4
a
b
c
d
e
(1,a)
(1,b) (1,c) (1,d) (1,e)
(2,a) (2,b) (2,c) (2,d)
(2,e)
(3,a) (3,b) (3,c) (3,d) (3,e)
(3,a) (3,b) (3,c) (3,d) (3,e)
(4,a) (4,b) (4,c) (4,d) (4,e)
X

Fig. 9. The backtracking treesearch to find a homomorphism from $R = \{(1,2), (2,3), (3,4), (4,2)\}$ to $S = \{(a,c), (c, b), (b, d), (d, c), (a, e)\}$. An "X" under a node indicates failure. The only homomorphism found is $f = \{(1, a), (2, c), (3, b), (4, d)\}$.

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 $b' \in B$, then instantiation of (a, b) fails immediately. The backtracking tree search with forward checking is as follows.

```
procedure forward_checking_treesearch ($a, b, f$, FTAB, $R, S$)
a := first ($A$);
for each $b \in B$
    if (FTAB($a, b$) == 1)
    then
        {
        $f' := f \cup \{(a, b)\}$;
        $A'$ := 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_treesearch
                    ($A', B, f'$, NEWFTAB, $R, S$);
            }
        endif
        }
    endif
```

```
**procedure** update(FTAB, $a, b$, future_units, $B, R, S, f'$)
update := false;
**for** each $a' \in$ future_units
      **for** each $b' \in B$ **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** $((a, a') \in R$ **and not** $((b, b') \in S))$ or
   $((a', a) \in R$ **and not** $((b', b) \in S))$
**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 that are already instantiated must be examined. The code for N-ary relations R and S is as follows:

```
**procedure** compatible($a, b, a', b', R, S, f'$)
$f'' := f' \cup \{(a', b')\}$;
compatible := true;
**for** each $r \in R$ containing $a$ and $a'$ whose other components
  are in domain $(f'')$
        **if** $r \circ f''$ 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.

3.3. 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 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.

So 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 will guarantee 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 is 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, then a list of N ordered pairs of the already instantiated units and their corresponding labels specifies 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.

4. 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 use discrete relaxation and operate on sequential computers and briefly discussed parallel algorithms for multiprocessor systems. We feel that the current trend toward massively parallel architectures will produce a number of new algorithms that can rapidly solve any relational matching problem. Such algorithms will be important in the solution of complex vision problems.

REFERENCES

1. L.G. Shapiro and R.M. Haralick, "A metric for comparing relational descriptions", *IEEE Trans. PAMI* **7** (1985) 90-94.
2. R.M. Haralick and L.G. Shapiro, "The consistent labeling problem — Part I", *IEEE Trans. PAMI* **1** (1979) 173-184.
3. R.M. Haralick and L.G. Shapiro, "The consistent labeling problem — Part II", *IEEE Trans. PAMI* **2** (1980) 193-203.
4. A. Rosenfeld, R.A. Hummel and S.W. Zucker, "Scene labeling by relaxation operations", *IEEE Trans. SMC* June (1976) 420-433.
5. L.G. Shapiro and R.M. Haralick, "Structural descriptions and inexact matching", *IEEE Trans. PAMI* **3** (1981) 504-519.
6. L.G. Shapiro and R.M. Haralick, "Organization of relational models for scene analysis", *IEEE Trans. PAMI* **4** (1982) 595-602.
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, 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.* **34** (1985) 973-980.
9. J.R. Ullman, R.M. Haralick and L.G. Shapiro, "Computer architecture for solving consistent labeling problems", *The Computer Journal* **28** (1985) 105-111.