Using Spatial Structure in the Associative Retrieval of 2-D Line Drawings

Patrick W. Yaner and **Ashok K. Goel**
Artificial Intelligence Laboratory
{yaner, goel}@cc.gatech.edu

December 2002

Abstract

We consider the problem of associative image retrieval, focusing on retrieval of 2-D line drawings by example. We represent 2-D line drawings as semantic networks of spatial elements and relations among them. We describe a process for retrieving the drawings based on a structural analogy between the query and the stored images. We then present several methods of retrieving the drawings: the first family methods uses logical unification and resolution to accomplish the matching; the second family of methods heuristically prunes the stored drawings and then does the resolution and unification on the remaining drawings; two more methods treat the retrieval problem as a constraint satisfaction problem and use common CSP techniques for solving it; and the last two methods combine the heuristic step of the second method with the CSP technique of the third and fourth. We report on experimental results that compare the performance of these methods on computer-based libraries of drawings. A surprising result of our work is that for the fastest of these methods two stage retrieval appeared to offer no benefit over one stage retrieval.

1 Introduction

This paper deals with the problem of content-based, or associative, image retrieval, i.e. retrieval of images from a computer-based library by example. Associative image retrieval involves several issues such as (a) characterizing the content of an image, (b) characterizing measures of similarity between the contents of two images, (c) representation of the content of the query image, (d) representation, indexing and organization of images stored in the external memory, and (e) methods for probing the memory and retrieving images similar to the query. Almost all earlier work on associative image retrieval has characterized content of an image, and similarity between two images, either in terms shapes of objects and their locations or color and texture distributions. As a result of these characterizations, earlier work typically has relied on numerical methods to compute and compare objects shapes and locations and statistical methods to compute and compare color and texture distributions.

With a few notable exceptions, which we discuss later, earlier work on associative image retrieval has not investigated characterization of the content of an image in terms of its *spatial structure*, i.e., in terms of *spatial relations* amongst elements in the image. This characterization of the content of an image raises several questions: how might we represent the spatial structure of an image, how might we characterize similarity between the spatial structure of two images, how might we index the stored images, and, given a query image, how might we retrieve similar images? We focus specifically on the issue of *retrieval of images given the representation of spatial structure using separate semantic networks for each image*. As such, we are essentially dealing with the problem of associative retrieval of semantic networks [15, 16].

In order to investigate content-based image retrieval based on the spatial structure of images, our work focuses on the domain of 2-D line drawings composed of spatial elements such as lines, circles and polygons. In this domain, color and texture are not present, and computation and comparison of object shapes and locations is simple, which enables us to isolate the issue of spatial structure. The specific task in this work assumes a computer-based library of 2-D line drawings along with their representations, takes as input a representation of a 2-D line drawing as a query, and its desired output is a set of drawings retrieved from the library that are similar to the query. The goals of this work are: (a) to re-characterize the problem of image retrieval as a problem of associative retrieval of semantic networks, (b) explore the structure of the problem by developing and applying a series of standard methods, and (c) to investigate whether these methods work, at least in a laboratory setting. Along these lines, we take "spatial structure" to mean the qualitative arrangement of the various shapes in a line drawing.

We represent the spatial structure of an image in the form of a semantic network in which the nodes represent the spatial elements and the links are labeled by the spatial relations among them. We characterize similarity between two drawings in terms of *subgraph isomorphism*: a stored drawing is judged to be a similar to a query drawing if the representation of the spatial structure of the query can be found in, the representation of the spatial structure of stored drawing. We use symbolic methods to compare the spatial structure of the query with the spatial structure of the stored drawings and retrieve drawings similar to the query.

In this paper we first describe a general computational architecture for retrieval of 2-D line drawings and explain our representation of the drawings. We then present two families of methods for associative retrieval of drawings based on their spatial structure. The first family of methods uses logical resolution and unification to perform the matching on each image in the library. The second treats the retrieval problem as a constraint satisfaction problem (CSP) and uses a common CSP technique for solving it. We present experimental results that compare the performance of these methods on three computer-based libraries of line drawings. We also discuss some ideas for improving these methods with the goal of associative, content-based retrieval of 2-D line drawings.

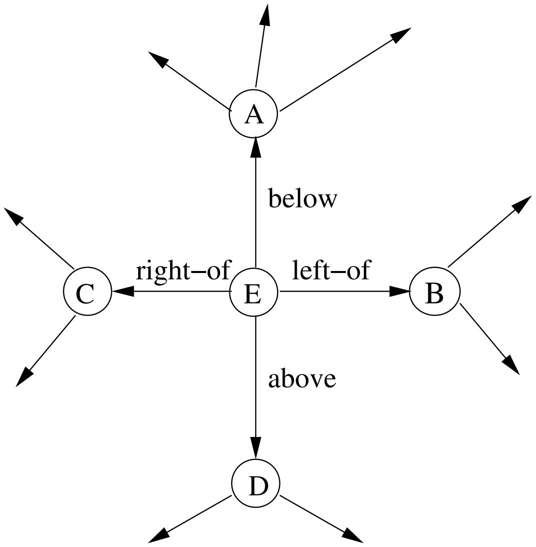

Figure 1: Part of a relational description represented as a semantic network

2 Associative Retrieval

We assume that the images can be described in a semantic network formalism. The problem, then, is to match the description of a target[1] with descriptions of source images from memory, and return all source images that do match (we should emphasize that networks for each image are separate). We consider the case in which "matching" means subgraph isomorphism, which is to say, the description of the target must be isomorphic to some portion of the description of the source. Intuitively, this means that, say, if the target has a square to the left of a circle, then the source must have square to the left of a circle (to use an elementary example). What this means will become more apparent when we discuss knowledge representations.

The query image, perhaps, may not be the same sort of image as those in memory; it may be a sketch. The issue is tangential to our point, however. That image can be described (or so we assume) as having some number of elements such as strokes or shapes or color or texture regions or other salient visual objects. These objects have some relational properties between them. For instance, one may be to the *left of* another, or a circle may circumscribe another shape and thus be said to *contain* it. If we have an image in which some set of objects or visual elements A, B, C, and D are, respectively, above, to the right of, to the left of, and below some fifth object or element E, the portion of the semantic network representing these objects and their interrelations might look something like figure 1.

Part of the matching task, then, will be to find another image whose description

---

[1] We use the terms "target" and "query" interchangeably. Likewise, we use the term "source" to refer to an image in memory to be matched with a target (or query). This is intended to be consistent with the literature on analogical reasoning.

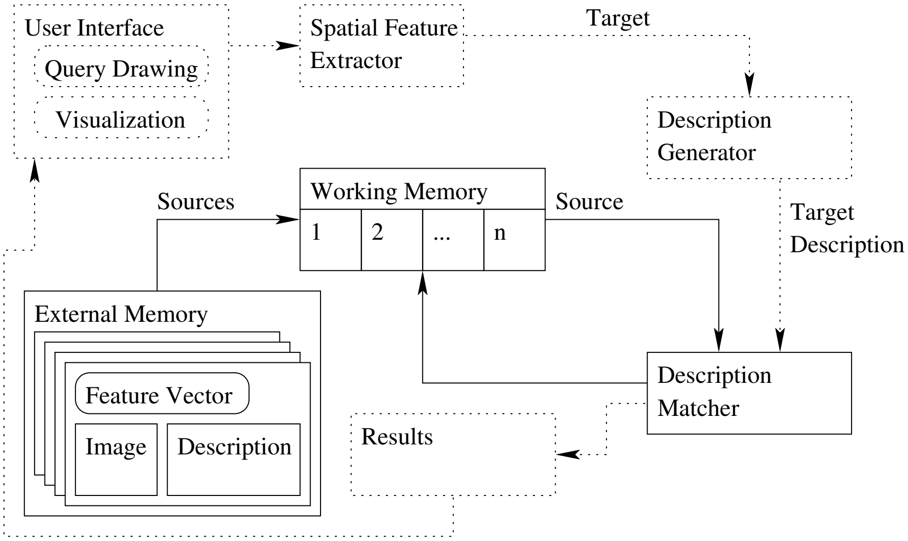

Figure 2: System Architecture—solid lines indicate the components and interactions dealt with in this paper

has at least those five elements with at least those relationships between them. The "at least" part is important, here: there may be other elements in the matched image, there may even be other relationships, but we're only concerned about those that are captured by the target (query) image. The complete task is, as we've said before, that of *subgraph isomorphism*: the complete target description, as represented in a semantic network, must be present within the description of any source that is returned as a match. Thus, our task is to represent the content of an image in a relational description—and specifically in the form of a semantic network—and match it to source images by projecting the labeled graph that is the semantic network representing the target image's description onto the networks for the source image descriptions.[2]

2.1 Architecture

We describe, here, a general architecture for solving these sorts of problems. Since the problem is one of matching sources to targets from memory, clearly there must be a memory of images together with their relational descriptions. Some work in this area (e.g. [19]) has considered hand-annotated images, but we are dealing with images in which the description is computed by some process, so that process must be a component of this architecture, as it must run on the target at the very least (presumably it has already been run on all of the images in memory, and they have been annotated already with their descriptions). Also, clearly, there must be a description matcher that actually computes matches between source and target. Finally, some means of gathering the

---

[2] This raises an interesting question of what sorts of properties of images can be captured in a semantic network. We do not address this issue in this paper.

results that are to be returned is necessary.

In some methods that we discuss below, we consider doing an initial retrieval based on feature vectors to quickly compute potential matches between targets before the (potentially expensive) description matcher is run. This is to narrow the set of possible matches and potentially make the retrieval faster. This requires several things. First, it requires a working memory in which to store the potential matches. Second, the feature vectors must be generated somehow, so a feature extractor must be run on the target, and presumably all images in memory, as well, so that they each already have a feature vector associated with them.

Our basic computational architecture for image retrieval, illustrated in figure 2, consists of (up to) six basic components [24]:

1. An external memory of images indexed by feature vectors
2. A feature extractor for generating feature vectors
3. A process that generates a semantic network describing the contents (spatial structure in this case) of an image.
4. A process that matches a target's description (semantic network) to source descriptions from memory
5. A working memory with potential sources to match with the target query
6. A user interface for drawing the query and visualizing the responses

This architecture can support a retrieval process consisting of two stages: reminding (or initial recall), and selection (or matching). The first stage takes as input a query example and returns as output references to stored images whose feature vectors match that of the target. The stored images, which in general are external to this architecture (though we show them anyway), are indexed by feature vectors describing their spatial elements; of course the feature vector for the query is constructed dynamically. Pointers to those images with sufficiently similar feature vectors (according to some appropriate criteria) are brought into the working memory. The reason for this decomposition of the retrieval process into stages is that matching in the second stage is computationally expensive to perform; the first stage attempts to minimize the number of images over which the matching must be performed. Note that this stage may or may not be present in some particular system. If it is not, then the working memory and feature extractor would go away, and images in memory would no longer be indexed by feature vectors. Several of the methods described herein are one stage methods of exactly this sort.

In the selection stage, the semantic networks of the images in working memory are matched with that of the query example. The images that match the target sufficiently well (that is, their descriptions match, as discussed above) are collected and returned to the user. Semantic networks describing the spatial structure of the image are constructed for each source image when they are entered into library, and, of course, the semantic network for the query is generated dynamically. The matching process does not depend on the specifics of the representations in the semantic networks, though results of matching naturally depend quite heavily on the specific representations. Hence,

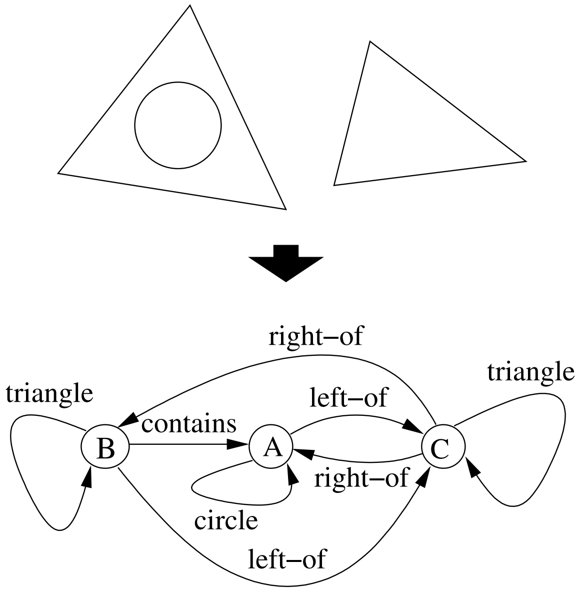

Figure 3: An example three-node semantic network in our language

the architecture is relatively independent of any particular representation used for the spatial structure; it only requires that the knowledge representation be based on the semantic network formalism.

3 Knowledge Representation

As indicated above, the first stage of the retrieval process uses partial knowledge of the contents of a drawing in the form of feature vectors, and the second stage uses more complete knowledge of the drawing's contents in the form of semantic networks. Here we discuss these issues in more detail.

3.1 Feature Vectors

The first stage of the retrieval process uses a vector of features, i.e., a vector of attribute-value pairs, as a heuristic to gauge the potential of a source drawing matching the query drawing. The feature vector simply is a multiset of the object types contained in a drawing. Given a list of object shapes, we represent the feature vector as a mapping

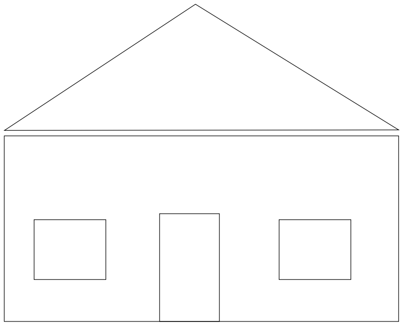

Figure 4: An example of a target image

from object type to its multiplicity:

$$\begin{aligned} Rectangle &\longmapsto 2 \\ Circle &\longmapsto 3 \\ Triangle &\longmapsto 1 \end{aligned}$$

A drawing is recalled if the multiset of shape types contained in it is a superset of that of the query. The method scans all stored drawings, calculating whether or not the multiset of objects in the target is a subset of the multiset of objects in each source image, returning those for which this is the case. That is, if T is the feature vector for the target, and $S_1, S_2, \ldots, S_k$ are the feature vectors of the images currently in memory, then the method returns those images for which $T \subset S_i$. The system recognized individual lines, triangles, rectangles, and ellipses (circles and squares are special cases of ellipses and rectangles, in which the height and width are equal—however, they are not treated as being of a separate type).

Of course, more sophisticated systems could also be devised using, say, some primitive and easily calculated spatial relationships (see, for example, [11]). Given a relational description like that in figure 3, a more interesting feature vector might look something like this:

$$\begin{aligned} triangle &\longmapsto 2 \\ circle &\longmapsto 1 \\ left\text{–}of &\longmapsto 2 \\ right\text{–}of &\longmapsto 2 \\ contains &\longmapsto 1 \end{aligned}$$

In this case we're now using the labels on the links as the "features" in our feature vector. In the context of the problem, this makes sense, since the matching is based on subgraph isomorphism, and can only map edges with the same label. As such, computing the multiset subset on these sorts of features should add a bit more information

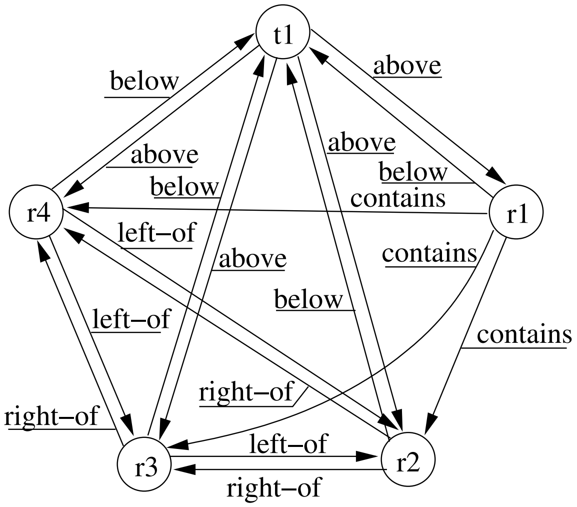

Figure 5: Relational description of the image in figure 4

beyond simply the object types. We also use this scheme for feature vectors—we have developed methods for both.

3.2 Relational Descriptions

The representation of the spatial structure of a image depends very heavily on what is meant by "spatial structure." Here we take it to mean the qualitative arrangement of the various shapes in a line drawing. Given that, it still could mean many different things, depending on how one chooses to represent the arrangement of a set of shapes, and precisely what one means by "arrangement." We do not attempt to address this question, here. Rather, we are using an architecture and a method that does not depend too heavily on what representation is chosen. Nevertheless, reasonable constraints must be placed on the form of the representation.

Since we actually did build a system (several, in fact), we had to fix a language for relational descriptions. That language was developed for proof of concept. However, the relational descriptions were automatically generated, and as such we had to settle on precise definitions of the terms. The five relation types we had were "left-of", "right-of", "above", "below", and "contains". Some of these are a little redundant (e.g. "left-of" usually implies "right-of" as well), but not entirely, as you'll see. The automatic generation works by taking the input drawing (in XFig format) and comparing every pair of shapes using the available predicates. If a particular predicate holds, a link is added between the associated nodes in the semantic network, with the appropriate label. The definitions of these predicate are as follows ($s1$ and $s2$ are two shapes appearing in some drawing):

left-of $s1$ is "left-of" $s2$ if and only if the centroid[3] of $s1$ is to the left of the centroid

---

[3] The centroid of a shape is the average of the centers of its vertices; in the case of ellipses and circles we

of $s2$, the left edge of $s1$ (meaning the left-most extremity) is at least 20% of the width of $s1$ *past* (on the left of) the left edge of $s2$, *and* the right edge of $s1$ is at least 20% of the width of $s1$ *past* (on the left of) the right edge of $s2$

right-of $s1$ is "right-of" $s2$ if and only if the centroid of $s1$ is to the left of the centroid of $s2$, the left edge of $s1$ is at least 20% *past* (on the right) the left edge of $s2$, *and* the right edge of $s1$ is at least 20% *past* (on the right) the right edge of $s2$

above $s1$ is "above" $s2$ if and only if the centroid of $s1$ is above the centroid of $s2$, the top edge of $s1$ is at least 20% *past* (above) the top edge of $s2$, *and* the bottom edge of $s1$ is at least 20% *past* (above) the bottom edge of $s2$

below $s1$ is "below" $s2$ if and only if the centroid of $s1$ is below the centroid of $s2$, the top edge of $s1$ is at least 20% *past* (below) the top edge of $s2$, *and* the bottom edge of $s1$ is at least 20% *past* (below) the bottom edge of $s2$

contains $s1$ "contains" $s2$ if and only if the bounding box for $s1$ properly contains the bounding box for $s2$, where the bounding box of a shape is defined by the top, bottom, left, and right edges as defined above (top-most, bottom-most, left-most, and right-most extremities).

Each of these predicates is applied to each pair of shapes in the image, and when the predicate is true (by the above definitions) of some pair of shapes $s1$ and $s2$, then a link from node $s1$ to $s2$ is added to the network. In figure 3, the image contains three shapes—two triangles and a circle. In the network, A represents the network and B and C represent the two triangles, B being the one containing circle A. As a more interesting example, the image in figure 4 would be represented by a semantic network like that in figure 5.

3.3 Reasoning

Details of both of the matching methods are discussed in the next section, but here we discuss how this knowledge representation is used in these methods.

Resolution and unification treat a semantic network as a conjunction of ground literals, where each ground literal is a relation that holds between two constants or variables representing nodes in the semantic network (and, hence, the corresponding visual objects). A semantic network with nodes corresponding to spatial objects and links corresponding to spatial relationships between these objects is represented in logical form as a conjunction of ground literals: the spatial relations are the predicates, and the shapes are the arguments. For the target drawing, we take the constants corresponding to the objects as existentially quantified variables instead of constants. Given descriptions of the target and source drawings, we add the source description to a knowledge base (a local buffer), and send the target description (with existentially quantified variables) as a query to the knowledge base. Resolution succeeds only when the terms of the target (the links, in other words) can be applied to *some set* of constants (spatial objects) from the source description.

---

used the proper geometric center

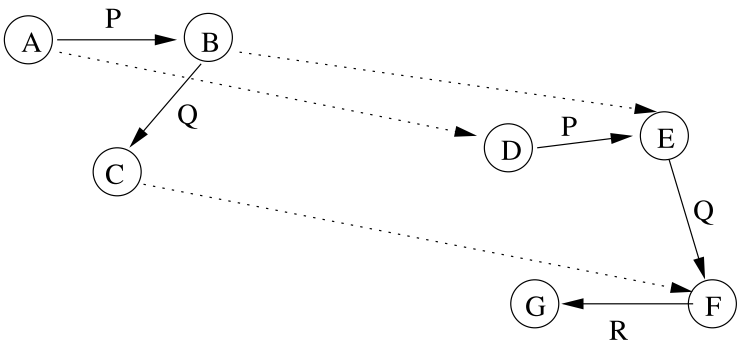

Figure 6: A matching between two simple networks

For instance, if our target is $P(A, B) \wedge Q(B, C)$, where A, B, and C are constants, and our source is $P(D, E) \wedge Q(E, F) \wedge R(F, G)$, where D, E, F, and G are constants, we first turn the target description into $\exists x, y, z P(x, y) \wedge Q(y, z)$, where x, y, and z are variables, then resolve this with the source description. Resolution, being essentially a *reductio ad absurdum* proof method, works by negating the query and using the resolution rule of inference to cancel out matching terms. In this case the unification of the two would bind x to D, y to E, and z to F, and the result would cancel out all of the terms of the target. This would lead, eventually, to a contradictory conclusion and success. The two networks corresponding to these logical terms, and the associated matching, are shown in figure 6.

What has really been done is resolution has been used for it's unification step. That is, we are using unification to construct bindings from nodes in the target network to nodes in the source, and resolution is simply the process by which global consistency is ensured.

In the constraint satisfaction method, the variables in the query description are taken as variables to which values must be assigned. These values are constants (i.e. nodes) from the source descriptions. The constraints can be found from the query terms themselves: all of the relations that hold between a variable v_i and v_j must hold of the values (nodes) to which they are assigned. This forms the set of constraints which must be satisfied.

It is important to point out a critical difference between the two methods: the resolution and unification method is a test that is applied to each item in memory separately, whereas the CSP-based method can be run on *all* the images in working memory *at once* [4]. If we want to return all matches, the CSP problem can be rephrased as the problem of finding *all* variable assignments that satisfy the constraints, rather than simply finding a single assignment that satisfies the constraints.

---

[4] not necessarily in parallel, but the domains of the variables can range over the entire scope of memory

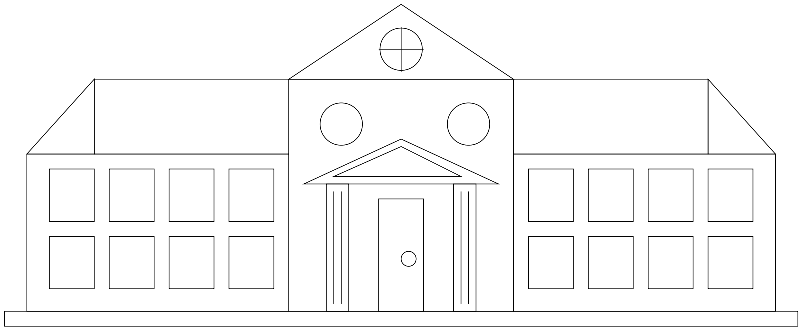

Figure 7: An example of a source image. This image was retrieved by the image in figure 4; the idea is that the shapes in that figure can be found in this one in approximately the same basic arrangement.

4 Methods

The core of the sorts of systems we're describing here is the matching process. This process finds a correspondence between the query drawing and the source drawings in working memory, eliminating drawings with which no correspondence can be found. We have used three basic kinds of methods for doing this: a depth-first search using logical resolution and unification, and two methods for constraint satisfaction.

For these methods, we have some test data which were used to test their execution. The test data consists of a library of images for memory. In fact, there are three libraries. In the case of the resolution and unification methods we had a small set of eight images for populating memory (with three or four objects in each one), and nine targets, each with two or three objects. This test set was mostly intended to demonstrate that the algorithm was actually working correctly, and not as an exhaustive demonstration of the system's capabilities. Call this test set A.

A second test set consisted of 28 source images for memory, ranging from 3 to several dozen objects in each one, and the number of terms in the relational description (or edges in the semantic networks) ranged from a couple of dozen to three or four thousand in some cases. There were 15 targets in this test set, the first nine of which were identical with those in test set A. Most of the source images were new, as well (not all of the eight sources from test set A were used in this test set, but some were). Call this test set B.

A third test set consisted of 42 source images (for memory), ranging from 3 to over 50 objects in each one (the average was around 12 or so), and the number of terms in the description ranged from a couple of dozen to over eight thousand thousand (again, by number of terms we mean the total number of links or edges in the semantic network). There were 21 targets with this test set, with two to five objects in each, and up to several dozen terms. The first fifteen of these targets were identical with the fifteen targets from test set B (and, hence, the first nine were from test set A), and some of the source images were taken from test set B, as well (but more than a dozen were added, and some of the old ones were altered). Call this test set C.

```
function RESOLVE($c_1$, $c_2$) returns target description or $\perp$
$unified \leftarrow$ FALSE
$unifier \leftarrow \emptyset$
$retval \leftarrow$ NIL
for all $\ell \in c_1$ do
    if $\neg unified$ then
        $x \leftarrow$ UNIFY(FIRST($\ell$), $c_2$, $unifier$)
        if $x \neq \perp$ then
            $unifier \leftarrow x$
            $unifed \leftarrow$ TRUE
    else
        $retval \leftarrow$ APPEND($retval$, $\{\ell\}$)
if $unified$ then
    return APPLY-UNIFIER($unifier$, $retval$)
else
    return $\perp$
```

Algorithm 1: The resolution and unification steps

Work in image retrieval (and information retrieval more generally) often tests *recall* and *precision*. The former measures the fraction of those images that should have been recalled by the search that actually were (according to some metric), and the latter measures the fraction of images actually recalled that should have been. Since we are defining the similarity metric in the problem to be one of subgraph isomorphism, any algorithm that attempts to solve the problem will have a precision and a recall of 100%, regardless of the domain.[5] As such, these are not terribly relevant measures given our characterization of the problem. Instead, we need to measure performance. Subgraph isomorphism is known to be NP-Complete, and so any method that solves it generally will break eventually. The question is how far it can be pushed before it reaches its limit. Each of the methods run can be looked at a a search of some sort, and so we have measured two things: states generated and running time. The running time is potentially misleading, and hence we have state counts. On the other hand, the time required to generate a state and test it may not correspond too well between different sorts of methods, and so the relative running times are interesting to some extent.

4.1 Resolution and Unification

Given a query example, our first set of methods all test each source drawing separately for subgraph isomorphism. That is, they determine if there is some correspondence between the nodes of the query and the nodes of the source that preserves the link structure. They do this by actually trying to find one: they match links, one at a time, in the source with the query. When two such links share a label, they attempt to

---

[5] On the other hand, in a particular domain there may be a question of how well a particular language for representing spatial structure captures the “right” information with which to make a matching, and in this case recall and precision may become relevant again. However, this issue is beyond the scope of this paper.

```
function MATCH($target$, $source$) returns TRUE or FALSE
  $Open \leftarrow \{target\}$
  while $Open \neq \emptyset$ do
    $current \leftarrow$ POP($Open$)
    for all $\ell \in source$ do
      $term \leftarrow$ RESOLVE($current$, $\ell$)
      if $term \neq \perp$ then
        if $term =$ NIL then
          return TRUE
        PUSH($term$, $Open$)
```

Algorithm 2: The core matching procedure

find a correspondence between their associated nodes. If the correspondence conflicts with existing correspondences already made, it is abandoned; otherwise it is left in. When a link in the query is matched with a link in the source, it is removed from the query description, so that it cannot be matched again. The search is completed by using the query description as a sort of “state” in a graph search. The goal is to eliminate all terms, and the operators take terms from the source and match them with the target as described above.

Given a query description and a term from the source, the resolution procedure finds the source term within the target, unifies them, and returns the resolved result. In terms of semantic networks, given the query network and an arch in the source (an edge and the nodes it connects), the resolution procedure finds an arch in the query to map onto the source arch, or returns failure ($\perp$) if none can be found.

For unification, we don’t actually need a complete unification algorithm; we only need a fairly basic pattern-matching procedure that matches, say, `left-of(` x `,` y `)` with `left-of(r-1, r-2)` by substituting the constants `r-1` and `r-2` for the variables x and y. If a term is simply a list, then this procedure runs in linear time (in the length of a term, which here is three–the predicate name and two arguments): it first compares predicate names, then compares either a variable with a constant, yielding a potential substitution, or else a constant with a constant. The result is either failure or a (potentially empty) list of bindings of variables to constants.

The resolution step, then, is straightforward. Resolution as a rule of inference that works as follows: given, say $\neg\mathtt{P}(x, y) \vee \neg\mathtt{Q}(y, x)$ and $\mathtt{P}(\mathtt{a}, \mathtt{b})$, the resolvent of the two is $\neg\mathtt{Q}(\mathtt{b}, \mathtt{a})$. The resolution step tries to unify the source term with each of the terms in the target description, looking for a successful match. When it finds one, it applies the binding to the whole target term, removes the matching term, and moves on. Algorithm 1 gives the details of the method.

The matching procedure begins with the query as an initial state, and continually resolves each of the terms in the source with it one by one in a depth-first manner until an empty target description has been reached, at which point success is reported. If at some point there are no more possible paths to take, then failure is reported. The basic outline of the procedure is like this:

Target	time	states
target A1	41 ms	217
target A2	15 ms	199
target A3	15 ms	153
target A4	16 ms	217
target A5	26 ms	153
target A6	15 ms	196
target A7	12 ms	115
target A8	3,759 ms	66,948
target A9	19 ms	196
Average	435 ms	7599

Table 1: Test results for 1 stage resolution method on test set A.

1. Choose a term ℓ from the source description, and resolve it with the current state (i.e. the target description) as described above.
2. If this resolution fails, go back to step 1. Otherwise, we have a correspondence between an edge (term) in the target and the edge represented by ℓ in the source, as well as correspondences between two nodes (objects or object types) in the target and two nodes in the source.
3. If all edges in the target have been mapped, return success. Otherwise, save the mapping and return to step 1.

Algorithm 2 describes this procedure in detail.

We implemented this algorithm in ANSI Common Lisp and ran it on a workstation using the set of eight sources and nine targets described above. The results are described in table 1.

In and of themselves these numbers mean nothing. They simply serve to demonstrate that the program does, in fact, run, and on a very simple test set it can perform retrieval in under 1 second in most cases, and in the worst case under five seconds. For most of the queries, the number of states generated was in the hundreds, but for one of the targets it was over sixty thousand. These numbers will become more relevant when we compare them to some other results later on. It should be pointed out that this system was too slow to run on test sets B or C—it did not finish a single target within a reasonable amount of time (meaning hours).

4.1.1 Two-stage retrieval

The method described so far is essentially running this rather expensive test (algorithm 2) on every image in memory. We have mentioned two-stage retrieval already, and the use of feature vectors. A natural way to avoid having to run this expensive test on every image in memory is to use superficial features of the images to retrieve candidates which *might* match, and only run the matching procedure on those (or, equivalently, to prune out those which will very obviously *not* match)

Target	time	states
target A1	2 ms	35
target A2	10 ms	114
target A3	13 ms	127
target A4	16 ms	145
target A5	1 ms	17
target A6	10 ms	111
target A7	1 ms	37
target A8	1,294 ms	16,302
target A9	9 ms	111
Average	151 ms	1889

Table 2: Test results for 2 stage resolution method using test set A

Section 3.1 already described the two schemes we implemented: object types and relation types. The implementation simply adds a slot to each item in memory with this feature vector, generating it for each image in memory when that image is added. The recall filters memory by comparing the feature vector of the target with those in memory and returning only those that match the target (that is, the count on each feature is higher than the corresponding one for the target).

Implementing this in the system just described resulted in the numbers from the original test set (of 8 sources and 9 targets) summarized in table 2. The running times and state counts have dropped noticeably, now. The two stage retrieval has seen some improvement. Again, however, this program was too slow to run on test set B.

A slightly more sophisticated piece of information to capture in the feature vectors, and a little closer to home in terms of relevance to the matching process, would be to capture relation "types" (meaning labels: left-of, above, contains, etc.). It's exactly the same as before, with the multiset subset comparison, but now each number corresponds to the number of edges with a particular label in the relational description. Implementing and running this new first stage, however, resulted in nearly identical performance on the test set above, with the only difference being with target 8, for which it took 11,431 states (about five thousand fewer than the original two-stage method) and 626 ms (about half the time). And again, test sets B and C proved too much for it.

4.2 Constraint Satisfaction

From what we have seen so far, the problem we are solving is essentially one of matching objects (variables and constants) in the source and target under the constraints imposed by the terms in which they appear. The target has a set of variables to be matched to some constant (i.e. value), and the relationships between these variables impose constraints on the values to which they can be matched. This sounds like constraint satisfaction [13]. We describe, here, a method for solving the problem using constraint satisfaction using backtracking, which we describe later.

A constraint satisfaction problem is one of assigning values to variables under some constraints [13]. In general, these constraints are represented as relations (i.e. sets of

```
function InitDomains returns Initial Target Domains
Let $Nodes$ be a list of all the nodes in the query
$InitDomain[*] \leftarrow \emptyset$
for all $w \in Nodes$ do
    Let $Arches$ be a list of all the arches in which $w$ appears
    $MappedNodes \leftarrow$ none
    for all $arch \in Arches$ do
        Let $Candidates$ be a list of all nodes from memory incident on an arch
        whose label matches $arch$ (either incoming or outgoing as appropriate)
        if $MappedNodes =$ none then
            $MappedNodes \leftarrow Candidates$
        else
            $MappedNodes \leftarrow Candidates \cap MappedNodes$
    $InitDomain[w] \leftarrow MappedNodes$
return $InitDomain[*]$
```

Algorithm 3: Initialize Domains

ordered n-tuples) over subsets of the variables. Each variable in a CSP has a domain of allowed values (possibly the same domain for all variables). The size of the domain(s) can have a large influence on the difficulty of solving the CSP, but mostly the nature of the constraints determines the difficulty of the CSP.

The constraint satisfaction methods described here treat the nodes in the query as variables to be assigned values. The potential values are the nodes from the source descriptions in memory, all of which are considered at once—that is, we are no longer performing a separate test on each source in memory, as with the resolution and unification methods; we are now running a search procedure on the entire memory considered collectively. The constraints on the values assigned to the variables (the target nodes) are precisely those imposed by the subgraph isomorphism problem: if nodes A and B from the target are to be matched with nodes X and Y from memory, respectively, then, first, X and Y must be in the same description; second, all relations that hold between A and B must also hold between X and Y, respectively[6]. If these constraints are met, then A can be matched with X and B can me matched with Y. Here our constraints are all either unary (say, A is a circle—a type constraint), or binary (say, A is left of B). The only exception is the constraint that all values be from the same description, but this can be inferred from the binary constraints.

4.2.1 Generate and Test

The key advantage in the methods chosen to solve the problem is not actually the constraint satisfaction method (which is rather unremarkable), but the fact that a couple of passes can be made beforehand to reduce the domains of the variables before the backtracking search runs. This serves to reduce the total number of possible solutions

---

[6] Observe, here, that the latter condition implies the former.

```
function REDUCEDOMAINS returns Reduced Target Domains
$InitDomain[*] \leftarrow$ INITDOMAINS()
Compute the document ids for each list in $InitDomains[*]$
Let $ReferenceList$ be the intersection across all of these lists
$Domain[*] \leftarrow \emptyset$
for all $w \in Nodes$ do
    $CurrentList \leftarrow \emptyset$
    for all $i \in InitDomain[w]$ do
        if the document id of $i$ is in $ReferenceList$ then
            Add $i$ to $CurrentList$
    $Domain[w] \leftarrow CurrentList$
return $Domain[*]$
```

Algorithm 4: Reduce Domains

that can be generated, and hence speed up the process (potentially by a lot). The first stage simply initializes the target domains to sets of values which have the same incoming and outgoing edges. The second stage reduces these domains by eliminating values that are not all in the same image. We describe these stages in more detail below.

This algorithm works by maintaining an index of all the arches across all of the source descriptions. It recalls individual terms from memory and puts them together to form the complete matching. This process is the core of the algorithm, and works in three phases: initialization of domains, reduction of domains, and finding the matching, where matching means subgraph isomorphism. The first two phases serve to limit the selection of values for each variable, and the third actually computes the matchings (in fact, it computes all possible matchings). Note that in the following, an *arch* is a link (relation) together with its two incident nodes (the arguments to the relation), the "relative position" of a node in an arch refers to whether it is on the incoming or outgoing end of the arch (in general, links in a semantic network are *directed*), and, to say one arch "matches" another means their links represent the same relation.

The first phase (initialize domains) is described by algorithm 3. It works by finding nodes in memory that "look similar" to the query nodes: if a query node A is incident on, say, three links whose labels are R, S, and T, then the algorithm builds a list of all nodes in memory—across all the source descriptions—that have *at least* three incident links with labels R, S, and T.

The second phase (reduce domains), described in algorithm 4, works by ensuring that the set of source descriptions (document ids) that are represented in the domain of (list of values for) each variable is the same. This serves to eliminate any value from the domain of any variable that does not come from a description represented in every other variable's domain.

The last phase (find matchings), described in algorithm 5, actually solves the problem. The basic procedure is one that generates all likely solutions, and go back and eliminate the ones that do not work. The test, here, is actual subgraph isomorphism: if A is related to B in the target, then the relations (links, edges) between $m(A)$ and $m(B)$ must include at least those that held between A and B, where $m(*)$ is a matching

```
function FINDMATCHINGS
$Domain[*] \leftarrow$ REDUCEDOMAINS()
Let $Mappings[*]$ be nil
{Create Combinations}
Let $i$ be 0
for each unique choice of one item from each list in $Domain$ do
    if All nodes in this list are from the same document then
        Let $Mappings[i]$ be this list
        Increment i by 1
{Eliminate Wrong Combinations}
for $i = 0$ to $|Mappings| - 1$ do
    for each distinct pair of nodes $a, b$ in $Nodes$ do
        for each relation $r$ that exists between $a$ and $b$ do
            if no relation $r$ exists between $Mappings[i][a]$ and $Mappings[i][b]$ then
                eliminate $Mappings[i]$
Each item (list) in $Mappings$ now corresponds to a projection from the query to
some document in memory.
```

Algorithm 5: Find Matchings

from target to source. This algorithm returns all valid matchings. The idea is that the first two phases have restricted the set of possible matchings so that there aren't nearly as many, now, as there would have been if pure generate-and-test had been done.

Discrimination Trees An issue with the method discussed above which has not been mentioned yet is indexing. Specifically, the method is constantly searching memory for individual terms that match a given pattern. This requires indexing the semantic networks in memory by the individual terms they contain. In test set C, there are over forty thousand terms across all 42 source images. Retrieving them requires a pattern-matching function—the same sort of simplified "unification" routine used in the resolution-based method. Running this routine on every item in a list of forty thousand is probably not the wisest design decision we could make.

Logical reasoning systems often use discrimination trees to index logical terms. Discrimination trees are a common tool in AI, but in this application the root node asks what the predicate is, the next node down in the tree asks what the first argument is, and so on. In some ways, this is a generalization of a common data structure known as a trie, often used to index words in a dictionary for use in spell checkers (for example). Our trees are going to be of fixed shape because of the limited nature of the terms (every term has two constants for arguments—never one or three, never functions as arguments, etc.), and we could probably replace it with a table of some sort. However, at the level of implementation the distinction is going to be lost, and the two methods will work more or less identically.

The system actually works by maintaining a separate tree for each predicate (there are five in our language, so there are five trees in the index). In each tree is actually a list of n items, where n is the number of arguments to the predicate in question. Since we're dealing with semantic networks, each predicate is actually an edge between two nodes, and so there are two arguments. Thus, the list is of length two. The first item under that list has two parts: the first part is a list of every edge with the right label across all source descriptions in memory, and the second is another list, this time of every node in every semantic network in memory. Under each node is, finally, a list of all the edges coming out of that node (i.e. terms for which that node is the first argument). Likewise, in our original, top-level list of two items, the second item has two items: the first is that same list of all edges with the appropriate label across all source descriptions in memory, and the other is another list of all of the *nodes* across all images in memory. Under each one of these is, again, a list of all of the edges that come into that node (i.e. terms for which that node is the second argument to the predicate). So, given a pattern such as $\mathbb{P}(x, y)$, the index looks up the answers as follows:

1. Find the tree for predicate $\mathbb{P}$.
2. There are two lists for $\mathbb{P}$. If x is a constant (i.e. a specific node), then lookup that constant in the second part of the first list for $\mathbb{P}$—this will be a list of edges that go out of the node referred to by x. Otherwise, if x is a variable, return the first part of the first list for $\mathbb{P}$—this will be a list of all edges named $\mathbb{P}$ in the graph.
3. If y is a constant, look up that constant in the second part of the second list for $\mathbb{P}$—this will be a list of all edges coming into node y. Otherwise, if y is a variable, return the first part of the second list for $\mathbb{P}$—this will be a list of all edges named $\mathbb{P}$ in the graph..
4. Compare the lengths of the lists from steps 2 and 3 (if the first item of a list is its length, this can be computed almost instantly). Take the shorter of the two.
5. Run a pattern-matching routine (such as a simple unification procedure) on each item in the list of answers you have, and return all those items that unify with (i.e. match) the pattern.

This algorithm can be generalized into a recursive algorithm that can index any single predicate calculus term (i.e. a predicate with any number of arguments that can be variables, constants, or functions, whose arguments in turn can be variables, constants, or other functions). However, we have no need for the general discrimination tree algorithm, here.

Test Results Table 3 summarizes the results of running this method on test set A. Three things are of interest: (1) the times are all four ms or less, and most are two, (2) the number of states generated for each target is an order of magnitude lower than for the resolution-based methods, and (3) target 8, which gave the resolution methods so much trouble, was retrieved in 3 ms and with only 3 states (the fewest possible—there were 3 objects, and hence 3 variables to match, so apparently the first two phases exactly determined the matching, leaving essentially no work for the matching phase). The results of running the method on test set B is summarized in table 4, and for test set C in table 5. The important thing to note about test set B is that it actually finished

Target	time	states
target A1	2 ms	42
target A2	2 ms	9
target A3	2 ms	12
target A4	2 ms	20
target A5	2 ms	6
target A6	2 ms	12
target A7	4 ms	16
target A8	3 ms	3
target A9	2 ms	12
Average	2.3 ms	14.7

Table 3: Test results for 1 stage constraint satisfaction method on test set A

all of the tests, and all but one in under 5 seconds, while generating thousands of states in most cases. In test set C, the important thing to note is that retrieval times are new up to several seconds, and the state counts are tens of thousands in most cases, while millions of states were generated for a few. Also, the last five targets never resulted in an answer after several hours of running.

4.2.2 Backtracking

Like the last one, this algorithm works by maintaining an index of all the arches across all of the source descriptions (using the same discrimination tree method described above), and recalling individual terms from memory to put them together to form the complete matching. Once again, there are three phases: initialization of domains, reduction of domains, and finding the matching. The first two phases are identical to the previous method, and are described in algorithms 3 and 4. The key difference, now is in phase three.

This last phase (find matchings), described in algorithm 6, is a backtracking procedure, now, that works essentially as a depth first search (once again). It proceeds in a way not unlike the resolution method did, assigning one variable from the target at a time. The only test is that the current assignment be consistent with the previous assignments made. The set of assignments made so far, as well as the domains of unassigned variables, are all stored in the current state. Algorithm 7 details this consistency check. The depth first search this time builds a complete search tree, returning all valid matches instead of just one (as the resolution method did).

Table 6 summarizes the test results of running this method on test set A. Not one target required a search of more than 8 states. Even target A8, which gave the resolution methods so much trouble, used a measly 3 states and 3 ms. This performance, however, is comparable to those of the generate and test method described above. A bit more information can be seen in test set B, summarized in table 7. Times on the first seven targets are similar, but the state counts, now are an order of magnitude lower, in the dozens or hundreds rather than the thousands. Target 8 saw a state count of 723 instead of over three hundred thousand, and (perhaps the most dramatic difference), target 11

Target	time	states
target B1	0.6 s	8,204
target B2	0.7 s	6,104
target B3	0.5 s	735
target B4	0.5 s	548
target B5	0.7 s	9,223
target B6	0.6 s	5042
target B7	0.9 s	12,244
target B8	3.2 s	304,250
target B9	0.6 s	5,042
target B10	4.0 s	14,005
target B11	58.3 s	17,654,995
target B12	4.2 s	286,304
target B13	2.1 s	28,895
target B14	1.2 s	5,128
target B15	4.2 s	357,008
Average	5.48 s	1,246,515

Table 4: Test results for 1 stage constraint satisfaction method on test set B

```
function FINDMATCHINGS
$n \leftarrow |Nodes|$
Let $Mappings[*]$ be nil
$k \leftarrow 0$
$Open \leftarrow \{(\text{nil}, \text{nil})\}$
while $Open \neq \emptyset$ do
    $(w, current) \leftarrow$ POP(Open)
    if $w = \text{nil}$ then
        $w \leftarrow 1$
    else
        $w \leftarrow w + 1$
    for all $j \in Domain[w]$ do
        if CONSISTENT($j$, $current$) then
            $new \leftarrow$ APPEND($current$, $j$)
            if $w = n$ then
                $Mappings[k] \leftarrow new$
                $k \leftarrow k + 1$
            else
                PUSH($(w, new)$, $Open$)
Each item (list) in $Mappings$ now corresponds to a matching from the query to some document in memory.
```

Algorithm 6: Find Matchings (Backtracking)

Target	time	states
target C1	2.9 s	29,160
target C2	3.1 s	24,905
target C3	2.5 s	5,455
target C4	2.6 s	5,432
target C5	3.2 s	36,954
target C6	2.8 s	22,554
target C7	3.7 s	39,452
target C8	20.8 s	3,160,394
target C9	2.8 s	22,554
target C10	16:04 m:s	359,116,143
target C11	13:25 m:s	324,153,998
target C12	27 s	3,330,498
target C13	10.1 s	63,777
target C14	6.8 s	74,157
target C15	17.4 s	2,069,391
target C16	2:43 m:s	2,560,188
target C17	-	-
target C18	-	-
target C19	-	-
target C20	-	-
target C21	-	-
Average	2:16 s	46,314,334

Table 5: Test results for 1 stage constraint satisfaction method on test set C. Targets 17 to 21 were not completed within a reasonable amount of time (several hours). Averages reported are over the first 16 targets.

```
function CONSISTENT($i$, $j$) returns TRUE or FALSE
if $current = \text{nil}$ then
    return TRUE
for all $i \in \{1, \ldots, w-1\}$ do
    if Not all relations between $i$ and $w$ can be found among the relations between $current[i]$ and $j$ then
        return FALSE
```

Algorithm 7: Consistency Check

Target	time	states
target A1	38 ms	8
target A2	1 ms	6
target A3	2 ms	6
target A4	2 ms	8
target A5	2 ms	4
target A6	4 ms	6
target A7	2 ms	6
target A8	3 ms	3
target A9	2 ms	6
Average	6.2 ms	5.9

Table 6: Test results for 1 stage backtracking method using test set A

was retrieved in seven seconds instead of nearly a minute, and while generating only about a thousand states instead of seventeen million.

Even more interesting results can be seen in test set C, summarized in table 8. Upon comparison to the generate and test method, first of all, it actually returned answers for all 21 targets. On the first seven, once again the state counts are an order of magnitude lower, closer to 1,000 than to 20,000. In target 8, the state count went from over three million to less than three thousand, a 99% reduction. The rest of the targets showed similar improvement in terms of state count. Notice, however, than running times are less predictable; even when there is a substantial reduction in the number of states generated, running time may not drop by that much, or at all in some cases. It's not immediately clear what this means, but for now we can simply say that it serves to underscore the unreliability of running time as an objective measure of performance.

4.2.3 Two-Stage Retrieval

If the two-stage retrieval sped up the resolution method, it seems reasonable to expect it to improve the performance of our constraint satisfaction methods, as well. Once again, the first kind of feature vector used was the object type comparisons, and the second was the relation types. The first stage (feature vector comparison) was the same one as used in the resolution-based method (section 4.1.1). The results, however, were a little surprising: both in terms of running times and state counts, there was virtually no difference *at all* between the original and the two stage constraint satisfaction methods. Furthermore, relation types once again made no difference at all over object types. The only savings observed was that on target 11 the running time of the generate and test method (section 4.2.1) dropped by about ten seconds and the state count went down by a proportional amount. On several other targets, similarly small savings (5-15%) were seen. Other than these, the running times were nearly identical and the state counts actually were identical; on average the drop in both was on the order of about 1%. Similarly small or nonexistent reductions were observed when the two stage methods were applied to the backtracking algorithm. It seems that the first stage of the retrieval, here, is not eliminating anything that couldn't have been eliminated in the first two

Target	time	states
target B1	1.1 s	393
target B2	1.0 s	519
target B3	0.6 s	62
target B4	0.5 s	47
target B5	1.0 s	620
target B6	0.9 s	251
target B7	1.6 s	707
target B8	4.0 s	723
target B9	0.9 s	251
target B10	3.6 s	53
target B11	7.3 s	1131
target B12	3.1 s	603
target B13	2.0 s	35
target B14	1.1 s	119
target B15	4.0 s	1251
Average	2.18 s	451

Table 7: Test results for 1 stage backtracking method using test set B

Target	time (s)	states
target C1	0:06	1064
target C2	0:06	828
target C3	0:03	248
target C4	0:03	289
target C5	0:06	1718
target C6	0:05	864
target C7	0:09	1524
target C8	0:34	2,722
target C9	0:05	564
target C10	0:26	1,029
target C11	1:13	13,093
target C12	0:20	3,089
target C13	0:10	43
target C14	0:07	500
target C15	0:20	2,167
target C16	0:38	848
target C17	0:49	4,099
target C18	0:30	1,322
target C19	0:36	3,732
target C20	0:48	1,851
target C21	3:00	3,574
Average	29.23	2991.9

Table 8: Test results for 1 stage backtracking method on test set C

phases of the methods described (initialization and reduction of the domains). There are other interpretations, which we discuss later.

4.3 Summary

The core of the resolution-based methods—the MATCH function described in section 4.1—is a depth-first search. In general, the time complexity of depth-first search is on the order of $O(k^d)$ in the worst case, where k is the branching factor of the state space and d is the maximum depth. Depth here refers to the number of terms in the query, and the branching factor is the number of terms in the source description, since at each step the set of operators is really the terms in the source available to resolve with the target. However, the space complexity, as with depth-first search in general, is only $O(kd)$, i.e. it's linear in the size of the problem. These numbers only hold per test, however, and the algorithm must repeat the test for each image in memory (or working memory, in the case of the 2 stage version). When the initial recall is not done (as in the 1 stage version), and assuming the largest source has k terms in it, and there are n sources in memory, the overall time complexity is $O(n \cdot k^d)$.

The time (and space) complexity of a generate and test constraint satisfaction algorithm in general is $O(k^d)$ for much the same reason: it covers the entire search space. However, in this case the k is the number of terms across *all* the sources. Using the terms from above, the new k is at most $n \cdot k$, so we now have an algorithm with time complexity of $O((n \cdot k)^d)$. There are two things to consider, here: first, the *largest* source may have k terms, but the average may be substantially less than that, and second, this algorithm prunes the domain size for each variable considerably in the first two steps (initialization and reduction of domains), and so the time complexity is actually much less than that. The resolution method must resolve a given source term with any term from the target at each step in the above formula. The generate and test method, however, is generating the states rather quickly, with very little processing, and taking most of the time in the checking of those final states (which represent complete assignments of values to variables). An interesting question to answer would be whether it is the last stage or the first two stages that make the most difference. One way to answer that would be to see what the output of the first two stages is; that is, by how much is the state space reduced in size by the first two stages? A second way to answer the question would be to explore the results of *only* the third stage versus the resolution and unification methods.

The complexity of the backtracking method is not captured very well in big-O notation; it still spans the same search tree. However, it cuts off large subtrees that the generate-and-test method does not, which may represent a substantial fraction of the search tree. In addition, once all leaves are generated, the generate-and-test algorithm only *then* goes through and tests them for validity. The backtracking method, on the other hand, amortizes this cost of consistency checking over the entire search process, checking as it goes. These two factors combined give a dramatic speedup over the generate and test method.

Note that the retrieval process and the selection methods are largely independent of the specifics of the spatial objects and spatial relations as long as they are represented in the form of a semantic network. Therefore, we believe that the process and the methods will work still well for an expanded vocabulary of object types and relation types with little degradation in performance. In fact, our simple language had links between every pair of objects in the network. This means that, in terms of constraint satisfaction, the constraint graph is a complete graph on n nodes. This is the worst-case scenario, in some sense, for constraint satisfaction, since it means there is little or no structure to be exploited by such improvements as forward checking, intelligent variable ordering, dependency-directed backtracking, and so on. A more complex language that, for instance, only compared "nearby" objects, had groups (and represented groups as nodes in the network), and had other such notions to complicate the representation would probably allow more sophisticated techniques to be used to exploit this new structure in the state space.

5 Discussion

The problem of associative image retrieval recently has received enormous attention in the literature on information retrieval and knowledge management. Extensive surveys of this literature show that almost all earlier work has focused on the use of numerical and statistical techniques for extracting and comparing low-level features such as color, texture, shape and spatial locations from bit-mapped encodings of complex images [12, 23, 21, 22]. Various approaches involve the making of histograms of various sorts (such as simple color histograms) representing some distribution that characterizes the image somehow (such as the distribution of colors in the image over some color space like RGB). Similarity can then be computed simply by comparing the histograms. One possibility is a "filter histogram", in which some number of filters—say, a number of texture filters, gauging the presence of some particular texture pattern in the image—are all performed on all images, and their responses are gathered in a sort of histogram, on which comparisons may be made. "Texture," in this context, is usually taken to mean a small repeated spatial pattern of some sort, say of small dots and bars at various orientations. For instance, a field of yellow and blue flowers may appear as a pattern of repeated yellow and blue dots, with some bits of green scattered between them. Filter responses can gauge the presence of particular kinds of patterns in an image. Some more sophisticated approaches may segment images somehow—say, using color or texture—and characterize these segmented regions by their color, texture, and shape. This sort of information is generally modeled statistically, and can be gathered into feature vectors which can then be used as the basis for retrieval methods [10]. Once images have been thus described, standard sorts of information retrieval methods may be applied.

The problem with much of this work is that it is characterizing the "content" of images at a very low level, one that may not be terribly useful in many applications. Although the work sometimes deals with spatial layouts of color or texture regions, it treats this layout as a sort of template. Furthermore, the layout itself is generally characterized only implicitly, as numerical interrelationships that happen to exist between various pieces of the data (say, the locations within the image that these shapes appear). Statistical models may offer some robustness, here, but cannot capture the sort of abstract, qualitative reasoning we're speaking of in this paper.

Levinson and Ellis [15, 16] have investigated the general problem of matching two semantic networks, though not specifically the task of retrieving 2-D line drawings by example. They describe a matching method in which the source networks are organized in a graph-subgraph hierarchy. While our retrieval process and selection methods are quite different from theirs, our notion of similarity as graph containment *is* similar to their proposals.

Do and Gross [4, 11] describe a method for retrieving 2-D line drawings in the domain of architectural design. Do and Gross' heuristic method is very simple: given two drawings, it compares the type and number of spatial elements and the spatial relations by counting. Their method is roughly equivalent to the heuristic step in our first method. One complication in their ideas is that the user can select the level of description; that is, what objects and relations are represented, and hence counted. Their system makes frequent concessions to user interaction like this, leaving many decisions open to the user rather than to some AI method. Certainly there is some merit to this idea.

Ounis and Paşca [18, 19] view the problem of associative image retrieval as one of computing projections over conceptual graphs representing their content. Although they do not describe it as constraint satisfaction, their algorithm, in fact, is doing constraint satisfaction to compute the projection. Our constraint satisfaction algorithm is inspired in part by their work, and represents yet another variation of the standard CSP techniques. Their system, in practice, is quite different from ours. Images must be annotated by hand with descriptions of their content, and the system itself was actually written on top of a DBMS. Their representation also had type hierarchies, both of objects and of relations, and indices involved "acceleration tables" that computed matches using these type hierarchies as guides. It's not entirely clear what the advantage of these type hierarchies is, but it is an interesting approach.

Börner et al. compute the maximal common subgraph of the query drawing with the stored drawings. Given the computational complexity of computing maximal common subgraph, we expect that this method can be successful only in specific domains in which the graphs representing the drawings are small and sparse.

We should point out that the idea of a two stage retrieval is not new. Indeed, two stage retrieval appears to be a prominent theme in much of information retrieval (as even a cursory web search reveals). In addition, MAC/FAC [9] describes a similar idea in the context of cognitive models of analogical reasoning. Of course, wee are not attempting to *model* any particular aspects of human reasoning.

Although we stated earlier that little or no work is out there on visual retrieval using spatial relationships, in fact there has been some work in information retrieval in this direction [2, 14, 17]. There has also been some work in what are sometimes called "spatial databases," such as in geographic information systems [5, 6, 20]. This research has focussed on relational query languages that can either be automatically extracted from some drawn query (as with Egenhofer's work on Spatial-Query-by-Sketch, a query drawing system for geographic information systems), or else entered textually somehow (say, using SQL). The work in these areas tends towards, on the one hand, expressive languages that make the retrieval problem intractable [20], or, on the other hand, much less expressive representations that effectively turn the queries into mere templates. Egenhofer's work, we should point out, deals with topological relations between spatial regions (such as overlap or containment) represented as semantic networks in very much the way we are talking about [6]. However, though it deals with query formation, this work does not deal with retrieval *per se*, which as we have pointed out is, in fact, a hard problem in and of itself, given these sorts of network representations.

Ferguson and Forbus [8] too address a different problem from our work, viz., detection of regularities in a 2-D line drawing. Their representation of a drawing is similar to ours: encoding a drawing in a graph of spatial elements and spatial relations among them. The system described—GeoRep—is essentially a rule engine based on an LTMS that can compute relational descriptions of images given some rule set defining an appropriate language. It is divided into two components; the first computes salient, low-level visual features such as proximity, parallelism, and connectedness of objects, and feeds this information to the rule engine. The rules are meant to capture high-level, domain-specific (and possibly task-specific) knowledge that can aid representation (for instance there is a rule set for circuit diagrams).

6 Future Work

Subgraph isomorphism, or projection, is a fairly restrictive notion of similarity. Certainly, given the relationship of the image to its description, some more interesting measures of similarity might be devised with more interesting domain languages, but the problem of similarity of description remains. One possible extension is to relax the problem into one of computing the maximal common subgraph—or, more precisely, to find the MCS of the target and some set of source images [1]. In addition to offering a more relaxed notion of similarity, this approach allows a quantitative measure of similarity: results could be ranked by the fraction of the target description that was matched.

Another area of future work is with more interesting domains. Our examples were rather simple; it seems that a more interesting domain would require the relational description to be more elaborate. In addition to offering a better test bed, this could offer the chance for some more interesting constraint satisfaction techniques to take advantage of structure inherent the problem—structure that was not present in our examples. For instance, forward checking or intelligent variable ordering are two examples of CSP techniques that take advantage of structure in the problem that are probably not very useful in our language, here, since the constraint graphs are complete graphs. We expect efforts in this direction to eventually break the methods described here; subgraph isomorphism and MCS are NP-Complete, in general, and at some point methods based on them will eventually explode (in terms of running time). The interesting question to us is regarding how far they can be pushed, and what interesting AI techniques can be used to push them.

One interesting potential application of this work has to do with case-based decision support in medical domains. Specifically, in the case of certain kinds of congenital heart defects, there is ongoing work on automatic construction of patient-specific 3-D models of the dynamics of the right ventricle of the heart [7]. The idea is to use these models as the basis for a case-based reasoning system for aiding the assessment and management of congenital heart disease. A system to do this would have to be capable of making some sort of analogical comparison between the RV models in different cases, determining those cases in memory which best resemble the given case.

On a somewhat different note, vision-based applications seem interesting to us, and it seems to us that there is a possibility of connecting our work with earlier CBIR work. In particular, the work that has gone on has focused on either feature-based or (somewhat less often) template-based representations. It would be interesting to explore what sorts of representations could be devised using more powerful languages based on relational properties. Given that some interesting representations can be devised, our work offers a method for retrieving images with similar descriptions. There may also be some interesting representational issues in the domain of sketch-based recognition, which may offer some good applications for this work (it should be pointed out that Do and Gross's work [4] is an example of previous work in this latter area).

In the longer term, we expect our work on use of spatial structure for image retrieval by example will support visual analogy in general [3], by operating as a retrieval mechanism for more abstract representations of visual scenes that describe problem solving instances. People seem to solve some problems by solving them visually (think of sketching on a white board), or else by seeing a visual resemblance to something else, and letting that inspire a solution. This is what we are referring to by "visual analogy". It should be pointed out that in this instance, the representations are now only visual in that they describe visual aspects of a scene (appearance, shape, color, and so on), and not because they describe any particular image or photograph.

One problem that our work has not yet begun to address, and in fact is largely unaddressed in the literature, is that of ambiguity. In section 4.1.1, when we discussed feature vectors we mentioned how drawings are sets of shapes, and the multiset of shape types in a drawing could be used for initial recall (reminding). If we have a drawing of two vertically adjacent squares (e.g., the number "8" on a digital clock), do we really have two adjacent squares, a rectangle with a line through the middle, three horizontal and two vertical lines, four vertical and three horizontal lines, or what? A source drawing in memory could be any of these. A drawing retrieval program should be able to deal smoothly with the multiple levels of description.

7 Conclusion

The problem of content-based image retrieval is quite open-ended with wide-ranging applications. Most earlier work has investigated computation and comparison of low-level content features of images such as object shapes and locations, and color and texture distributions. In contrast, the emphasis of our work is on the use of spatial structure for characterizing the contents of an image, evaluating similarity between two-images, and retrieving of images by example. To isolate this issue of spatial structure from other content features, our work focuses on associative retrieval of 2-D line drawings.

This paper describes several methods, based on resolution and unification on the one hand, and constraint satisfaction on the other, for retrieving 2-D line drawings similar to a query example. These methods operate on a relational description that explicitly represents the spatial structure of the image, treating the relations as constraints and the visual elements as the variables to be assigned values from memory.

We have also reported on several sets of experiments with the above retrieval process and matching methods. These experiments lead to three specific conclusions:

1. The constraint satisfaction methods provide substantial computational benefits over the resolution-based methods. Resolution and unification worked only for the first test set, characterized by very small and simple problems, and did not scale up to the larger and more complex problems in the second and third test sets.
2. The backtracking constraint satisfaction method offers similarly substantial benefits over constraint satisfaction by generate and test. The latter did not scale up to the larger and more complex problems in the third test set.
3. The backtracking constraint satisfaction method with discrimination trees was the fastest of the methods considered, and was the only one that finished all of the tests in a reasonable amount of time.

While the above results were what one might expect given previous literature on constraint satisfaction problems, our work also led to a surprising result: for the fastest of the methods considered, two stage retrieval offered no substantial improvement in performance over the one stage version. This clearly runs against the conventional wisdom. Our work does not settle the issue by far, and so unraveling this issue will require more work.

References

[1] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3–4):255–259, March 1998.

[2] S. K. Chang, Q. Y. Shi, and C. W. Yan. Iconic indexing by 2-D strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[3] Jim Davies and Ashok K. Goel. Visual analogy in problem solving. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 377–382, Seattle, WA, August 2001. Morgan Kaufmann Publishers.

[4] Ellen Yi-Luen Do and Mark D. Gross. Drawing analogies: Finding visual references by sketching. In *Proceedings of the Association of Computer Aided Design in Architecture*, Seattle, WA, 1995.

[5] Max J. Egenhofer. Spatial-Query-by-Sketch. In W. Burnett and W. Citrin, editors, *IEEE Symposium on Visual Languages*, pages 60–67, Boulder, CO, 1996. IEEE Press.

[6] Max J. Egenhofer. Query processing in Spatial-Query-by-Sketch. *Journal of Visual Languages and Computing*, 8(4):403–424, 1997.

[7] David H. Fann. *Dynamic Modeling of the Right Ventricle for the Analysis of Congenital Heart Defects*. PhD thesis, Georgia Institute of Technology, 2002. In preparation.

[8] Ronald W. Ferguson and Kenneth D. Forbus. GeoRep: A flexible tool for spatial representation of line drawings. In *Proceedings of the 18th National Conference on Artificial Intelligence*, Austin, Texas, 2000. AAAI Press.

[9] Kenneth D. Forbus, Dedre Gentner, and Keith Law. MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, 19(2):141–205, 1995.

[10] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*, chapter 25, pages 599–619. Prentice Hall, New Jersey, 2003.

[11] Mark D. Gross and Ellen Yi-Luen Do. Diagram query and image retrieval in design. In *Proceedings of the 2nd International Conference on Image Processing*, Crystal City, VA, 1995. IEEE Computer Society Press.

[12] Amarnath Gupta and Ramesh Jain. Visual information retrieval. *Communications of the ACM*, 40(5):69–79, May 1997.

[13] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, Spring 1992.

[14] S. Y. Lee, M. K. Shan, and W. P. Yang. Similarity retrieval of iconic image database. *Pattern Recognition*, 22(6):675–682, 1989.

[15] Robert Levinson. Pattern associativity and the retrieval of semantic networks. *Journal of Computers and Mathematics with Applications*, 23:573–600, 1992. Also appears in Technical Report UCSC-CRL-91-14, University of California at Santa Cruz, Computer Research Laboratory, Santa Cruz, CA 95064.

[16] Robert Levinson and Gerard Ellis. Multi-level hierarchical retrieval. *Knowledge-Based Systems*, 5(3):233–244, 1992.

[17] J. Z. Li, M. T. Özsu, and D. Szafron. Spatial reasoning rules in multimedia management systems. In *Proceedings of the International Conference on Multimedia Modeling*, pages 119–133. World Scientific Press, November 1996.

[18] Iadh Ounis. Organizing conceptual graphs for fast knowledge retrieval. In *Proceedings of the 10th International Conference on Tools with Artificial Intelligence*, pages 120–129, Taipei, Taiwan, 1998. IEEE Computer Press.

[19] Iadh Ounis and Marius Paşca. RELIEF: Combining expressiveness and rapidity into a single system. In *Proceedings of the 21st Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 266–274, Melbourne, Australia, 1998. ACM Press.

[20] C. H. Papadimitriou, D. Suciu, and D. Vianu. Topological queries in spatial databases. *Journal of Computer and System Sciences*, 58(1):29–53, 1999.

[21] Yong Rui, Thomas S. Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions and open issues. *Journal of Visual Communications and Image Representation*, 10(4):39–62, April 1999.

[22] Simone Santini and Ramesh Jain. Similarity measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):871–883, September 1999.

[23] Remco C. Veltkamp and Mirela Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Institute of Information and Computing Sciences, Utrecht University, October 2000.

[24] Patrick W. Yaner and Ashok K. Goel. Retrieving 2-D line drawings by example. In *Diagrammatic Representation and Inference: Proceedings of the 2nd International Conference, Diagrams 2002*, volume 2317 of *Lecture Notes in Artificial Intelligence*, pages 97–99. Springer-Verlag, 2002.