Visual Analogy: Reexamining Analogy as a Constraint Satisfaction Problem[1]

Patrick W. Yaner (yaner@cc.gatech.edu)
Ashok K. Goel (goel@cc.gatech.edu)
Artificial Intelligence Laboratory
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

Holyoak and Thagard proposed that the retrieval and mapping tasks of analogy can be viewed as constraint satisfaction problems, and described a connectionist implementation of their proposal. In this paper, we describe another constraint satisfaction method for the two tasks in the context of visual analogy: in our method, the source cases are organized in a discrimination tree, and all the source cases are searched *at once*. We also present an evaluation of the method for retrieval and mapping of 2-D line drawings from an external memory. The evaluation is based on structural constraints, and uses subgraph isomorphism as the similarity measure. One result is that a decomposition of the retrieval task into feature-based reminding and structure-based selection appears to provide little computational benefit over just selection.

Introduction

Holyoak and Thagard proposed that the retrieval (Thagard, Holyoak, Nelson, & Gochfeld, 1990) and mapping (Holyoak & Thagard, 1989) tasks of analogy can be productively viewed as constraint satisfaction problems. Their proposal incorporated structural, semantic and pragmatic constraints and used graph isomorphism as the primary similarity measure. Their mapping system, called ACME, and the complementary retrieval system, named ARCS, provided connectionist implementations of their proposal. In ACME, nodes are constructed for each map hypothesis (between a source element and a target element), with inhibitory and excitatory links between different nodes, and the network is run until it reaches quiescence. The work described here builds on Holyoak and Thagard's proposal but seeks a different solution to the retrieval and mapping tasks. While we also view the retrieval and mapping tasks as constraint satisfaction problems (CSPs), our method for addressing the tasks (i) organizes the source cases in a discrimination tree, (ii) uses (general-purpose) heuristics to guide the search, (iii) performs a backtracking search, and (iv) searches all the source cases *at once*.

The goal of our current work is to develop a computational theory of *visual* analogy. Analogies transfer relational knowledge from a source (or base) case to a target problem. Depending on the nature of the target and the source, the knowledge transferred in an analogy may pertain to different kinds of relations, for example, causal, functional or teleological relations. In visual analogy, the pertinent relations are spatial relations among visual elements. In a different part of the project, we have developed a technique for transfer of spatial knowledge, given a target problem and a source case and given a mapping between the two (Davies & Goel, 2001, 2003). In the part described in this paper, we focus on the retrieval and mapping tasks.

Our methodology is to start with simple problems and incrementally add complexity to them. This incremental nature of the methodology is manifested in three ways: firstly, visual knowledge can be of many forms, such as depictive bit-mapped representations, sketches, or animations, but our work deals specifically with diagrammatic knowledge represented symbolically as discrete geometric elements and the spatial relations between them; secondly, though visual analogies, like analogies more generally (as proposed by Holyoak and Thagard), can involve semantic and pragmatic constraints, we start with just the structural constraints imposed by requiring source and target to match structures; and thirdly, from a graph theoretic perspective, there may be more than one sort of graph isomorphism measure that may be the ideal measure, such as maximal common subgraph, but we begin with subgraph isomorphism as our metric.

The retrieval task, in this work, assumes a computer-based library of 2D line drawings, takes as input a query (target) in the form of a drawing (and no other information), and gives as output the source drawings that are most similar to the target. The mapping task takes as input a target problem and a source case, and gives as output correspondences between the basic elements of the source case and the target problem.

Retrieval

Following earlier work on analogical retrieval—e.g., MAC/FAC (Forbus, Gentner, & Law, 1995)—our retrieval architecture supports a two-stage process for diagram retrieval: reminding (or initial recall), and selection. The architecture consists of (up to) six basic components: an initial stage generating feature vectors, a process that generates a semantic network describing the contents (spatial structure in this case) of an drawing, a process that matches a target's description (semantic network) to source descriptions from memory, a working memory with potential sources to match with the target, and finally, an interface to the rest of the analogy system in which this retrieval would be taking place.

The reminding task takes as input a target example and returns as output references to stored drawings whose feature vectors match that of the target. The stored drawings are indexed by feature vectors describing their spatial elements; the feature vector for the target is constructed dynamically. References to those drawings with sufficiently similar feature

vectors (according to some appropriate criteria, as explained below) are brought into the working memory. In the selection stage, the semantic networks of the drawings in working memory are matched with that of the target example. Drawings whose descriptions match the target description sufficiently well are collected and returned.

While the reminding 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 target drawing, the selection task uses the spatial structure of line drawings—i.e. the qualitative arrangement of the various shapes in them—to actually match the target to the source drawings.

Visual cases are represented in three distinct ways: the the drawings themselves, the feature vectors, and the network of spatial relations. The representation of the drawings themselves is simply object-based: a list of each visual element, such as lines, triangles, etc., and their specific geometric properties (location, and so on). The feature vector is a multiset of the object and relation types contained in a semantic network. A multiset is a set that can contain more than one of each element (e.g. $\{2 \cdot A, 3 \cdot B, \ldots\}$). Given a semantic network describing an drawing, a feature vector in our system would look something like this: $\{3 \cdot \texttt{rectangle}, 2 \cdot \texttt{circle}, 3 \cdot \texttt{leftOf}, 1 \cdot \texttt{contains}, \ldots\}$.

A drawing is recalled, in the first stage, if the multiset of shape and relation types contained in it is a superset of that of the target. The method scans all stored drawings, calculating whether or not the multiset of objects and relations in the target is a subset of the multiset of objects in each source drawing, and returning those for which this is the case. That is, if Q is the feature vector for the target, and $S_1, S_2, \ldots, S_k$ are the feature vectors of the drawings currently in memory, then the method returns those drawings for which $Q \subseteq S_i$.

Figure 1 illustrates a simple 2D line drawing and its representation in terms of spatial relations in our system. The system at present recognizes four types of spatial elements: individual lines, triangles, rectangles, and ellipses (circles and squares are special cases of ellipses and rectangles, and are not treated as being of a separate type). Also, it presently recognizes five types of relations among the elements: `left-of`, `right-of`, `above`, `below`, and `contains`. The automatic generation of a semantic network for a target drawing 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. As an example, the semantic network in Figure 1 would represent the drawing shown above it.

Memory Organization

When a source drawing is added to memory, several things happen. First, its description is generated, the network of relations describing the spatial layout of the drawing, as well as its feature vector. Second, once this network is generated, each "term" in the network, by which we mean a link (relation) together with its incident nodes (elements), is added to a discrimination tree. This allows the selection method to match individual terms in the target with all terms of the same form that appear across *all* source drawings in memory, thus
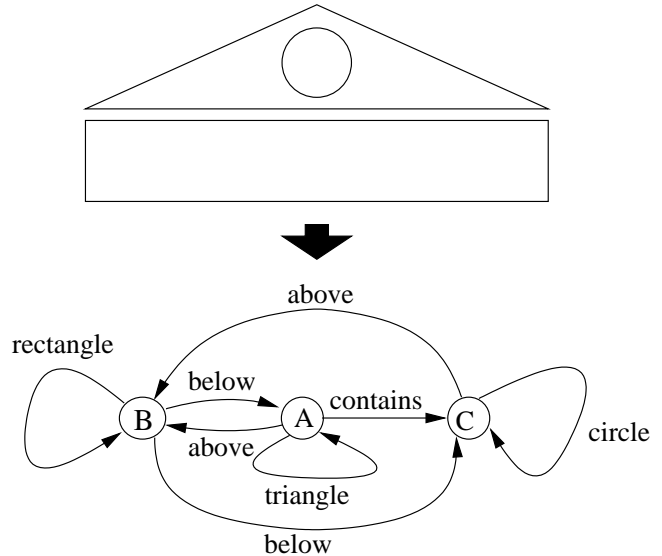

Figure 1: An example of a three-node semantic network in our language. Each pair of objects is tested, and links added for each relation that holds.

allowing all of the descriptions of all of the drawings to be searched at once.

The selection method, described below, builds a set of potential assignments for each target element, and in evaluating these, it looks to see what terms each source element is involved in, and this involves the index into memory by individual terms. The overall scheme is to build a representation of all possible mappings, and reduce this list by screening out the ones that don't work. Ones that don't work are screened out because they do not satisfy the constraints imposed by the problem. This is constraint satisfaction, and this is what it means to solve the problem by constraint satisfaction.

Constraint Satisfaction

The core of the system is the selection process. The process finds a correspondence between the target drawing and the source drawings in working memory, eliminating drawings for which no correspondence can be found.

The selection problem is essentially one of matching objects (variables and constants) in the target and the source under the constraints imposed by the terms in which they appear. The target has a set of variables (its objects, the nodes in the semantic net) to be matched to some constants (i.e. values) from the sources and the relationships between these variables impose constraints on the values to which they can be matched. This is constraint satisfaction. This algorithm works by maintaining an index of all the terms across all of the source descriptions. It recalls individual terms from memory and puts them together to form the complete matching. When a source drawing is stored in memory, its description is generated and indexed in this way, by each term that appears in it. There is a separate table for each type of term, i.e. one for `left-of`, one for `above`, etc.

Treating the target elements as variables to be assigned values, the potential values are the nodes from the source de-

scriptions in memory, all of which are considered at once. That is, the method is not performing a separate test on each source in memory, but, rather, it is 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. If these constraints are met, then A can be matched with X and B can me matched with Y. Here the constraints are all either unary (say, A is a circle—a type constraint), or binary (say, A is left of B—a relational constraint). The only exception is the constraint that all values be from the same description, but this can be inferred from the binary constraints.

This matching process works in three phases: initialization of domains, reduction of domains, and finding the matching, where matching means subgraph isomorphism. The first phase initializes the target domains to sets of values that have the same incoming and outgoing edges. The second phase reduces these domains by eliminating values that are not all in the same drawing. These two phases reduce the selection of values for each variable. The third phase actually computes the isomorphism using constraint satisfaction and backtracking.

The first phase (initialize domains) works by finding nodes in memory that "look similar" to the target nodes: if a target 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) 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.

These two stages are the "real" first stage of the algorithm, and our results, described in (Yaner & Goel, 2003), showed that the feature-vector-based first stage was really quite redundant, and offered little improvement. Viewed as such, this first stage applies two heuristics to the sources from memory: (i) prune any individual element (as opposed to entire drawings) that don't have the same "signature" (as just described) as the corresponding target element, and (ii) prune any terms whose associated drawings are not represented in every target element's domain. The latter one enforces subgraph isomorphism. It is important to note that these are both logically implied by the similarity metric that the last phase, described below, implements. It would be an interesting experiment to look at other heuristics that prune out mappings that might have otherwise been returned by the last phase.

The last phase (find matchings) is the one that actually does the work. The basic procedure is one that generates matchings, checking them for consistency as it goes, and backtracking when necessary. 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 mapping from target to source. This algorithm returns all valid mappings. The idea is that the first two phases have restricted the set of possible mappings so that there aren't nearly as many, now, as there would have been if a pure depth-first search had been done.

In general, the time complexity of depth-first search, such as this is, 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. In this case the depth is the number of elements in the query, and the branching factor is the number of elements across all sources in memory. 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. Note that this is a backtracking search, however, so large portions of the state space are cut off at each step. With 42 test images in memory, the number of objects in a drawing ranging from 3 to over 50 (the average was about 12), the number of terms in the description ranged from a couple of dozen to over eight thousand. There were 21 queries with this test set, with two to five spatial elements in each, and up to several dozen terms. With this test data, the system was retrieving drawings in about 9.32 seconds on average (across all 21 drawings), doing an average of about 1.49 million memory accesses (to the index of terms across all the drawings) per retrieval.

Galatea

Since the system does retrieval essentially by producing all possible mappings that it is capable of finding, we adapted a version of the system to the mapping process for use in a system called *Proteus*, a visual analogical reasoning system. The transfer stage of *Proteus*—implemented in a system called *Galatea*—is described in Davies and Goel (2001, 2003).

Galatea solves problems represented in a high-level visual language called Covlan (Cognitive Visual Language). The system solves these problems by analogy to existing problems whose solutions are mapped out as a sequence of transformations on the knowledge states that are represented in this language. *Galatea* solves the problem by taking a mapping between the initial knowledge states of the source and target and mapping the transformations and generating the intermediate knowledge states (and mappings between them), and thereby constructing the rest of the transformations and knowledge states leading to the solution to the target problem. The mapping system, then, needs to connect the initial knowledge state of the source and the target drawing. From the perspective of retrieval and mapping, the relevant issues pertaining to *Galatea* are: (1) what is that knowledge representation, and (2) what are the nature is of the required mappings?

Covlan consists of knowledge states, primitive elements, primitive relations, primitive transformations, general visual concepts, and correspondence and transform representations. In Covlan, all knowledge is represented as propositions. In this paper we will only be concerned with the primitive elements and the primitive visual relations. The primitive elements are `polygon`, `rectangle`, `triangle`, `ellipse`, `circle`, `arrow`, `line`, `point`, `curve`, and `text`. There is also a `set` element type, with members that have `in-set` relations back to the set they are members of, though these do not correspond to visible entities—this is purely for group-

ing purposes. Each element is represented as a frame with attribute slots such as `location`, `size`, `orientation`, and `thickness`, but these attributes will not concern us, since mappings between attribute values are not part of the required mappings, and thus representing them in the semantic network is not necessary.

Primitive visual relations represented are touching, `above-below`, `right-of-left-of`, `in-front-of-behind`, and `off-s-image`. A typical knowledge state is represented with a node corresponding to that knowledge state (e.g. `L14-simage1`), and elements (which may be sets) are represented with `contains-object` relations from the knowledge state element to the visual elements themselves.

We describe next some example problems originally designed for *Galatea*. The first example problem is a fairly simple one: dividing a pizza into some number of slices based on analogy to the problem of dividing up a cake into some number of pieces. In this case, there is a cake (or pizza), and a set of people in the initial problem state. Set members are not mapped, and the division is made in transformations in later problem states, so the only possible mappings are cake to pizza and set of people to set of people, or cake to set of people and set of people to pizza. The problem, as represented in *Galatea*, does not contain any visual relations between the set of people and the cake (or pizza), and thus there is nothing constraining the mapping to be the "correct" mapping. The latter mapping will probably lead to failure in the transfer stage, but both are returned by our system.

A more complex and interesting example is based on Gick and Holyoak's fortress/tumor problem (1980). In this problem, we have an army attacking a fortress over mined roads, and the general decides to split his army to avoid setting off the mines, and a target case in which there is a patient with a tumor and a doctor who wants to kill the tumor with radiation. The supposed analogy is to split the beam (somehow) to avoid killing the healthy tissue that is in the way. The visual representation of these problems has a fortress (and a tumor represented similarly) and four roads (sections of the body surrounding the tumor), and an army represented by an arrow (a ray of radiation represented similarly). The "correct" analogy maps the set of roads to the set of body parts, the fortress to the tumor, and the army to the ray. However, there being three of each thing to match, and the particular representation chosen not using the visual relations (though it could have), there was nothing constraining the mapping, and all six possible correspondences were returned. Had visual relations constrained it, the number of possible mappings would have been smaller.

Mapping

Galatea has set up the requirements for the mapping task such that only visual elements are to be mapped, not attribute values, and so attribute values (which can be represented as propositions, and hence can be represented in a semantic network) are not included in the input to the mapping system. In addition, members of sets are (generally) not to be mapped, and so any visual element on the left-hand side of an in-set relation can be pruned from the mapping system's input, as well. With these two constraints, the mapping system was run on several sample problems, two of which were described

---

function GENERATEMAPPINGS

1: *sourceRels* ← first simage from source problem
2: *targetRels* ← target problem simage
3: *sRelLables* ← names of all relations represented in *sourceRels*
4: *tRels* ← remove from *targetRels* all relations that don't match one in *sRelLabels* and all relations involving a literal (i.e. attribute-value pairs)
5: *tRelLabels* ← names of all relations represented in *tRels*
6: *sRels* ← remove from *sourceRels* all relations that don't match one in *tRelLabels* and all relations involving a literal (i.e. attribute-value pairs)
7: *sNodes* ← list of all nodes (elements) from *sRels*
8: *tNodes* ← list of all nodes (elements) from *tRels*
9: *domains* ← GENERATEDOMAINS(*sNodes*, LENGTH(*tNodes*))
10: *rDomains* ← GENERATEDOMAINS(*tNodes*, LENGTH(*sNodes*))
11: *fMappings* ← FINDPROJECTIONS(*sRels*, *tNodes*, *tRels*, *domains*)
12: *rMappings* ← FINDPROJECTIONS(*tRels*, *sNodes*, *sRels*, *rDomains*)
13: *rMappings* ← reverse each of the mappings returned in *rMappings* so that they map source onto target properly instead of target onto source
14: **return** *fMappings*$\bigcup$*rMappings*

Algorithm 1: Generate Mappings

---

above. Four other problems of similar nature and size were also run on this system.

The mapping algorithm (see Algorithm 1) works as follows: the outer procedure (generate mappings) first retrieves the named source and target representations from memory, then applies the above heuristics to it, and finally generates the mappings and returns them. Since it computes subgraph isomorphism, as above, we run it both ways—attempting to map source onto target, and also attempting to map target onto source and reversing the returned mappings. Thus it is possible to find the target within the source or vice versa, finding the source within the target. The algorithm for FINDPROJECTIONS is identical with that of the third phase, "find matchings", above.

It's important to note that this does not actually solve the mapping problem; it particular, it returns *all* mappings, so that an additional search or evaluation stage is necessary to find the relevent ones. This is where pragmatic and semantic constraints may start to enter back into the picture. Our work to date has begun with only structural constraints as an experiment, and we plan to reintroduce other constraints as the larger problem context is reintroduced.

At any rate, the cake/pizza example described above, when run through this system, came up with two mappings: one that maps the cake to the pizza and the set of people to the set of people, and one that maps the cake to the set of people and the other set of people to the pizza:

Cake *maps-to* Pizza

Set12 *maps-to* Set14

Cake *maps-to* Set14
Set12 *maps-to* Pizza

Set12 is the set of people in first cake problem knowledge state, and Set14 is the set of people in the first pizza problem knowledge state. *Proteus*, recall, does not map members of sets, and so the individual people are not mapped onto each other, only the sets. The first one, obviously, is the "correct" one, the one that would lead to a successful transfer and evaluation of the problem solution.

The fortress/tumor problem was more interesting. The heuristics pruned out the set of roads and body parts, as well as the shapes and sizes and positions of all the elements, and so the only details left to influence the mappings were the fact that the elements were part of the problem. There were three elements, thus, remaining, for each one: Fortress and Tumor, Soldier-Path and Ray, and Set1 (the set of roads) and Set2 (the set of body parts surrounding the tumor), and six mappings produced:

Fortress *maps-to* Tumor
Soldier-Path *maps-to* Ray
Set1 *maps-to* Set2

Fortress *maps-to* Tumor
Soldier-Path *maps-to* Set2
Set1 *maps-to* Ray

Fortress *maps-to* Ray
Soldier-Path *maps-to* Tumor
Set1 *maps-to* Set2

Fortress *maps-to* Ray
Soldier-Path *maps-to* Set2
Set1 *maps-to* Tumor

Fortress *maps-to* Set2
Soldier-Path *maps-to* Tumor
Set1 *maps-to* Ray

Fortress *maps-to* Set2
Soldier-Path *maps-to* Ray
Set1 *maps-to* Tumor

Now, this really represents all correspondences between three things and three things. The primary reason for this is that the representation chosen for this particular problem does not involve any reference-frame relations such as `left-of` or `right-of`. If it had, these relations would constrain the mappings.

Discussion

In the introduction, we mentioned Holyoak and Thagard's ACME system (1989) and noted the similarities and differences between our work and theirs. ANALOGY (Evans, 1968) was an even earlier AI program that performed the task of finding similarities and differences between visual cases. It performed simple geometric analogies of the kind that appear on many intelligence tests. Let us suppose that each of A, B, C, D, E and F is an arrangement of simple geometric objects, e.g., a small triangle inside a large triangle, a small circle inside a larger circle, etc. Given an analogy A:B, and given C and multiple choices D, E and F, ANALOGY found which of D, E, and F had a relationship with C analogous to that between A and B. It represented the objects and the spatial relationships between them in the form of semantic networks, which enabled it to compare the spatial structure of the various arrangements. However, since ANALOGY performed an exhaustive and linear search of the mappings, its method cannot scale up to any realistic problem.

While ANALOGY was an early program that matched symbolic descriptions of two drawings and found similarities and differences between the drawings, MAGI (Ferguson, 2000) and JUXTA (Ferguson & Forbus, 1998) are two recent systems that find mappings between symbolic representations of two drawings (or two portions of the same drawing). These systems use truth maintenance as the mechanism for keeping track of new constraints and retracting old conclusions.

Our decomposition of the retrieval task into feature-based reminding and structure-based selection is similar to that of MAC/FAC (Forbus et al., 1995). The similarity is specially striking because in its current stage ours deals only with structural constraints; as noted in the introduction, we plan to explore and exploit semantic and pragmatic constraints in the next stage. However, in contrast to MAC/FAC, the experiments described in (Yaner & Goel, 2003) indicate that the two-stage decomposition of the retrieval task provides little computational benefit over just one-stage retrieval based on structure-based selection.

In computer-aided design, FABEL (Gebhardt, Voß, Gräther, & Schmidt-Belz, 1997) was an early project to explore the automated reuse of diagrammatic cases. In particular, TOPO (Börner, Eberhard, Tammer, & Coulon, 1996), a subsystem of FABEL, used the maximum common subgraph (MCS) of the target drawing with the stored drawings for retrieve similar drawings. Gross and Do (1995) describe a method for retrieving designs that contain a given design pattern in the domain of architectural design. Gross and Do's 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 first stage in the two-stage retrieval process.

In computer vision, Grimson and Huttenlocher (1991) developed a similar method for object recognition. They begin with a model with a set of features, such as a set of potential edges in some arrangement, and sensor data with a set of sensor features (edges, vertices, etc.); a lot of sensor features might be noise. The task is to find a set of sensor features that comes from one (and the same) object. Their method matches model features to sensor features under some transformations within specific limit of tolerance. The model imposes constraints, for instance, by its arrangement of features. Although they do not describe it as constraint satisfaction, their method in fact is in assigning values to variables under unary and binary constraints imposed by the arrangement by using a backtracking depth-first search.

Constraint satisfaction methods have become common in AI: Prosser (1993) describes methods of constraint satisfaction with backtracking; and Bayardo and Schrag (1997) provide evidence of applicability of constraint satisfaction with

backtracking for real-world intractable problems in planning and scheduling. Our method of constraint satisfaction with backtracking, with the case memory organized into discrimination trees, builds on the work of Ounis and Paşca (1998). They view the general problem of associative image retrieval as one of computing projections over conceptual graphs representing their content. Although they do not describe it as a constraint satisfaction method, their algorithm, in fact, is doing constraint satisfaction to compute the projection. However, their method is limited to constraint satisfaction with generate and test with no backtracking.

Conclusions

We have described a constraint satisfacton method for the retrieval and mapping tasks of analogy. Our method (i) organizes the source cases in a discrimination tree, (ii) uses (general-purpose) heurstics to guide the search, (iii) backtracks (if and when needed), and (iv) searches all the source cases at once. We also presented an evaluation of the method for the retrieval and mapping of diagrams from an external memory.

Our laboratory-scale experiments, with drawings containing only up to fifty spatial elements and their representations containing only up to eight thousand terms, indicate that the method of constraint satisfaction is fast and appears quite promising for use in practice. On the one hand, we fully expect that the complexity of the task will significantly worsen for larger drawings and larger libraries of drawings, but, on the other, we also expect that it should be possible to develop significantly faster methods for the task. For example, we expect that use of spatial aggregations and abstractions to organize the representation of the spatial structure of a drawing in the form of a linked hierarchy of semantic networks would partition the search space performance especially for large, complex drawings (e.g. Papadias, Kalnis, & Mamoulis, 1999). In addition, more sophisticated constraint satisfaction techniques such as forward checking and intelligent variable ordering, to name just a couple of common ones, can be brought to bear on the problem as well, taking advantage of structure in the knowledge representation and the search space.

References

Bayardo, R. J., Jr., & Schrag, R. (1997). Using CSP lookback techniques to solve real-world SAT instances. In *Proc. AAAI-97* (pp. 203–208). Providence, Rhode Island: AAAI Press.

Börner, K., Eberhard, P., Tammer, E.-C., & Coulon, C.-H. (1996). Structural similarity and adaptation. In I. Smith & B. Faltings (Eds.), *Advances in Cased-Based Reasoning: Proc. 3rd European Workshop on Cased-Based Reasoning* (Vol. 1168, pp. 58–75). Lausanne, Switzerland: Springer-Verlag.

Davies, J., & Goel, A. K. (2001). Visual analogy in problem solving. In *Proc. IJCAI-01* (pp. 377–382). Seattle, WA: Morgan Kaufmann Publishers.

Davies, J., & Goel, A. K. (2003). Representation issues in visual analogy. In *Proc. 25th Annual Conf. Cognitive Science Society*. Boston, MA: Lawrence Erlbaum Associates.

Evans, T. G. (1968). A heuristic program to solve geometric analogy problems. In M. Minsky (Ed.), *Semantic Information Processing*. Cambridge, MA: MIT Press.

Ferguson, R. W. (2000). Modeling orientation effects in symmetry detection: The role of visual structure. In L. R. Gleitman & A. K. Josh (Eds.), *Proc. 22nd Annual Conf. Cognitive Science Society*. Philadelphia, PA: Lawrence Erlbaum Associates.

Ferguson, R. W., & Forbus, K. D. (1998). Telling juxtapositions: Using repetition and alignable difference in diagram understanding. In K. Holyoak, D. Gentner, & B. Kokinov (Eds.), *Advances in Analogy Research* (pp. 109–117). Sofia, Bulgaria: New Bulgarian University.

Forbus, K. D., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, *19*(2), 141–205.

Gebhardt, F., Voß, A., Gräther, W., & Schmidt-Belz, B. (1997). *Reasoning with complex cases* (Vol. 393). Boston: Kluwer Academic Publishers.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, *12*, 306–355.

Grimson, W. E. L., & Huttenlocher, D. P. (1991). On the verification of hypothesized matches in model-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(12), 1201–1213.

Gross, M. D., & Do, E. Y.-L. (1995). Diagram query and image retrieval in design. In *Proc. 2nd Int'l Conf. on Image Processing*. Crystal City, VA: IEEE Computer Society Press.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science*, *13*(3), 295–355.

Ounis, I., & Paşca, M. (1998). RELIEF: Combining expressiveness and rapidity into a single system. In *Proc. 21st Annual ACM SIGIR Conference* (p. 266-274). Melbourne, Australia: ACM Press.

Papadias, D., Kalnis, P., & Mamoulis, N. (1999). Hierarchical constraint satisfaction in spatial databases. In *Proc. aaai-99*. Orlando, FL: AAAI Press.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, *9*(3), 268-299.

Thagard, P., Holyoak, K. J., Nelson, G., & Gochfeld, D. (1990). Analog retrieval by constrain satisfaction. *Artificial Intelligence*, *46*, 259–310.

Yaner, P. W., & Goel, A. K. (2003). Visual case-based reasoning I: Memory and retrieval. In *Proc. IICAI-03*. Hyperbad, India: Springer-Verlag.