Transfer in Visual Case-Based Problem Solving

Jim Davies[1], Ashok K. Goel[2], and Nancy J. Nersessian[2]

[1] School of Computing, Queen's University, Kingston, Ontario, K7L 3N6, Canada
jim@jimdavies.org
[2] College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332, USA
{nancyn, goel}@cc.gatech.edu

Abstract. We present a computational model of case-based visual problem solving. The Galatea model and the two experimental participants modeled in it show that 1) visual knowledge is sufficient for transfer of some problem-solving procedures, 2) visual knowledge facilitates transfer even when non-visual knowledge might be available, and 3) the successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target cases. We describe Galatea, the two models created with it, and related work.

1 Introduction

Experimental evidence shows that visual knowledge often plays a role in case-based reasoning [2,7,11]. Why might this be? What functions do the visual representations serve in retrieval, adaptation, evaluation and storage of cases? These questions are very broad because they pertain to a variety of cognitive phenomena ranging from visual perception to external memory to mental imagery. In order to explore these issues deeply, in the following discussion we focus exclusively on case-based problem solving. Problem solving involves generating a procedure which may contain a number of steps. We will call procedures with the following two properties "strongly-ordered procedures:" 1) two or more steps are involved, and 2) some steps cannot be executed before some other steps have already been executed. Case-based problem solving is taking a solution from a source case and applying that solution or a modification of it to a target case.

Many past case-based systems in problem-solving domains have used visual knowledge and have supported visual reasoning (e.g., ARCHIE [13]. AskJef, [1]). However, these systems typically contain multi-modal cases, i.e., cases that contain both visual (e.g., photographs, drawings, diagrams, animations and videos) and non-visual knowledge (e.g., goals, constraints, plans and lessons). As a result, the precise role of visual knowledge in case-based problem solving remains unclear. In contrast, the present work deals with cases that contain only visual knowledge. Further, past case-based systems such as ARCHIE and AskJef leave the adaptation task to the user and do not automate the transfer of diagrammatic knowledge from a source case to a target problem. The present work directly addresses the transfer task in case-based problem solving.

H. Muñoz-Avila and F. Ricci (Eds.): ICCBR 2005, LNAI 3620, pp. 163 – 176, 2005.

Some domains are replete with visual information (e.g. libraries of CAD files, photograph databases), but others that need not explicitly contain visual information can be visually represented all the same. For example, effectively connecting a battery to wires might be represented, among other ways, functionally (the battery needs to be physically in contact with the wire so it can conduct electricity) or visually (the image of the end of the wire is adjacent to the image of the battery). Even though other kinds of knowledge and representations of these domains might be used to reason, human beings often claim to experience visual imagery when reasoning about them. The first hypothesis of this work is that visual knowledge alone is sufficient for automatic transfer of problem-solving procedures in some domains. The second hypothesis is that visual knowledge facilitates transfer even when non-visual knowledge might be available. One important implication of this hypothesis is that cases that when represented non-visually are semantically distant, could be represented in visually similar ways, thus facilitating transfer.

In this paper we describe the Galatea computational model, which, given a source problem-solving case and a target problem case, both represented visually, can solve the problem by transferring the solution from the source to the target case. We present Galatea and models of two human experimental participants implemented with it. The data we modeled comes from a cross-domain case-based problem-solving experiment [3]. Here we focus on two participants, L14 and L22. The source case (see Figure 1) is about a laboratory that needs to keep contaminated air from entering from the outside through its single door. The solution is to put in an airlock or vestibule, so that air is less likely to blow through both doors at once.

The unsolved target case describes a weed trimmer at the end of an arm that extends from the side of a truck. It clips the grass and weeds along the side of the road. The problem is that street signs get in the way. The task is to make the arm so that it can pass through the street signs. The transferred solution is to make an arm with a vestibule: While one door lets the sign into the vestibule, the other supports the arm. Then the first door closes, supporting the arm, and the second opens to release the sign on the other side. L14 was one of the participants who successfully solved this problem. The marks L14 made on his or her paper can be seen in Figure 2.

In the following section we will describe Galatea, using our model of L14 as a running example.

2 Galatea

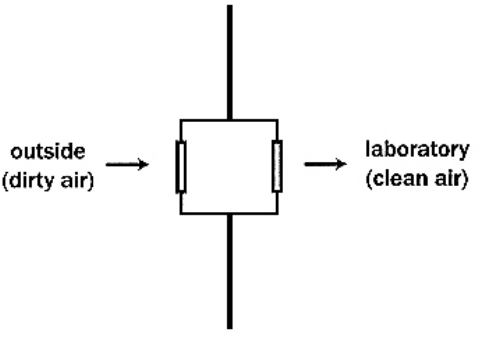
The modeling architecture used to model L14 is an implemented LISP computer program called Galatea. The issue is how a case-based problem solver might represent its diagrammatic knowledge of the source case and target problem, and how might it transfer the relevant problem-solving steps from the source to the target?

Galatea represents a source case as a series of knowledge states starting from the initial knowledge state and ending in the final or goal knowledge state. A knowledge state is represented diagrammatically in the form of shapes, their locations, sizes, and motions (if any), and the spatial relationships among the shapes.

Please read the two problems below. At the bottom of the page, please try to solve **Problem 2.** *Draw a diagram to show what you're thinking. The solution to* **Problem 1** *may be helpful in solving* **Problem 2.**

Problem 1: A computer chip manufacturer has designed a special lab for manufacturing microscopic devices. They have taken great care to seal off the lab from the surrounding environment in order to keep the air inside the lab free of dust and undesirable gases. The problem, though, is that whenever lab workers enter or leave the room, the seal is broken and contaminated air is allowed in. The company is trying to design a door that will allow workers to enter and leave the lab easily, while minimizing the amount of contaminated air that is let in.

Solution: Have workers enter a vestibule space before entering the lab.



Problem 2: In order to trim the weeds that grow along the side of the road, the Department of Transportation has designed a weed trimmer that attaches to the end of a long pole sticking off the side of a truck. As the truck drives down the highway, the trimmer is extended about 6 feet to the right, perfectly positioned to trim the weeds at the side of the road. The problem is that the 6-foot pole is obstructed by sign posts that are positioned at the curb in certain parts of the city. The weed-trimmer pole, in fact, is exactly 2 feet too long to clear the sign posts. Although the weed-trimmer pole could be retracted or lifted out the way to clear the sign posts, this would interfere with the weed trimming. And although the pole could bend over the top of the sign posts, this would be impractical since in some areas the signs are 15 feet tall. The Department of Transportation is trying to design a pole that can pass *through* the sign posts without stopping or changing the position of the trimmer.

Fig. 1. L14's stimulus

In the space below, try to design a weed-trimmer pole that can pass through sign posts. Draw a diagram to illustrate what you're thinking.

The problem is 2 ft too long, so from 3½ ft to 4½ ft. Have a sliding bar that unlatches (twist) and retracts (Pull), for upcoming signs. And slides back up and latches when sign passes in. Then other arm does same and the sign passes through

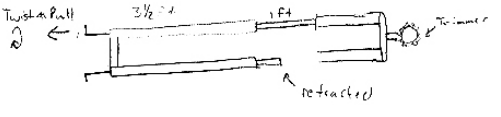


Fig. 2. The inscriptions L14 made on his or her experiment sheet

Succeeding states in the series of knowledge states are related through visual transformations such as move, rotate, scale and decompose. Each transformation relates two knowledge states. Transfer works by applying, step by step, each transformation in the source case to the knowledge states of the target case (See Figure 3).

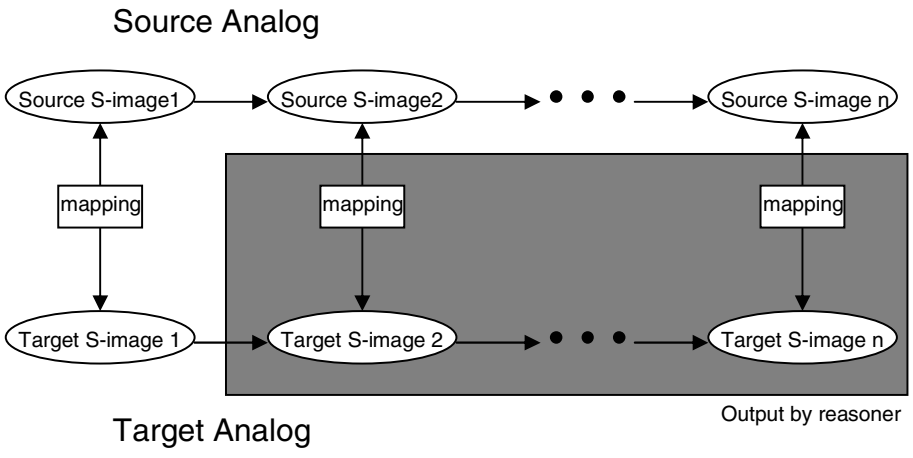


Fig. 3. Galatea's processing in the abstract

2.1 Knowledge Representation

Galatea describes visual cases using Covlan (Cognitive Visual Language), which 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 relating two elements with a relation.

Knowledge States: Knowledge states in Covlan are symbolic images, or s-images, which contain visual elements, general visual concepts, and relations between them. Cases are represented by a series of s-images, connected with transformations.

Visual Transformations. An s-image in the sequence is connected to other s-images before and after it with transformations. Transformations, like ordinary functions, take arguments to specify their behavior.

These transformations control normal graphics transformations such as translation (*move-to-location*), and rotation (*rotate*). In addition there are transformations for adding and removing elements from the s-image (*add-element, remove-element*). Certain transformations (*start-rotating, stop-rotating, start-translation, stop-translation*) are changes to the dynamic behavior of the system under simulation. For example, *rotate* changes the initial orientation of an element, but in contrast *start-rotating* sets an element in motion.

Primitive Elements are the visual objects in a diagram. The element types are *rectangle, circle, arrow, line,* and *curve.* Each element is represented as a frame with attribute slots, such as *location, size, orientation,* or *thickness.* A particular example of an element is referred to as an *element instance.*

General Visual Concepts. These act as slot values for the primitive elements as well as arguments for the visual transformations. The concepts are *location, size, thickness, speed, direction, length, distance, angle,* and *direction.* Each concept has several

values it can take. For example, the *size* can be *small, medium,* or *large,* and *thickness* can be *thin, thick* or *very-thick. Location* specifies an absolute qualitative location in an s-image (*bottom, top, center,* etc.)

Primitive Visual Relations. This class of symbols describes how certain visual elements relate to each other and to the values taken by general visual concepts. The visual relations are *touching, above-below,* and *right-of-left-of.* The motion relation is *rotation*.

Correspondence and Transform Representations. The knowledge of which objects in one s-image correspond to which objects in another is a *mapping,* which consists of a set of alignments between objects. Different sets of alignments compose different mappings. The i^{th} s-image in the source and the i^{th} s-image in the target have a *correspondence* between them; each correspondence (or *map*) can have any number of *mappings* associated with it (determining which mapping is the best is the "mapping problem.") The correspondence and mapping between the initial s-images (i=1) in the source and target is given as part of the input to Galatea; the system generates the subsequent correspondences and mappings.

Similarly, successive s-images in a series have *transform-connections*. These are needed so that Galatea can track how visual elements in a previous knowledge state change in the next.

2.2 Algorithm

Following is the control structure for Galatea's transfer of problem-solving procedures from a source case to the target problem. Figure 4 shows the s-image structure for L14's problem and solution. The Figure references in the algorithm description below refer to Figure 4.

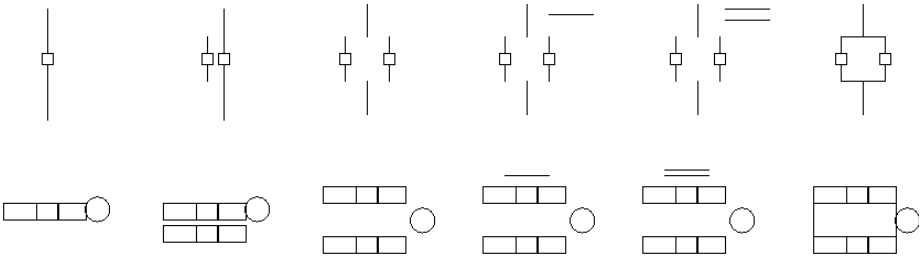


Fig. 4. Outline of our model of L14. The six images along the top represent the source, and the six images along the bottom the target, left to right, are separated by transformations: 1) *replicate*, 2) *add connections*, 3) *add component*, 4) another add *component*, and 5) *add connections*

The solution procedure (for the source, and then for the target) is that the doorway mechanism gets replicated, and then moved to the correct positions. Two walls are created to complete the vestibule, and finally they are placed in the correct position so that the vestibule is complete.

1. **Identify the first s-images of the target and source cases.** These are the current source and target s-images.

2. **Identify the transformations and associated arguments in the current s-image of the source case.** This step finds out how the source case gets from the current s-image to the next s-image. The model of L14 involves five transformations (see Figure 4). The first transformation is *replicate*. The second transformation is *add-connections* which places the door sets in the correct position in relation to the top and bottom walls. The third and fourth transformations are *add-component*, which adds the top and bottom containment walls. The fifth transformation, another *add-connections*, places these containment walls in the correct positions in relation to the door sets and the top and bottom walls.

3. **Identify the objects of the transformations.** The object of the transformation is what *object* the transformation acts upon. For L14's first transformation, this object is the parts of the door in the first s-image (we'll call it *door-set-l14s1*).

4. **Identify the corresponding objects in the target problem.** In the target, the trimmer arm's door mechanism is the corresponding object.

5. **Apply the transformation with the arguments to the target problem component.** A new s-image is generated for the target problem (bottom middle) to record the effects of the transformation. Replicate takes two arguments: some *object* and some *number-of-resultants*. In this case the *object* is *door-set-b1s1* (b1s1 means "base one, s-image two") and the *number-of-arguments* is two. The *replicate* is applied to the first L14 s-image, with the appropriate adaptation to the arguments: The mapping between the first source and target s-images indicates that the *door-set-b1s1* maps to the *door-set-l14s1*, so the former is used for the target's object argument. The number *two* is a literal, so it is transferred directly. Galatea generates *door-set1-l14s2* and *door-set2-l14s2* in the next s-image.

The second transformation is *add-connections*. The effect of this transformation is to place the replicated door-sets in the correct spatial relationships with the other element instances. It takes *connection-sets-set-b1s3* as the *connection/connection-set* argument. This is a set containing four connections. Galatea uses a function to recursively retrieve all connection and set proposition members of this set. These propositions are put through a function which creates new propositions for the target. Each proposition's relation and literals are kept the same. The element instance names are changed to newly generated analogous names. For example, *door1-endpoint-b1s3* turns into *door1-endpoint-l14s3*.

Then, similarly to the replicate function, horizontal target maps are generated, and the other propositions from the previous s-image are instantiated in the new s-image.

The inputs to this transformation are *nothing* (a literal denoting that there is not any thing in the previous s-image that is being modified), the connection set *connection-sets-set-b1s3*, the source s-image *lab-base1-simage2*, the current and next target s-images *l14-simage2* and *l14-simage3*, the mapping *l14-simage2—l14-simage3-mapping1,* and the rest of the memory.

6. **Map the original objects to the new objects in the target case.** A transform-connection and mapping are created between the target problem s-image and the new s-image (not shown). Maps are created between the corresponding objects. In this example it would mean a map between door-sets, as well as their component objects. Galatea does not solve the mapping problem, but a mapping from the correspon-

dences of the first s-image enables Galatea to automatically generate the mappings for the subsequent s-images.

7. **Map the new objects of the target case to the corresponding objects in the source case.** Here the parts of the door set in the target s-image are mapped to the parts in the second source s-image. This step is necessary for the later iterations (i.e. going on to another transformation and s-image). Otherwise the reasoner would have no way of knowing which parts of the target s-image the later transformations would operate on.

8. **Check to see if goal conditions are satisfied.** If they are, exit, and the solution is transferred. If not, and there are further s-images in the source case, set the current s-image equal to the next s-image and go to step 1.

We now present the main algorithm in pseudo code, followed by English descriptions of some of its functions.

```
Main
Input:
1. Source
2. Target Problem
3. Vertical mapping between source and target cases
Output:
1. A set of new target s-images
2. Vertical mappings between corresponding source and
        target s-images
3. Horizontal mappings between successive target states
4. Transformations connecting successive target states
Procedure
While more-source-states(goal-conditions, memory) do
          Current-target-s-image <- get-next-target-s-
            image(target problem, current s-image)
          Current-source-s-image <- get-next-source-s-
            image(source, current-s-image)
          Current-transformation <- get-
            transformation(current-s-image)
          Current-arguments <- get-arguments(current-
            source-s-image)
          Source-objects-of-transformation <- get-
            target-object-of-trans(current-source-s-
            image)
          Current-vertical-mapping <- get-
            mapping(current-target-s-image, current-
            source-s-image)
          Target-object-of-transformation <- get-source-
            object-of-transformation(current-vertical-
            mapping, source-objects-of-transformation)
          Target-arguments <- adapt-arguments(get-
            arguments(current-source-s-image)
          Memory <- memory + apply-
            transformation(current-transformation, tar-
            get-object-of-transformation, target-
            arguments)
          Memory <- memory + create-horizontal-
            mapping(current-target-s-image, get-next-
            target-s-image)
```

```
Current-target-s-image <- get-next-target-s-
  image(target problem, current-s-image)
Current-source-s-image <- get-next-source-s-
  image(source, current-s-image)
Memory <- memory + carry-over-unchanged rela-
  tionships(applied-transformation)
Memory <- memory + create-vertical-
  mapping(current-target-s-image, current-
  source-s-image)
```

Adapt Arguments. When an argument needs to be adapted to the target analog, Galatea looks at the argument and determines whether it is a literal, a function, or an element instance component of an s-image. Literals are returned verbatim. If the argument is a function (e.g. the number of people in a group) then Galatea applies the same function to the analogous group in the target and returns that value. If the argument is an element instance, then Galatea returns the analogous object in the target.

Carry Over Unchanged Relationships. The *get-analogous-chunks* sub-function constructs and returns chunks that are identical to the input chunks, except that the symbols that have maps in the input mapping are replaced with those symbols they are associated with in those maps. The vertical map relationships are carried over as well, constituting the vertical maps for unchanged element instances.

Creation of Horizontal Maps Between Changed Components. The *creation-of-horizontal-maps-between-changed-components* is embedded in each of the transformations. The transformation results are obtained from running the transformation. The *target-objects-of-transformation* are known because they are the input to the transformation. The two lists are put in alphabetical order and maps are created between each nth list object.

Creation of Horizontal Maps Between Unchanged Components. Similarly, *creation-of-horizontal-maps-between-unchanged-components* makes maps between old objects (the objects in the old s-image and new objects (from the current-s-image, minus the objects created by the transformation), alphabetizes them, and creates maps between the *n*th item in each list.

Creation of Vertical Maps Between Changed Components. The algorithm for creating vertical maps between changed components takes as input the transformation results in the source and target, alphabetizes them, and creates maps between the *n*th item in each list.

We can now evaluate what made L14's data (Fig. 2) differ from the stimulus drawing (Fig. 1): L14 features a longer vestibule in the drawing than the vestibule pictured in the stimulus. In fact, there is no trimmer arm (analogous to the wall in the lab problem) in the drawing at all that is distinct from the vestibule, save a very small section, apparently to keep the spinning trimmer blade from hitting the vestibule. The entire drawing is rotated ninety degrees from the source. The single lines in the source are changed to double lines in the target. The doors also slide in and out of the vestibule walls. What's interesting about this modification is that it does not appear that this kind of door opening is possible with the diagram given of the lab in the source: Since the door is a rectangle that is thicker than the lines representing the walls, the door

could not fit into the walls. In contrast L14 explicitly makes the doors and walls thick (with two lines) and makes the doors somewhat thinner. L14 adds objects to the target not found in the source: a blade and a twisting mechanism to describe how the doors can work. L14 also included numerical parameters to describe the design of the trimmer: to describe length. Finally, L14 includes some mechanistic description of how the trimmer would work.

Of these seven differences, our model successfully re-creates four of them. The *rotation* of the source is modeled by a rotation in the target start s-image. In the s-image, all spatial relationships are defined only relative to other element instances in the s-image. Each instance is a part of a single set which has an orientation and direction. In the case of s-image 1 of the target, it is facing right. Since all locations are relative, there is no problem with transfer and each s-image in the model of L14 is rotated to the right. The *line to double line* difference is accounted for by representing the vestibule walls with rectangles rather than with lines, as it is in the source. Because the mapping between the source and target correctly maps the *side1* of the rectangle to the *startpoint* of its analogous line, the rectangle/line difference does not adversely affect processing transfer. The *long vestibule* difference is accounted for by specifying that the heights of the vestibule wall rectangles are *long*. In the source the vestibule wall lines are of length *medium*, but this does not interfere with transfer. The trimmer head *added object* is accounted for by adding a circle to the first s-image in the target.

Unaccounted for are the two bent lines emerging from the vestibule on the left side, the numeric dimensions and words describing the mechanism. Also, L14 shows one of the doors retracting, and the model does not. The model also fails to capture the double line used to connect the door sections, because the single line is transferred without adaptation from the source. This could be fixed, perhaps, by representing the argument to the *add-component* as a function referring to whatever element is used to represent another wall, rather than as a *line.*

3 The Galatea Model of Participant L22

L22 worked on the same problem as L14 and received in his or her stimuli the image presented in Fig. 5. The marks L22 made on the experiment sheet are reproduced in Fig. 6. Our model of L22 involves five transformations. The first transformation is *replicate*. To *replicate* the door mechanism, the starting state, s-image 1, must have a single door. A portion of the information in the first s-image of the source can be seen in Fig. 7. All of the objects in Fig. 7 are a part of *door-set-s1*. It takes in the *door-set1-s1* as an argument, generating *door-set1-s2* and *door-set2-s2* as output in the second s-image. There are three connected rectangles, corresponding to the top wall, door, and bottom wall. The second transformation is *add-connections* which places the door sets in correct position in relation to one another. The third and fourth transformations are *add-component*, which add the top and bottom containment walls. The fifth transformation, another *add-connections*, places these containment walls in the correct positions in relation to the door sets.

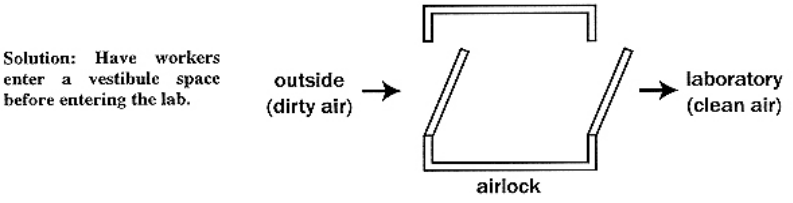


Fig. 5. The image from the source stimulus that L22 received in the experiment. It is a top-down view of the airlock. This stimulus's text (not shown) is identical to that of L14 (Figure 1)

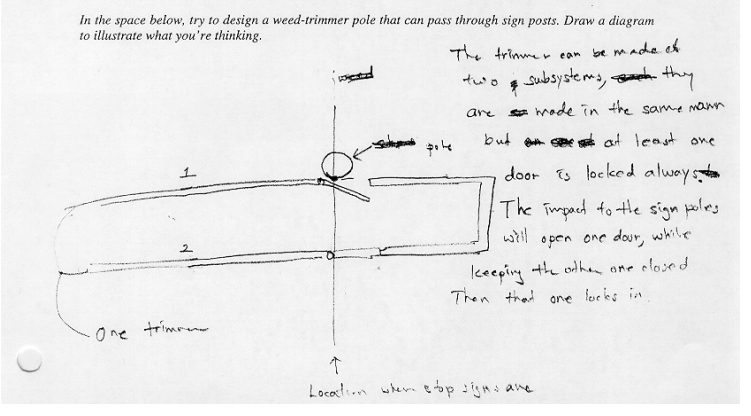


Fig. 6. The marks L22 made on his or her experimental sheet

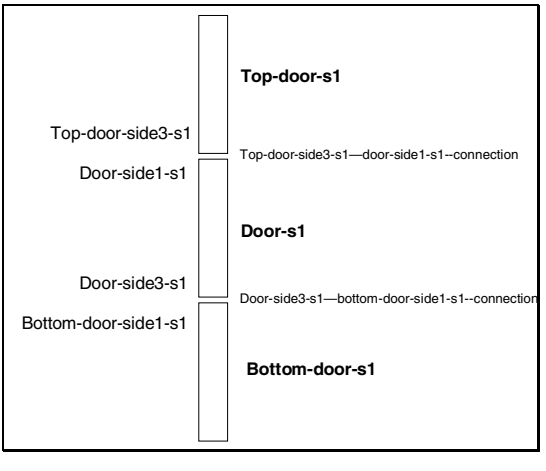


Fig. 7. A portion of the first s-image of the L22 model. S1 refers to the fact that the symbols are in the first s-image. The top-door, door, and bottom-door are all in the door set that gets replicated in transformation one

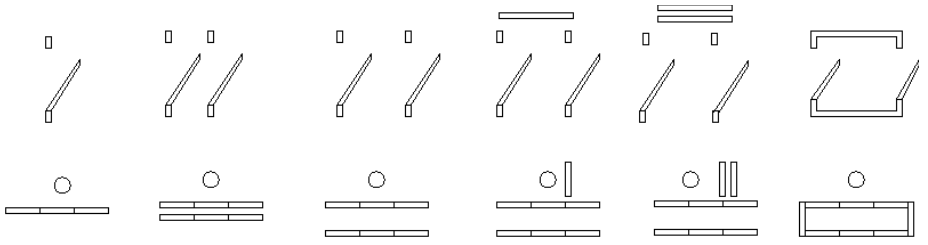


Fig. 8. Outline of our model of L22. The six images along the top represent the source, and the six images along the bottom the target, left to right, are separated by transformations: 1) *replicate*, 2) *add connections*, 3) *add component*, 4) *add component*, and 5) *add connections*

The images along the top of Fig. 8 represent the source s-images. The images along the bottom of Fig. 5 show the sequence of target s-images. Only the first s-image of the target is given as input to the system. The rest are generated by Galatea. The circle depicted represents a cross-section view of the sign that must pass through the arm. The door mechanism in the arm, which gets replicated and connected up properly, is oriented differently than the door in the lab problem, but the transformations are general enough to allow transfer. In the bottom right s-image we can see the solution state, as generated by Galatea. The redundant door mechanism will allow the sign to pass through one, into the vestibule, while the other door keeps the structure in place. Then the first door can close, supporting the structure, while the second door opens to let the sign post out the other side.

We can now examine what made L22 differ from the stimulus drawing: The entire drawing is rotated ninety degrees from the source. An object is added to the target that has no analog in the source: the trimmer. L22 features a proportionately longer vestibule than in the source, and has some explicit simulation diagrammed. Of these differences, all but the last were modeled by changing the nature of the start s-image for L22.

4 Related Work

In the introduction, we noted that the issue of visual knowledge in case-based reasoning is very broad and thus has attracted the attention of many researchers in several areas. In order to look at the issue deeply, we focused our discussion exclusively on case-based problem solving, i.e. case-based transfer of a procedure from a source case to a target case. Below we relate our work to some representative case-based problem solving systems with emphasis on systems use visual knowledge in transferring problem-solving procedures.

FABEL [8] is an example of a case-based system that adapts diagrammatic cases in the domain of architectural design. In FABEL, the source diagram specifies the spatial layout of a building or similar structure. FABEL adapts source diagrams by extracting and transferring specific structural patterns to the target problem. It uses domain-specific heuristics to guide pattern extraction and transfer. Galatea too adapts diagrams by extracting and transferring patterns. Pattern transfer in Galatea is facili-

tated by three main elements. Firstly, Galatea explicitly represents the knowledge states of its source cases in the form of s-images. Secondly, each s-image is composed of primitive visual elements and relations. Thirdly, succeeding knowledge states in Galatea's source cases are related by primitive visual transformations. In this way, Galatea captures the diagrammatic *problem solving* of the source cases. Given a mapping between the visual elements in the target problem and a source case, this knowledge enables Galatea to extract and transfer the appropriate series of visual transformations from the source case to the target problem. In particular, the knowledge states identify the names and arguments of specific transformations that need to be transferred from the source case to the target problem.

REBUILDER [9] is a case-based reasoner that does retrieval, mapping, and transfer of software design class diagrams. The diagrams are represented structurally, not visually, however. This means that, for example, that the connection is between two nodes is more important than the length and direction of that connection. That is, REBUILDER works with a different level of visual abstraction, a level at which only the structural relationships, such as connectedness, between visual elements are relevant to the task. In contrast, Galatea takes into account additional geometric information such as the length, direction and thickness of lines. What is the right level of visual abstraction for visual case-based problems requires additional research. The choices made by Galatea and REBUILDER depend largely on the specific domains in which they operate. In REBUILDER's domain of software design class diagrams, only the structural relations appear to be important.

FAMING [6] is a case-based reasoning system that uses cases describing physical mechanism parts. FAMING uses the SBF (Structure-Behavior-Function) ontology to describe the cases. The structure is described in terms of a metric diagram (a geometric model of vertices and connecting edges), a place vocabulary (a complete model of all possible qualitative behaviors of the device), and configuration spaces (a compact representation of the constraints on the part motions.) Shape features can involve two objects, expressing, for example, one part's ability to touch another part. Human designers are necessary for FAMING's processing. The designer chooses which cases and functions should be used, which dimensions the system should attempt to modify, and which shape features should be unified. It uses qualitative kinematics to propose design solutions for the desired function following the designer-suggested idea. Though not described as a visual system, the important parts of physical mechanisms of the sort FAMING uses inevitably contain much knowledge that could be construed as visual. However, FAMING modifies cases according to shape substitution, and, unlike Galatea, makes no attempt to transfer strongly-ordered procedures of any sort.

Non-visual case base problem-solving systems, such as CHEF [10] and PRODIGY [14] provide interesting points of comparison regarding the transfer process. CHEF is a case-based reasoner that transfers and adapts cooking recipes from a source to a target. CHEF does not create intermediate knowledge states. This is because it does not transfer procedures that create new objects. The Prodigy case-based reasoning system implements the theory of Derivational Analogy. It models transfer using memories of the justifications of each step, allowing for adaptation of the transferred procedure. Traces, called "derivations," are scripts of the steps of problem solving, along with the justifications for why the steps were chosen over others. PRODIGY too does not store the intermediate steps; instead it stores only a record of the changes

made to them. This means that the states can be inferred, but are not explicitly present in the case memory. CHEF and PRODIGY avoid the generation of intermediate knowledge states and mappings because the examples with which they have been implemented do not have procedures that create new objects.

5 Conclusions

In the introduction to this paper, we described the two main hypotheses of this work: (1) visual knowledge alone is sufficient for transfer of problem-solving procedures in some domains, and (2) visual knowledge facilitates transfer even when non-visual knowledge might be available. Both hypotheses were strongly supported by the evidence described above, and we had an unexpected discovery of a third, which makes for three claims.

First, visual knowledge is sufficient for transfer of some problem-solving procedures. There are seven models written in Galatea that support this claim. We described the models of L14 and L22 in this paper. We modeled two additional participants from the Craig *et al.* experiment, a historical example from the scientific thinking of Maxwell [5], the fortress/tumor problem [4] and the cake/pizza problem [4]. Each of these models uses case-based reasoning to solve a problem using only visual knowledge. The fact that four of these models are based on human experimental participant data lend support to the hypothesis that this claim might apply to human problem solving, as well as artificial case-based reasoning systems, although more empirical research would be needed to substantiate this. As shown above, most of the differences between source and target, as displayed in the participant data, were accounted for in our models. In light of this research we can speculate for which domains visual knowledge might be sufficient for transfer of problem-solving procedures: those domains, the solution procedures of which *could* be adequately described with descriptions of changes to visio-spatial properties. A way to think about this is if the important differences between the problem and the solution are reflected in *visual* differences, then that problem is likely to fall in this class.

The second claim is that visual knowledge facilitates transfer even when non-visual knowledge might be available. L22's lab/weed trimmer problem involves physical systems that can be described visually or non-visually. Galatea's visual ontology of primitive elements and transformations allows transfer between systems that, though they may be semantically distant, have visual similarities, which facilitates the transfer. This is also true for the three other lab/weed trimmer participants, as well as for the fortress/tumor example.

In the course of building the models of Galatea, we discovered that the successful transfer of strongly-ordered procedures in which new objects are created requires the reasoner to generate intermediate knowledge states and mappings between the intermediate knowledge states of the source and target cases. Galatea shows why, in detail, this is so. Components of the problem are *created* by the operations, and these components are acted on by later operations. For L22's problem, for example, the door set must be replicated before the two sets can be moved in relation to one another. When the reasoner transfers the second operation of moving the door sets, how does it know what the corresponding objects are in the target? It must have some

mapping to make this inference. And since one of the door sets did not exist in the start states of the problems, this mapping cannot be given as input with the initial mapping. The new knowledge state with the duplicated door set must be generated, and then a mapping must be made on the fly between it and the second knowledge state of the source case.

References

1. Barber, J., Jacobson, M., Penberthy, L., Simpson, R., Bhatta, S., Goel, A., Pearce, M., Shankar, M. & Stroulia, E. Integrating artificial intelligence and multimedia technologies for interface design advising. *NCR Journal of Research and Development,* 6(1), 75—85, October 1992.
2. Casakin, H., Goldschmidt, G.: Expertise and the use of visual analogy: Implications for design education. *Design Studies.* (1999)
3. Craig, D. L., Catrambone, R., Nersessian, N. J.: Perceptual simulation in analogical problem solving. In *Model-Based Reasoning: Science, Technology, & Values.* New York: Kluwer Academic / Plenum Pubishers. (2002) 167–191
4. Davies, J., Goel, A. K.: Representation issues in visual analogy. *Proceedings of the 25th Annual Conference of the Cognitive Science Society.* (2003) 300--305.
5. Davies, J., Nersessian, N. J., Goel, A. K.: Visual models in analogical problem solving. *Foundations of Science, Special Issue on Model-Based Reasoning: Visual, Analogical, Simulative.* By Magnani, L. and Nersessian, N.J. (Eds.) (in press)
6. Faltings, B., Sun, K.: FAMING: Supporting innovative mechanism shape design. *Computer-Aided Design,* 28(3) (1996) 207—216
7. Farah, M. J.: The neuropsychology of mental imagery: Converging evidence from brain-damaged and normal subjects. In *Spatial Cognition- Brain Bases and Development.* Erlbaum (1988)
8. Gebhardt, F., Voss, A., Grather, W.: *Reasoning with Complex Cases.* Kluwer (1997)
9. Gomes, P., Seco, N., Pereira, F. C., Paiva, P., Carreiro, P., Ferreira, J. L., Bento, C.: The importance of retrieval in creative design analogies. In *Creative Systems: Approaches to Creativity in AI and Cognitive Science.* Workshop program in *The Eighteenth International Joint Conference on Artificial Intelligence.* (2003)
10. Hammond, K. J.: Case-based planning: A framework for planning from experience. *Cognitive Science*. (1990)
11. Monaghan, J. M., Clement, J.: Use of computer simulation to develop mental simulations for understanding relative motion concepts. *International Journal of Science Education.* (1999)
12. Gebhardt, F., Voss, A. Grather, W. & Schmidt-Belz. *Reasoning With Complex Cases*, Kluwer, 1997.
13. Pearce, M., Goel, A. K., Kolodner, J. L., Zimring, C., Sentosa, L., & Billington, R. Case-based design support: A case study in architectural design. *IEEE Expert: Intelligent Systems & Their Applications.* 7(5): 14-20, (1992)Shepard, R., Cooper, L.: *Mental Images and Their Transformations.* MIT Press. (1988)
14. Veloso, M. M.: Prodigy/analogy: Analogical reasoning in general problem solving. *EWCBR* (1993) 33—52