Design, Analogy, and Creativity

Ashok K. Goel, Georgia Institute of Technology

ANALOGICAL REASONING APPEARS TO PLAY A KEY ROLE IN CREATIVE DESIGN. THIS ARTICLE BRIEFLY REVIEWS RECENT AI RESEARCH ON ANALOGY-BASED CREATIVE DESIGN BEFORE ENUMERATING A RELATED SET OF RESEARCH ISSUES.

THAT DESIGN CAN BE ROUTINE, innovative, or creative, has become a standard cliche among researchers building AI theories of design. Whether or not this is an especially productive classification of design processing, the cliche indicates that innovation and creativity in design are important research topics. That analogy plays a central role in innovation and creativity is another common premise among AI in design researchers. Whether or not current theories of analogical reasoning lead to creative design, this premise has drawn increased attention in recent AI research on creative design.

In briefly reviewing recent research on analogy-based creative design, this article first examines characterizations of creative design and then analyzes theories of analogical design in terms of four questions: *why*, *what*, *how*, and *when*. After briefly describing recent AI theories of analogy-based creative design, the article focuses on three theories instantiated in operational computer programs: Syn,[1] Dssua,[2] and Ideal.[3] From this emerges a set of research issues in analogy-based creative design. The main goal here is to sketch the core issues, themes, and directions in building such theories.

Creative design

AI characterizations of creative design tend to be of two kinds. The first begins with a general theory of information processing and then delimits routine, innovative, and creative design in the terms of the theory. Allen Newell and Herbert Simon's theory of *problem-space search* is a very general theory of human information processing.[4] In this approach, designers solve problems by searching in problem spaces, where a problem space is abstractly defined by the reasoning goal and by domain knowledge in the form of operators that enable space search. This theory can, and has, served for modeling design processing. It specifies the reasoning goals as part of the design requirements that express design variables and specific ranges of values they can take; the operators are specific to the particular design domain. Several investigators have classified design processing in terms of problem-space search in this way: [5]

- if the design variables and the ranges of values they can take remain fixed during design processing, the design is routine;
- if the design variables remain fixed but the ranges of values change, the design is innovative; and
- if the design variables and the range of values both change, the design is creative.

The second kind of characterization arises from specific AI theories of design. The AI in design theorist posits a computational process for a kind of design activity, and then delimits routine, innovative, and creative design in the terms of the theory. For example, David Brown and B. Chandrasekaran

0885-9000/97/$10.00 © 1997 IEEE

describe a computational theory of routine design based on the notion of *design plans*.[6] This theory uses a structure-substructure hierarchy to organize skeletal design plans for the object to be designed. Designing a particular instance of the object—a particular air pump with specific dimensions, for example—involves selecting, instantiating, and expanding the design plans. Brown further proposes a classification of design processing based on this theory:[7]

- if the designer knows both the structure of the design space (the structure-substructure hierarchy of the object) and procedures for systematically searching the space (the skeletal design plans), the design is routine;
- if the designer knows only the structure of the design space, the design is innovative; and
- if the designer knows neither, the design is creative.

Three implications follow from these characterizations. The first characterization suggests that problem formulation and reformulation are integral parts of creative design. A designer's understanding of a problem typically evolves during creative design processing. This evolution of problem understanding can lead to (possibly radical) changes in the problem and solution representations.

The second characterization suggests that in creative design, knowledge needed to address a problem typically is not available in a form directly applicable to the problem. Instead, at least some of the needed knowledge must be acquired from other knowledge sources, by analogical transfer from a different problem, for example.

Both characterizations indicate that creativity in design lies on a continuum. That is, creativity in design can occur in degrees, where the degree of creativity depends on the extent of problem and solution reformulation and the transfer of knowledge from different knowledge sources to the design problem.

Analogical design

Recent AI research on creative design has explored the use of analogies in proposing solutions to design problems in the design process's conceptual phase. This focus on the use of analogies in proposing candidate designs appears to be a result of the traditional characterization of analogical reasoning in the AI in design research community: analogical design involves reminding and transfer of elements of a solution for one design problem to the solution for another design problem, where the selected design elements can be components, relations between components, or configurations of components and relations. That is, given a problem P_{new} and a (partial, possibly null) solution S_{new} for P_{new}, analogical reasoning involves retrieval of a familiar problem P_{old} from memory with a solution S_{old}, and transfer of selected elements from S_{old} to S_{new}.

Let's examine this characterization from the perspective of analogy-based creative design. It is productive to ask four questions in analyzing AI characterizations and theories of information-processing phenomena:

ANALOGICAL DESIGN INVOLVES REMINDING AND TRANSFER OF KNOWLEDGE ABOUT ONE DESIGN SITUATION TO ANOTHER, WHERE THE TRANSFER CAN OCCUR IN THE SERVICE OF ANY DESIGN TASK IN THE NEW SITUATION.

why, what, how, and *when*. In the context of analogical design, the *why* question pertains to the *task* (or the goal) for which analogy is used—for example, the task of proposing a candidate design. The *what* question pertains to the *content of knowledge* that is transferred—for example, transfer of knowledge of the heat-flow process from one design situation to another. The *how* question pertains to the *methods* for reminding and transfer, where a method uses specific kinds of knowledge representations, inference mechanisms, and control strategies. Finally, the *when* question pertains to strategic control of processing. Clearly, the four questions are closely interrelated.

If we ask these questions about the traditional characterization of analogical design, the characterization appears too narrow, both in terms of what can be transferred and why it can be transferred. (This characterization also implies that P_{old} and P_{new} are different design problems. But, in general, P_{old} and P_{new} can be subproblems of the same design problem. This is the familiar distinction between cross-problem and within-problem analogies sometimes made in the literature on analogical reasoning.[14])

Why? This characterization implies that analogical reminding and transfer occurs in service of generating the solution S_{new} to the design problem P_{new}—that is, in service of proposing a candidate design, modifying an initial design, and completing a partial design. But there is no principled reason to limit analogies to these design tasks. Design, especially creative design, involves a variety of other design tasks, such as interpreting and elaborating the design problem, decomposing the problem, anticipating potential difficulties with a candidate solution, refining a candidate design, evaluating a candidate design, interpreting the evaluation information, and reformulating the problem. Analogies, in general, can help address *any* of these design tasks. (The KA system, for example, uses past cases for the task of interpreting a design problem.[8])

What? The answer depends in part on the design task being addressed. The earlier characterization of analogical design implies that the content of knowledge transfer is in the form of design elements—for example, components and relations between components. This kind of knowledge transfer seems appropriate for the tasks of design proposition, modification, and completion. For other design tasks, however, analogies can result in the transfer of different kinds of knowledge—for example, the transfer of knowledge of familiar design problems for the tasks of interpreting and reinterpreting a new problem, and knowledge of criteria and methods of evaluating familiar designs for the task of evaluating a candidate solution to a new problem. In addition, for any of these design tasks, the transfer can take the form of *strategic knowledge* instead of domain knowledge. The transfer of a method for problem decomposition is but one example of this kind of analogy. The transfer of a design strategy in the form of a task structure is a more general example.

For all these reasons, we can generalize the traditional characterization of analogical design into an initial characterization like this: analogical design involves reminding and transfer of knowledge about one design situation to another, where the transfer can occur in the service of any design task in the

new situation, and the content of the transfer can be knowledge of a design problem, solution, domain, or strategy.

How? This question concerns methods for reminding and transfer. The answer, in general, depends on the answers to the why and what questions about a design situation. The method for analogical transfer in design, for example, might depend both on the design task and the transfer's knowledge content.

Case-based reasoning provides one general answer to the how question. Given a problem P_{new}, the designer is first reminded of a familiar problem P_{old} with a solution S_{old}, where P_{old} and P_{new} are so similar that S_{old} is an approximate solution for P_{new}. Then, the designer modifies selected components in S_{old} to obtain a candidate solution S_{new} for P_{new}. Thus, in case-based design, the entire solution S_{old} is transferred to S_{new} and modified to fit the specifications of P_{new}. Case-based reasoning, therefore, appears to be a limiting case of analogical reasoning in which the question of what to transfer degenerates into the question of what to modify. The case-based strategy, nevertheless, fits many tasks in variant design and some tasks in adaptive design. (Pahl and Beitz provide characterizations and examples of variant and adaptive design.[9])

The case-based reasoning literature often distinguishes between transformational and derivational analogy.[10] In transformational case-based reasoning, the solution S_{old} for P_{old} is modified to obtain the solution S_{new} for P_{new}. The modification knowledge typically is associative, and often is based on domain-specific heuristics. In derivational case-based reasoning, the trace of problem-space search that led to the solution S_{old} for P_{old} guides the adaptation of S_{old}. Although both methods have been tried in design, they have met only limited success. Associative methods for design modification appear adequate at best only for variant design in weakly interacting domains, and processing traces of design generation by space search typically are not available in practical design. Instead, AI in design research appears to have led to a third framework for case-based design that uses *model-based methods* for design modification and other subtasks spawned by the case-based strategy. The model-based methods use "deeper" design knowledge—for example, knowledge of the topology and teleology of S_{old}. (The article by Mary Lou Maher and Andrés Gomez in this special double issue presents an overview of case-based design.[11] Elsewhere, I sketch an early theory of using model-based methods for design modification in case-based design.[12] Linda Wills and Janet Kolodner discuss some issues in pushing traditional case-based reasoning towards innovative design.[13])

The literature on analogical reasoning sometimes distinguishes between within-domain and cross-domain analogies.[14] A domain can be characterized by the objects, relations, and processes that occur in it. Design domains such as engineering, architecture, software, and interface design clearly involve different kinds of objects, relations, and processes. But from the viewpoint of building AI theories, whether two design domains are very different, quite similar, or identical depends on the language for representing the objects, relations, and processes. From this viewpoint, the design domains of mechanical and electrical engineering are very different if one uses different representational languages for them, or similar if one uses the same language. Whether an AI theory of design can use the same representational language for two (apparently different) domains depends on the level of design detail.

ANALOGICAL TRANSFER REQUIRES THE USE OF GENERIC ABSTRACTIONS, WHERE THE ABSTRACTIONS TYPICALLY EXPRESS THE STRUCTURE OF RELATIONSHIPS BETWEEN GENERIC TYPES OF OBJECTS AND PROCESSES.

Again, the answer to the *how* question provided by case-based reasoning seems limited to design situations in which P_{old} and P_{new} are so similar that all of S_{old} can be transferred to S_{new} and selectively modified to fit P_{new}. What happens when P_{old} and P_{new} are not quite so similar? For example, what happens if P_{old} and P_{new} are problems from two different design domains in that not all the objects, relations, and processes in the two domains are identical? This question relates directly to the issue of creativity in design: if P_{old} and P_{new} are almost identical problems, S_{old} is unlikely to suggest changes to the variables characterizing P_{new}.

In general, analogical transfer requires the use of *generic abstractions*, where the abstractions typically express the structure of relationships between generic types of objects and processes. The analogical reasoning literature in AI,[15] cognitive psychology,[16] and cognitive science[17] suggests that the generic abstractions are not merely abstractions over features of objects, but that they capture the relational structure among objects and processes. In the context of design, generic abstractions might specify, for example, the structure of geometric, topological, temporal, causal, and functional relations among design elements. Generic design abstractions might also specify the structure of goals and methods in a design strategy. This implies that the learning of generic design abstractions is another important process in analogical design.

For this reason, we can specialize the initial characterization of analogical design like this: analogical design involves learning and transfer of generic design abstractions from one design situation to another, where the generic design abstractions specify the structure of relations among the elements of a design problem, solution, domain, or strategy, and where the transfer can occur in the service of any design task in the new situation.

When? This question pertains to the strategic control of processing. The learning of generic design abstractions provides a good illustration of this issue. Generic design abstractions can be learned at storage time, when the designer stores a new design in the design memory. This is an example of "eager" learning in which abstractions over known designs are learned as soon as new design knowledge becomes available.

Alternatively, learning might occur at retrieval time, when the designer is reminded of a known design. As yet another alternative, learning might occur at problem-solving time, when the designer transfers knowledge from one design situation to another. The latter two are examples of "lazy" learning, in which abstractions are generated when needed. Clearly, different answers to the when question about this learning result in different AI theories of analogy-based creative design. In addition, the different answers to this question might imply different answers to the questions of what gets abstracted and transferred and how.

Three theories of analogy-based creative design

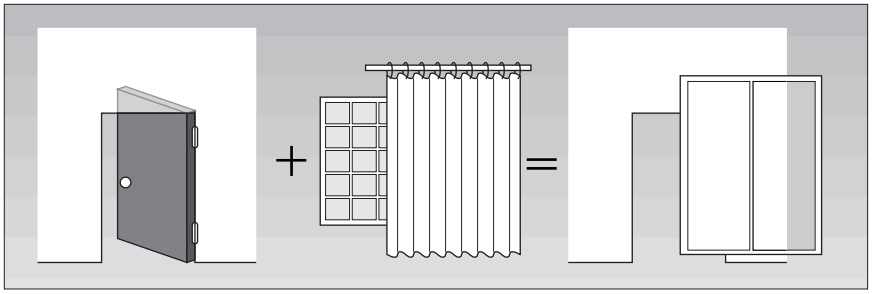

Figure 1. Analogical design of a sliding door in Dssua.

I have selected three theories to focus on that exemplify recent work on analogy-based creative design. The three theories are instantiated in operational computer systems called Dssua,[10] Ideal,[3,18,19] and Syn.[1] Dssua was developed in the early 1990s in Australia, Ideal in the early to mid-1990s in the US, and Syn in the mid-1990s in Germany. All three address the conceptual phase of the design process and use model-based methods for analogical transfer. However, they illustrate three kinds of generic abstractions useful in analogy-based creative design: *design prototypes* in the case of Dssua, *design patterns* in Ideal, and *design concepts* in Syn.

Design concepts as generic abstractions. Katy Börner, Eberhard Pippig, Elisabeth Tammer, and Carl Coulon describe a module called Syn within the Fabel system that enables Fabel to support analogy-based creative design.[1] Fabel is a case-based CAD environment intended to assist human architects in designing spatial layouts of buildings. Syn addresses the problem of designing air-circulation systems within the context of spatial layout design. A design problem presented to Syn specifies the main access to the air-circulation system, the required air outlets, and possibly some pipes connecting them. Initial problem specification takes the form of attributes and values that express the coordinates of the access, outlets, and pipes in a fixed grid, together with other features. Syn generates candidate designs that specify all the air-circulation pipes that connect the access and the outlets. Syn also represents the candidate designs in the form of coordinates of the main access, outlets, pipes, and interconnections in a fixed grid.

Familiar air-circulation designs in Syn are represented as *graph-based topological models*. In the graph-based representation, the main access, the outlets, and the pipe interconnections are the nodes, and the air-circulation pipes are the links. The graph-based design representations are organized in a tree of *design concepts*, where a design concept corresponds to a (named) maximal common subgraph between the designs under the concept. Using Ideal's terminology, one might view SYN's design concepts as *topological design patterns*.

Syn uses a *compile* operator for converting a problem specification from an attribute-value feature representation to a graph-based topological representation. In addition, it uses an inverse *recompile* operator for converting a graph-based representation for a candidate design to the attribute-value representation. When the task specifies a design problem (and its partial solution), Syn first uses the compile operator to generate a graph-based representation for it. Then, the system classifies the problem into its design concepts and retrieves the designs under it. The classification process is based on *structural similarity* between the problem and the concepts, and results in the most refined match possible.

To transfer knowledge from a retrieved design to the given design problem, Syn instantiates the maximal common subgraph (or topological pattern) for the retrieved design in the context of the design problem. This step adds air-circulation pipes and interconnections to the partial solution specified as part of the design problem. Finally, Syn uses the recompile operator to convert the graph-based representation of the new design solution into the attribute-value representation. At this stage, the system can change the dimensions of the pipes in the candidate design to meet the specifications of the design problem.

Syn is a very simple theory of analogical design. Its innovation appears to be quite mundane. The basic process classifies the given design problem into a library of stored design concepts that express topological patterns, retrieves the best matching concept, and instantiates the concept in the context of the problem to generate a candidate solution. Syn does not abstract any concept from specific designs; instead, it assumes the library of design concepts as given. In addition, it leaves the tasks of evaluating a candidate design and assimilating the evaluation information entirely to the human designer. Much of the power of the system apparently arises from its organization of the library of design concepts and its ability to transform a problem specification from its initial representation to a representation that enables reminding and instantiation of design concepts.

Design prototypes as generic abstractions. Lena Qian and John S. Gero describe an interactive analogy-based creative design system called Dssua (Design Support System Using Analogy), which actually predates Syn by a few years.[2] Dssua addresses mechanical design problems within the context of architectural design—for example, designing a door to a house. Figure 1 illustrates Dssua's design for a sliding door that is based in part on analogy to the design of window curtains.

Dssua stores knowledge of familiar designs (such as window curtains) in the form of *design prototypes*, where a design prototype is an abstraction over specific design instances. It represents a design prototype in the form of a *function-behavior-structure* model. The system represents the structure in an FBS model of a design prototype in the form of an undirected *structure graph* that abstractly specifies the components and relations among them, but not the specific values of the variables characterizing the components and relations in a particular design instance. As an example, the structure graph for the design prototype of window curtains might specify components such as window, rod, rings, and curtain; and relations such as "the rod is screwed to the window," "rings slide along the rod," and "the curtain is linked to the rings." Dssua represents the behavior as a directed *behavior graph* that, in addition to the structure, also specifies the direction of causal influence among the variables characterizing the components. The bottom half of Figure 2 illustrates the behavior graph for the window curtain. It specifies, for example, that the area covered by the curtain is directly proportional to the distance between the sliding rings on the rod. In the terminology of Ideal, one might view Dssua's design prototypes as *design patterns with physical structure*.

A design problem for Dssua is specified in terms of a structure graph. The structure graph for the problem of designing a door, for example, might specify the components (door frame, hinge, and door) and the component relations (hinges screwed to the

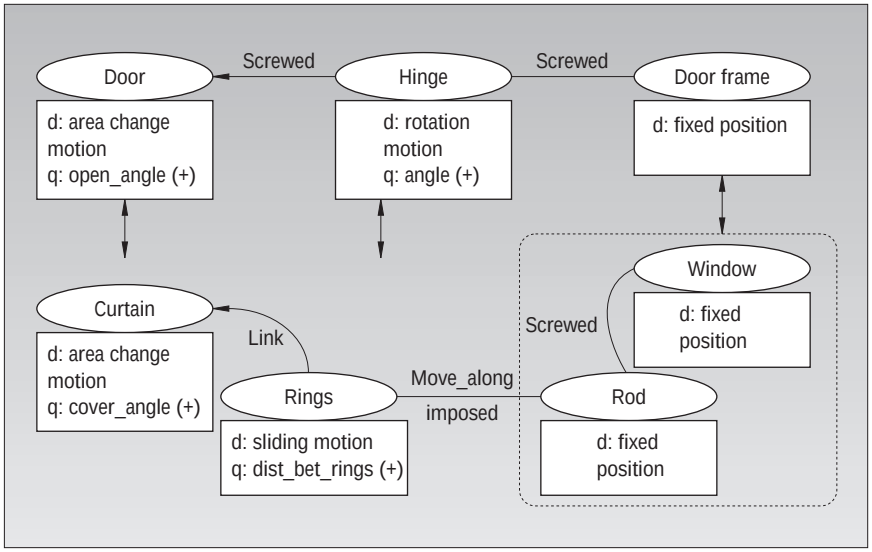

Figure 2. Mapping between behavior graphs in Dssua.

frame, door screwed to the hinges) of an ordinary door. Thus, the problem representation specifies both the problem (design of a door) and an abstract solution for it (the ordinary hinged door). Given the structure graph of a design problem and the initial solution for it, Dssua uses the structure graph as a probe into its library of design prototypes and retrieves a *structurally similar* design prototype. In the door design example, it retrieves the design prototype for the window curtain based on the similarity between the structure graphs of the initial door design and the curtain design prototype.

Next, Dssua uses domain-specific heuristics for constructing a behavior graph from the structure graph of a given design problem and its initial solution. For the initial door design, for example, it derives the behavior graph illustrated in the top half of Figure 2. This behavior graph includes the additional knowledge that the angle by which a hinged door is open is directly proportional to its rotation relative to the hinge fixed to the frame. The system then compares the behavior graphs of the design problem and the retrieved design prototype. If the two behavior graphs are isomorphic, the mapping between the design elements in the retrieved prototype and the given problem directly enables the system to transfer specific elements from the design prototype to the initial solution specified as part of the design problem.

If the two behavior graphs are not isomorphic, Dssua abstracts qualitative causal dependencies from the two behavior graphs. The behavior graphs for the initial door design and the curtain design prototype illustrated in Figure 2, for example, are not isomorphic. Therefore, the system abstracts directed dependency graphs that represent causal dependencies among the design variables but do not specify particular structural components or relations among them. In the case of our running example, Dssua abstracts a dependency graph from the behavior graph for the door design, which specifies that the angle of openness is directly proportional to the degree of rotation. Similarly, it abstracts a dependency graph from the behavior graph for the curtain design prototype, which specifies that the area covered is directly proportional to the sliding distance. Thus, abstraction in Dssua occurs at transfer time.

Next, the system compares the structures of the dependency graphs abstracted from the design prototype and the design problem along with its initial solution. If it finds similar structures between the two dependency graphs, it conjectures a transfer of the corresponding design elements from the design prototype to the initial solution for the given design problem. In our running example, Dssua hypothesizes a similarity between the causal relations in dependency graphs for the design prototype and the design problem. This enables it to generate a new candidate solution for the problem of door design in the form of a sliding door. Thus, transfer in Dssua is based on a similarity between the structures of the dependency graphs abstracted from the design prototypes and the initial solutions specified as part of design problems. Finally, the human designer can evaluate a candidate solution generated by Dssua for both local and global consistency.

Dssua's theory of analogy-based creative design illustrates the use of structure of representations for cross-domain analogical transfer. The representational structure itself depends on the representational vocabulary. The system represents design prototypes in the language of FBS models, and the structure of the representation of a prototype thus depends on the FBS vocabulary. Therefore, much of the power of the system apparently arises from its uses of generic abstractions in the form of design prototypes, its representation of design prototypes in the form of FBS models, and its ability to abstract causal dependency graphs from FBS models at transfer time.

This kind of analogical transfer introduces new variables into the initial solution for the given design problem. For example, in the case of initial door design, transfer of knowledge from the window-curtain prototype introduces the variable of sliding motion in the design of the door. Thus, given our characterization of creative design, Dssua is a creative design system.

Design patterns as generic abstractions. In my research group, Sambasiva Bhatta has developed a computational theory of analogy-based creative design called *model-based analogy*. He has also developed the Ideal system, which instantiates and evaluates the theory for conceptual design of engineering devices such as automatic coffee makers, electronic amplifiers, and gyroscopes. Ideal is a multistrategy system, capable of both case-based and analogical design. It contains several kinds of domain knowledge: design analogs (or cases), design patterns, design concepts, generic design components, and generic domain substances. Design analogs take the form of *structure-behavior-function* models.

The SBF model of a device is based on a general ontology of flow of substances through device components. This ontology gives rise to a set of design concepts in the form of structural connections and behavioral interactions among the components and substances of devices, behavioral states and state transitions in devices, and functions of devices and device components. These design concepts provide an SBF vocabulary for representing how devices work. The SBF model of a specific device explicitly represents the structural elements and topology of the device, the functions of the device and its components, and the internal causal processes (called behaviors) that explain the workings of the device. The design analogs are indexed by their functions and organized

around design concepts that specify the primitive functions of devices.

The design patterns in Ideal are generic (analog-independent) abstractions of design that the system has encountered in past design episodes. Ideal learns (and thus knows of) two kinds of design patterns: general physical principles and generic teleological mechanisms. A GPP expresses a pattern of causal relations—for example, the causal relations that characterize the principle of heat flow from a hot body to a cold one. A GTM expresses a pattern of functional and causal relations—for example, the functional and causal relations that characterize feedback in control systems. Neither kind of design pattern in Ideal specifies information about the physical structure of devices. Both kinds are expressed in a BF (*behavior function*) subset of the SBF language. Also, both kinds are indexed by the design goals they can help to accomplish.

Figure 3 illustrates a portion of Ideal's computational process that pertains to the learning, reminding, and use of GTMs in analogical design. A design problem in Ideal specifies the functional requirements of the desired device along with behavioral and structural constraints (if any). For example, one design problem presented to Ideal specified the function of an electronic amplifier as the desired function, with the additional requirement that the fluctuations in the output voltage be small. When a design problem is presented to Ideal, in the problem interpretation step, it first uses its conceptual knowledge of generic components and substances to elaborate on the problem specification. In the reminding step, it classifies the elaborated problem specification into its conceptually organized memory of design analogs and retrieves the best-matching analog.

This classification relies on the *functional similarity* between the design problem and the stored analogs. In the example of the amplifier, if Ideal knows of an amplifier design that allows large voltage fluctuations, it retrieves this design because of its functional similarity with the desired design. Next, in the design adaptation phase, Ideal compares the function of the desired design and the function delivered by the retrieved design, and spawns adaptation goals in the form of differences between the desired function and the function delivered by the retrieved design. In the amplifier example, Ideal forms the adaptation goal of reducing the voltage fluctuations in the retrieved design.

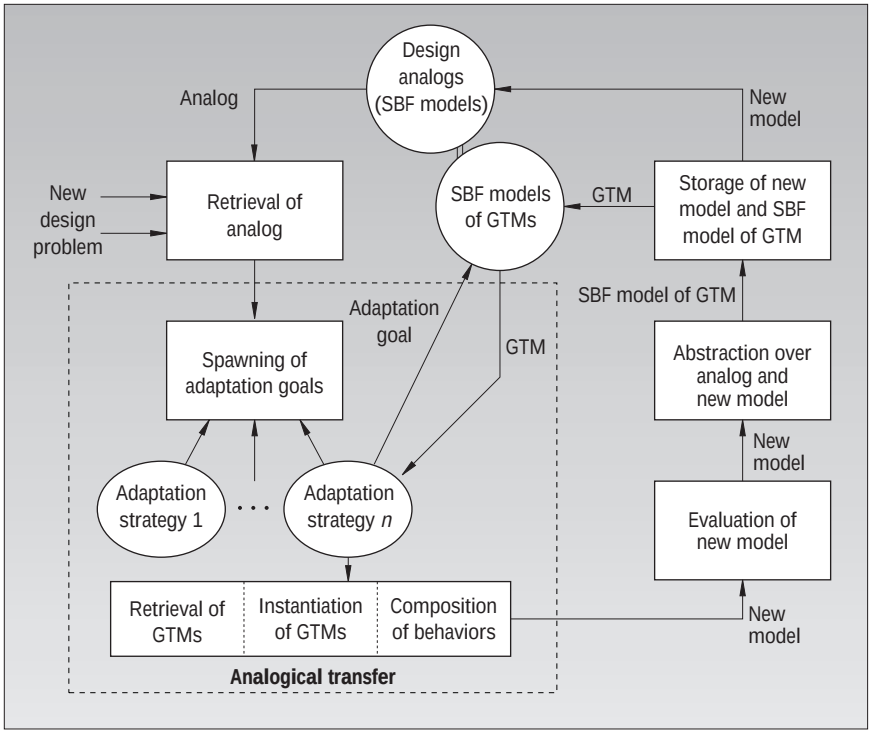

Figure 3. Model-based analogy in Ideal.

The left side of Figure 4 schematically depicts the amplifier design retrieved by Ideal. It also illustrates a small fraction of its SBF model for the design. In particular, it illustrates the top level of its understanding of the causal behavior of the design that results in the achievement of its function.

Briefly, a causal behavior is expressed as a directed graph of behavioral states and state transitions. It expresses each state as a schema that specifies a substance flowing between components, its properties at a specific component, and the property values. The system also expresses each state transition as a schema that specifies the causes for the transition (for example, the function of a component) and relations between the property values in the preceding and succeeding states.

To understand the use of analogies in Ideal, consider what happens if the system does not initially know about control mechanisms such as feedback and feedforward. In this knowledge condition, the system searches the SBF model of the retrieved design but *fails* to localize the cause of large fluctuations in the output voltage.

Suppose that at this stage an (external) oracle provides Ideal with the design for the desired amplifier along with its SBF model. The right side of Figure 4 illustrates this design and a fraction of a causal behavior in its SBF model. The system stores the new design and its model in its analog memory. But it also compares the SBF models of the old design (that allows large voltage fluctuations) and the new design (that regulates the voltage fluctuations). In particular, it inspects the structure of the causal behaviors in the SBF models of the two designs, notes both the similarity and the differences, and abstracts a causal pattern corresponding to the difference in the two causal structures. In addition, it abstracts the adaptation goal (in the form of reducing a functional difference) that it had earlier failed to achieve, uses this functional abstraction as an index to the abstracted causal pattern, and stores the new functional and causal pattern in its memory of GTMs. Figure 5 illustrates the GTM learned by Ideal from the amplifier example. In particular, Figure 5a illustrates the specification of the generic functional difference of reducing large fluctuations in the output of a device, and Figure 5b illustrates the specification of the abstract causal pattern that reduces this kind of generic functional difference.

The system expresses the generic functional difference as a schema that specifies a desired function (`F2`), and a known function (`F1`), and relations between `F1` and `F2`. It expresses each function in terms of the behavioral state it takes as input (**`Given`**) and gives as output (**`Makes`**), and expresses each state in terms of a substance (`?SUB1`), its properties (for example, `?prop1`), and their values (for example, `val21` and `val22`). The Condition slot specifies the

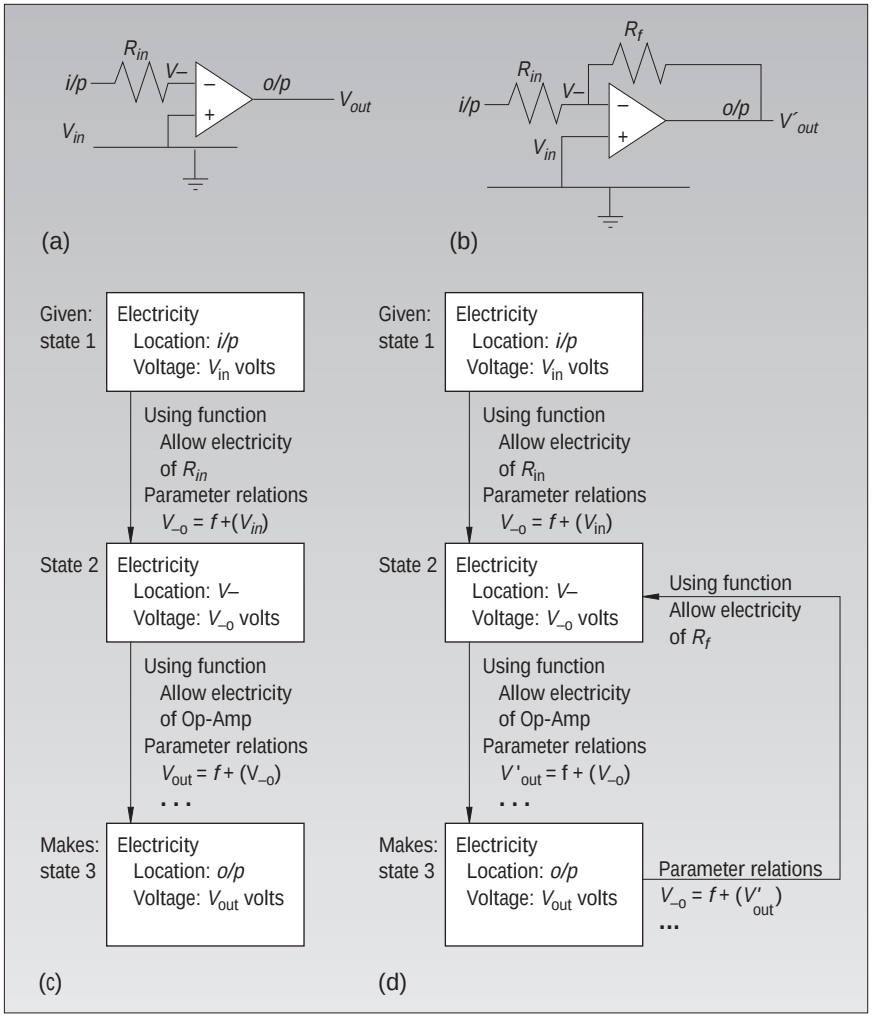

Figure 4. Mapping between behavioral models in Ideal: (a) a simple amplifier; (b) an inverting amplifier with op-amp; (c) behavior "amplify electricity" of the simple amplifier; and (d) behavior "controlled amplify signal" of the inverting amplifier with op-amp.

relationship between `F1` and `F2`. It specifies, for example, that `val21` fluctuates by a small amount (little delta) whereas `val22` fluctuates by a different and larger amount (big delta). It also specifies that `F2` is related to `F1` through some hypothetical function `f`, where `f` is a map to an intermediate value (`?val11'`). The system expresses the abstract causal pattern as a directed graph of behavioral states and state transitions that specifies that the behavior `B2` for accomplishing `F2` can be achieved by a particular combination of the behavior `B1` for accomplishing `F1` and a behavior `B22` that accomplishes the function `f`. Behavior `B22` is constrained by a specific relationship between the states in `B22` and `B1`, and `B2` combines `B1` and `B22` in a specific pattern of causal interaction.

What happens when Ideal is given a new problem in a different domain? For example, one design problem actually given to the system at this stage specified the function of a gyroscope with the requirement that the fluctuations in the output angular momentum be below a specified limit. As discussed, the system first elaborates the problem specification, next classifies it into the primitive device functions, and then retrieves the best matching analog. In the gyroscope example, it retrieves the design of a gyroscope that allows large fluctuations in the output angular momentum. As discussed earlier, the system compares the functions of the desired and the retrieved designs, and spawns adaptation goals. Again, it searches the SBF model of the retrieved design of the gyroscope, and again it fails to localize the cause for the functional difference. But then Ideal abstracts the adaptation goal and uses this abstraction as a probe into its memory of GTMs. It accesses the GTM it had learned in the previous design episode (and illustrated in Figure 5). Next, it instantiates the retrieved GTM in the context of the retrieved gyroscope design to generate an SBF model of a candidate design for the desired gyroscope.

At this processing stage, the candidate solution is only an initial conceptual design. Ideal further refines, evaluates, and completes the conceptual design. For example, it evaluates the candidate design for the gyroscope by tracing through the causal behaviors in the SBF model of the design, ensuring that the behaviors are internally consistent and lead to the achievement of the function desired of it. In this way, Ideal succeeds in generating a candidate design for the new design problem through analogical transfer of design patterns acquired from earlier design episodes.

This short description of Ideal does not cover many aspects of its processing. For example, Ideal also uses design patterns for assimilating information on a design failure and reinterpreting the design problem.[19] In addition, it refines the abstracted design patterns in subsequent design episodes. Ideal uses generic design patterns for cross-domain analogical transfer, which introduces new variables into a design problem space—for example, control variables in the design of the gyroscope. Thus, given our characterization of creative design, Ideal, like Dssua, can be viewed as an analogy-based creative design system.

AI RESEARCH ON ANALOGY-BASED creative design is still in its early stages. The three theories I've just described illustrate the current state of the art. They also indicate some issues, themes, and directions for AI in design research. We can again organize the research issues around the four questions we raised earlier.

- *Why?* Syn, Dssua, and Ideal all use analogies for completing a partial solution to a design problem; in Ideal, the partial solution might be null. Ideal also uses analogies for interpreting the evaluation of a design. For what other design tasks might analogical reasoning be especially useful? It seems analogies are useful for most any conceptual design task, including interpreting the design problem, decomposing the problem, generating a solution to a problem or subproblem, composing subproblem solutions into a candidate design, anticipating potential difficulties with a

candidate design, evaluating a candidate design, interpreting the evaluation information, and reformulating the design problem. Would Syn, Dssua, and Ideal's theories also work for these design tasks—for example, evaluating a candidate design? If not, why not? What additional issues arise as we build AI theories about the use of analogies for a range of design tasks in a variety of design domains?

- *What?* Analogical transfer in Syn, Dssua, and Ideal is mediated by generic design abstractions in the form of design patterns, although the descriptions of the three systems use different terms for design patterns. Syn uses topological patterns, Dssua uses behavioral patterns that retain information about design topologies, and Ideal uses functional and causal patterns with no topological information. In part, these differences reflect the design domains in which the three systems operate. What other kinds of design patterns might be productive for use in analogical transfer? What is the relationship between design domains and design patterns? Other than design patterns, what kinds of generic design abstractions might be productive for analogical transfer? What is the relationship between design tasks and different kinds of generic design abstractions?
- *How?* Syn, Dssua, and Ideal use model-based methods for analogical transfer. Syn uses topological models exclusively, while Dssua and Ideal's models also specify functional, behavioral, and causal knowledge. This difference again reflects the different design domains in which the systems operate. In Dssua's FBS models, the behavior graphs differ from the structure graphs only in that the former also specify the direction of causal influence. In Ideal's SBF models, the representation of causal behaviors, although related to the representation of the device structure, is very different. What are the appropriate knowledge representations for enabling the learning, reminding and transfer of generic design abstractions? What is the relationship between the knowledge representations, the inference they enable, and the design tasks they address? What other methods might enable analogical transfer for different design tasks and domains? In the reminding phase, Syn relies exclusively on structural similarity between the design problem and stored design cases to retrieve a best-matching case. Dssua, too, focuses on structural similarity (although it also seems capable of using functional similarity). In contrast, Ideal uses functional similarity between the design problem and the stored analogs as the primary criterion for retrieval (although it, too, uses structural similarity as a secondary criterion for analog selection). Once again, these differences in part reflect the design domains of the different systems. What are the trade-offs in using functional and structural notions of similarity? What other measures of similarity might be useful in analogical design? What is the relationship between similarity measures, design domains, and design tasks?
- *When?* While Syn assumes the topological patterns as given, both Ideal and Dssua learn their respective design patterns. In Ideal, design patterns are generated at storage time, while in Dssua they are created at problem-solving time. Ideal is an example of an eager learner, while Dssua is an example of a lazy learner. In general, design abstractions can also be learned at retrieval time, which would also make for a lazy learner. What are the trade-offs between eager and lazy learning in analogical design? How do these trade-offs change with the design task, the design domain, and the content of design abstractions? What other kinds of strategic control might be productive for analogy-based creative design?

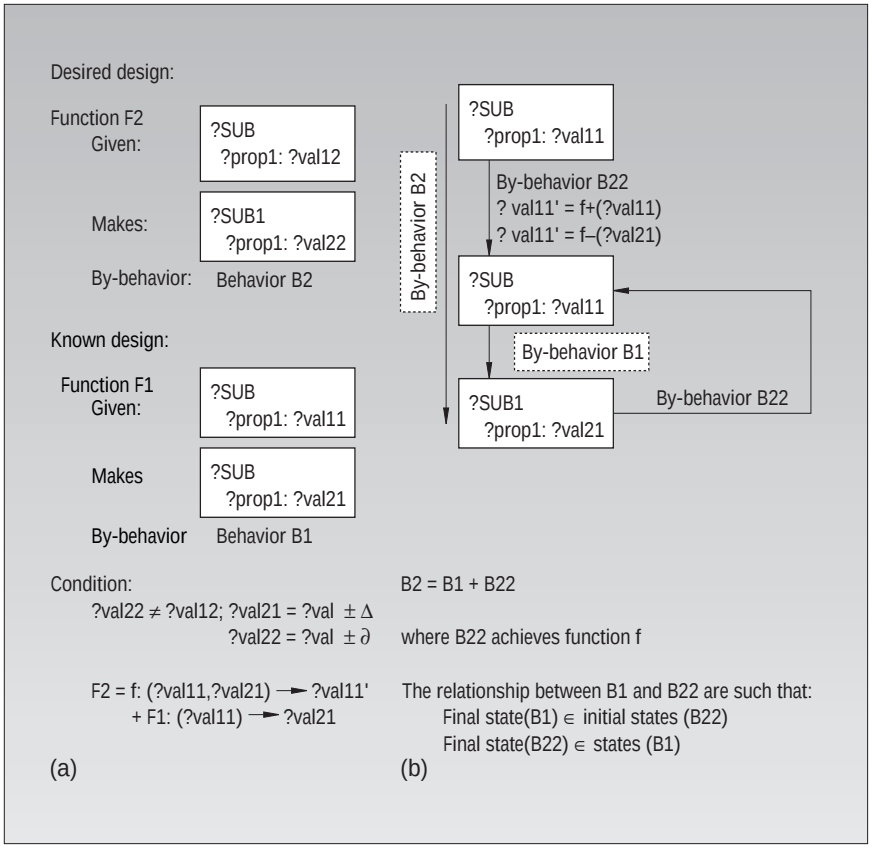

Figure 5. The generic abstraction of feedback in Ideal: (a) functional difference that the feedback mechanism reduces; (b) behavior modification that the feedback mechanism suggests.

This list of issues, of course, is only illustrative, not exhaustive, but even this partial list makes for an ambitious and exciting research agenda.

Acknowledgments

This work has benefited from many discussions with members of the Intelligence and Design Research Group at Georgia Tech, especially with Sambasiva Bhatta. Bhatta, William Murdock, and Todd Griffith helped prepare this article. The article has benefited from constructive critiques by David Brown, William Birmingham, and four anonymous reviewers. I adapted Figures 1 and 2 in this article from "A Design Support System Using Analogy;"[2] Figures 3, 4, and 5 have been adapted from "From Design Expereinces to Generic Mechanisms: Model-Based Learning in Analogical Design."[17] During the writing of this article, design-related research grants from NSF (DMI-94-20405) and DARPA (F33615-93-1-1338) supported my work.

References

1. K. Börner et al., "Structural Similarity and Adaptation," *Proc. Third European Workshop Case-Based Reasoning*, Springer-Verlag, New York, pp. 58–75.
2. L. Qian and J. Gero, "A Design Support System Using Analogy," *Proc. Second Int'l Conf. AI in Design*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992, pp. 795– 813.
3. S. Bhatta, *Model-Based Analogy in Innovative Device Design*, doctoral dissertation, Georgia Inst. of Tech., College of Computing, Atlanta, 1995.
4. A. Newell and H. Simon, *Human Problem Solving*, Prentice-Hall, Upper Saddle River, N.J., 1972.
5. J. Gero, "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, Vol. 11, No. 4, 1990, pp. 26–36.
6. D.C. Brown and B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies*, Pitman, London, 1989.
7. D.C. Brown, "Routineness Revisited," in *Mechanical Design: Theory and Methodology*, M. Waldron and K. Waldron eds., Springer-Verlag, 1996, pp. 195–208.
8. J. Peterson, K. Makesh, and A. Goel, "Situating Natural Language Understanding in Experience-Based Design," *Human-Computer Studies*, Vol. 41, 1994, pp. 881–913.
9. G. Pahl and W. Beitz. *Engineering Design*, Springer-Verlag, 1984.
10. J. Carbonell, "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell, eds., Tioga, Palo Alto, Calif., 1983.
11. M.L. Maher and A. Gomez de Silva Garza, "Case-Based Reasoning in Design," *IEEE Expert*, Vol. 12, No. 2, Mar./Apr. 1997, pp. 34–42.
12. A. Goel, "Integrating Case-Based and Model-Based Reasoning: A Computational Model of Design Problem Solving," *AI Magazine*, Vol. 13, No. 2, Summer 1992, pp. 50–54.
13. L. Wills and J. Kolodner, "Towards More Creative Case-Based Design Systems," in *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, D. Leake, ed., AAAI/MIT Press, Menlo Park, Calif., 1996, pp. 81–91.
14. S. Vosniadou and A. Ortony, *Similarity and Analogical Reasoning*, Cambridge Univ. Press, Cambridge, UK, 1989.
15. P. Winston, "Learning and Reasoning by Analogy," *Comm. ACM*, Vol. 23, No. 12, Dec. 1980, pp. 689–703.
16. M. Gick and K. Holyoak, "Schema Induction and Analogical Transfer," *Cognitive Psychology*, Vol. 15, 1983, pp. 1–38.
17. B. Falkenhainer, K. Forbus, and D. Gentner, "The Structure-Mapping Engine: Algorithms and Examples," *Artificial Intelligence*, Vol. 41, 1989, pp. 1–63.
18. S. Bhatta and A. Goel, "From Design Experiences to Generic Mechanisms: Model-Based Learning in Analogical Design," *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing*, special issue on machine learning in design, Vol. 10, 1996, pp. 131–136.
19. S. Bhatta, A. Goel, and S. Prabhakar, "Innovation in Analogical Design: A Model-Based Approach," *Proc. Third Int'l Conf. AI in Design*, Kluwer, 1994, pp. 57–74.

Ashok K. Goel is as an associate professor of computer and cognitive sciences with the College of Computing at the Georgia Institute of Technology. His current research investigates adaptive design, functional representations and reasoning, analogical reasoning and learning, creative design, and model-based reflection and self-adaptation. He received his PhD in computer and information science from Ohio State University. Since 1990, Goel has chaired the IEEE SMC Technical Committee on AI. This year, he is a vice-chair of the Fourth International Conference on AI in Design and a member of the Program Committee of the National Conference on AI. Contact him at the College of Computing, Georgia Inst. of Tech., Atlanta, GA 30332-0280; goel@cc.gatech.edu; http://www.cc.gatech.edu/aimosaic/faculty/goel.html.