CHAPTER 12

2

The Formation and Use of Abstract Concepts in Design

YORAM REICH
STEVEN J. FENVES

1. Introduction

Design is an ill-structured problem, in which there are often vague criteria for 'success', the search space is ill-defined, and resources (e.g., time) are limited. Building expert systems for design, henceforth called design systems, requires relevant domain knowledge and representation schemes that support the solution of this inherently ill-structured problem. However, manual knowledge acquisition has been a bottleneck in creating expert systems in general and design systems more specifically. One approach for alleviating some of the difficulties of knowledge acquisition is the introduction of learning into the development and maintenance stages of expert systems (Reich & Fenves, 1989; Steels & Van de Velde, 1986). Learning can assist in automatically building design systems as well as in understanding a specific design domain.

The principal emphasis of this chapter is on the acquisition of design knowledge from a *tabula rasa* stage to an expert-level stage. One promising solution uses *similarity-based learning* (SBL) methods of supervised and unsupervised concept learning. This chapter examines the relationships between these learning strategies and design. Section 2 begins this process by identifying design characteristics that constrain the acquisition of design knowledge. Section 3 shows that supervised concept learning provides limited support for acquiring design knowledge, and illustrates that unsupervised methods of concept formation may be more suitable to the task. Section 4 describes BRIDGER, a system that integrates concept formation into design. Section 5 evaluates BRIDGER in a number of domains, and provides a framework for understanding the relationship between design and concept formation.

2. The Nature of Design

The product of design is the description of an artifact (called the design description) that satisfies a set of specifications. The process of mapping from specifications to design descriptions is called *synthesis*. The process of mapping from design descriptions to behavior characteristics (which are essentially similar to the specification properties) is called *analysis* or *evaluation*.

Synthesis is difficult since its complexity is exponential in the number of properties describing designs. This complexity results from coupling between design properties, which often prevents good evaluations of partial designs. In contrast, behavioral analysis is relatively simple, since each behavior is directly dependent upon the design and is largely uncoupled from the remaining behaviors.

This chapter focuses on the acquisition of knowledge for the following synthesis task: given a (possibly partial) specification, find a set of design description properties that satisfy the specification. In learning, this process corresponds to *prediction* of missing properties (i.e., the design description) from the partial description (i.e., the specification). The above task is suitable for the first stages of *synthesis* in *conceptual design*, namely, the retrieval, assembly, and manipulation of relevant past solutions that can serve as a basis for a new design solution. However, before defining mechanisms for acquiring design knowledge, it is necessary to understand some of the important characteristics

of design. Usually, these characteristics are only implicit in common, general-purpose definitions of design. Instead of trying to define design precisely, it is useful to view it as an *ill-defined* or *ill-structured* problem (Reitman, 1964).

2.1 Design as an Ill-Structured Problem

Design is ill-structured (Simon, 1973), meaning that it lacks at least one of the following characteristics: (1) there is a definite criterion for testing solutions; (2) there is a problem space that contains the knowledge required to solve the problem (i.e., representation of states and operators) and that can include any newly acquired knowledge about the problem; and (3) all the processes for solving the problem can be performed in a reasonable time. A well-structured problem possesses all these characteristics; design, on the other hand, lacks all three. Let us address each characteristic in turn.

1. Test criteria. Design problems are almost always underspecified and loosely constrained. Not only is a definite testing criterion lacking in design, there is also no criterion for determining when the design process should terminate. In addition, there is never a single objective in design; rather, many objectives are sought, for example, minimizing cost and space and maximizing effectiveness. When multiple objectives compete, there is no clear criterion for selecting among alternatives. Usually, some compromise is made to the advantage of a subset of the objectives. In most design problems there are additional criteria that cannot be quantified, such as aesthetics.

2. Knowledge representation. The solution of design problems requires the use of diverse knowledge sources from several disciplines. Each of the knowledge sources can raise different concerns regarding the design, which might give rise to multiple solution strategies. Practically speaking, it is impossible to capture this diversity *a priori* in a closed, static problem space. Rather, the problem space evolves dynamically as the solution progresses.

3. Resources. Design processes can only be allocated limited resources for their completion. Examples include time limits that stem from due dates for design completion, the total man-hours that can be spent, and the available design aids. Small problems can be solved

within the available resources, and their solution might even be optimal with respect to certain criteria. Real design problems, however, rarely achieve optimality. The inability to optimize, and therefore the need to satisfice (Simon, 1969), and possibly compromise, is a direct consequence of design complexities in conjunction with limited resources.

There is another aspect of design that is usually neglected, but which reinforces the characteristics described above. This is that design knowledge is constantly evolving. Designers not only must cope with a complex task, but they must track the evolution of a domain. In this situation, designers determine whether new knowledge is related to the body of existing knowledge, or whether the new knowledge reflects a more fundamental change in technology; the latter may have a large effect on their problem-solving behavior.

An exemplar of theory change in bridge technology is the introduction of aerodynamic considerations due to the failure of the Tacoma Narrows Bridge in 1944. The phenomenon of dynamic instability due to wind was not widely known before that event. As a result, research and evaluation of existing bridges has proliferated, leading to drastic strengthening of some bridges such as the Golden Gate Bridge. Such transparent changes are rapidly assimilated. Less obvious changes are harder to accommodate.

2.2 General Requirements for Learning

Learning from experience allows designers to improve their management of ill-structured design problems. The designers' particular experience determines their style or the organizational structure of their problem-solving behavior. These same adaptive mechanisms are required by tools that partially automate the acquisition of design knowledge. A selected learning method must support the full range of design activities that we discussed earlier.

1. Test criteria. While there is no definite criterion for testing whether a design process ends, there are many ways to test whether a design conforms to a given specification and how it is rated on evaluation scales such as economy and efficiency. Learning should generate knowledge structures that facilitate evaluation in terms of multiple

criteria and objectives. They should also be able to implicitly accommodate subjective test criteria used by different designers.

2. Knowledge representation. Learning should support the acquisition and integration of diverse knowledge sources. The knowledge generated should be available to different tasks such as synthesis, analysis, explanation, and evaluation.

3. Resources. The limited-resource characteristic of design requires that learning gradually produce knowledge that is more efficient to use. This can be achieved by two complementary mechanisms. The first mechanism assimilates new information into an existing knowledge base, and the second reorganizes knowledge based on feedback or self-analysis. The knowledge generated should facilitate various strategies that effectively manage the limited resources available.

Finally, the evolutionary aspect of design should be supported by the use of incremental techniques. In light of the limited resources and the continuous flow of information, batch techniques are not suitable since they need to reprocess all existing knowledge to accommodate any new piece of information.

3. Unsupervised Versus Supervised Concept Learning

This section describes learning mechanisms for acquiring and structuring basic design knowledge in a way that facilitates efficient synthesis. Since the ill-structured nature of design often precludes formalized theories of synthesis, we are tentatively biased against methods like *explanation-based learning* (EBL) (Mitchell et al., 1986; Mooney, this volume) during the early stages of knowledge acquisition. On the other hand, *similarity-based learning* (SBL) techniques do not require background knowledge (although they may use it if available), since they operate on the basis of syntactic similarity. Manual knowledge elicitation techniques are not discussed as we wish to explore the limits of a fully-automated approach, i.e., an approach that requires minimal resources, little interaction with a domain expert, and generates simple and flexible knowledge structures.

The source of information available for automatic learning is a collection of case histories. These cases are the product of articulating design

knowledge, and may contain actual in-service behavior and maintenance data. Data like this are used by human designers to explain failures and to expand knowledge that enhances design practice (Magued et al., 1989). Case histories can also be used by machine learning techniques to produce similar results (Stone & Blockley, 1989). Another reason why the study of case histories is easier than manual expert elicitation is that experts can provide solved problems more readily than they can articulate their knowledge explicitly. This tendency leads to the preference of applying learning methods based to the results of expert problem solving, in the hope of extracting expert rules that underlie the behavior (Michalski & Chilausky, 1981). Two primary classes of machine learning techniques comprise the SBL approach: *supervised concept learning* and *unsupervised concept leaning*.

3.1 Supervised Concept Learning

Supervised concept learning creates a classification that can support a single task of classifying new objects into a set of pre-defined classes. In the case of design, examples are represented by a list of specification property-value pairs and are classified into a set of classes that can represent a *single* design descriptor. The task is to predict this design description property based on the specification properties. Several examples of the use of supervised concept learning in simple design tasks illustrate the potential and limitations of this approach.

Arciszewski et al. (1987) used a supervised concept learning to differentiate between feasible and infeasible designs. Each design is described by a list of discrete property-value pairs: $(d_1 = dv_{11}, d_2 = dv_{21}, \ldots, d_n = dv_{n1}, s_1 = sv_{11})$, where d_i denotes a design description property, dv_{ij} denotes one of the values of d_i, and s_1 has possible values of sv_{11} =*feasible* or sv_{12} =*infeasible* (see Figure 1–1a). The properties d_i reflect qualitative design decisions made at the early stage of design. The goal of generating a rule set is to predict whether a given combination of design description values is feasible or infeasible (Figure 1–1b).

A similar approach is presented by McLaughlin & Gero (1987). Instead of differentiating between feasible and infeasible designs, the task is to characterize designs that are optimal. Each design is described by a list of property-value pairs: $(d_1 = dv_{11}, \ldots, d_n = dv_{n1}, o_1 = ov_{11}, \ldots, o_m = ov_{m1})$, where d_i denotes a design descriptor, o_j denotes

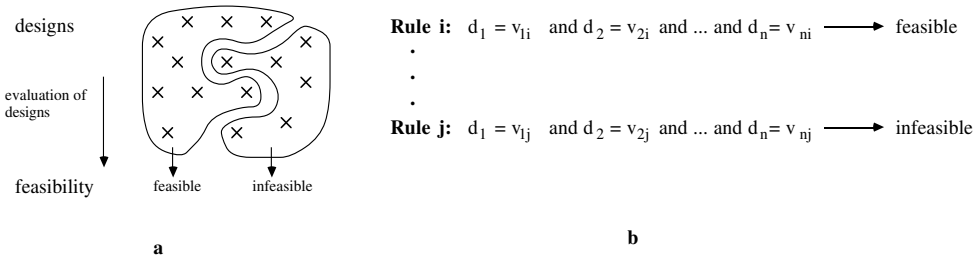

Figure 1–1. Classification of feasible designs

an objective, and dv_{ik} or ov_{ik} denote one of the values of d_i or o_i, respectively. Since a supervised concept learning technique (i.e., ID3; Quinlan, 1986) is used, a single classification of Pareto-optimal and inferior designs is generated for each aggregation of objectives $o_1, \ldots, o_m$. An example of this strategy for two objectives is illustrated in Figure 1–2a. A decision tree generated by the learning program is intended to predict whether a given design will be Pareto-optimal or inferior (Figure 1–2b).

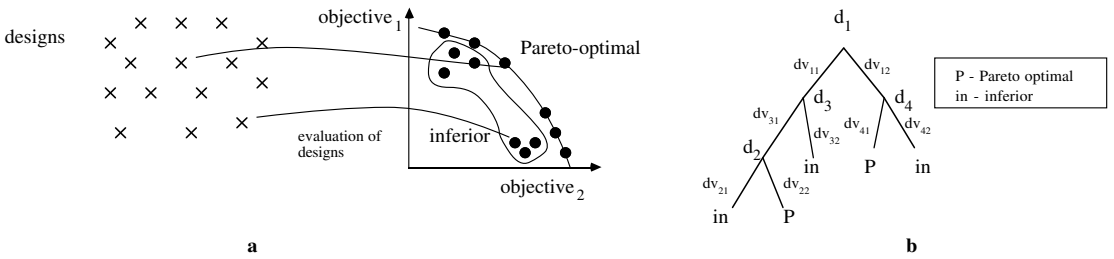

Figure 1–2. Differentiation of Pareto-optimal from inferior designs

These two approaches essentially extract *evaluation* knowledge and not *synthesis* knowledge. One approach assesses whether a design is feasible, while the other prunes inferior designs. Figure 1–3a summarizes this process. Neither of these approaches synthesizes designs from specifications. Rather, they map design descriptors onto behavioral descriptors. However, one can use similar approaches to perform synthesis. First, specification properties can be used to generate a classification over the set of designs (e.g., with 'optimal' and 'inferior' as labels). Concept descriptions in terms of the design description proper-

ties can then be used to characterize subsets of the training data that were distinguished by their specification properties. For example, we might distinguish design classes using a system like ID3 (see Fisher & Pazzani, Chapter 1, this volume), using only specification properties along arcs of the resultant decision tree. At each leaf of the tree we would then associate a pattern, much like in EPAM (see Richman, this volume), that characterized the common design properties of artifacts classified to that leaf. This process captures a *many-to-one* mapping between designs and classes of specifications. Figure 1–3b shows how the classification could be used for a very restricted type of synthesis. In this process, new specifications are classified (e.g., via a decision tree) into a subset of designs (e.g., leaves of a decision tree). The pattern associated with this subset is forwarded as the synthesized design. Unfortunately, it appears that this strategy may yield patterns that are too general for practical purposes.

Figure 1–3. Supervised concept learning and synthesis

A different approach to synthesis using supervised concept learning techniques is to discover rule sets that directly map the specification properties to a single design property (Quinlan, 1989). In this case each design is described by a list $(s_1 = sv_{11}, \ldots, s_m = sv_{m1}, d_1 = dv_{11}, \ldots, d_n = dv_{n1})$, where d_i denotes a design descriptor, s_j denotes a specification property, and sv_{ik} or dv_{ik} denote one of the values of s_i or d_i, respectively. Each design is used to generate n training examples, $(s_1 = sv_{11}, \ldots, s_m = sv_{m1}, d_1 = dv_{11}) \ldots (s_1 = sv_{11}, \ldots, s_m = sv_{m1}, d_n = dv_{n1})$. Over many designs this results in n example sets that generate n decision trees, one for each d_i (see Figure 1–4).

Although some benefits are possible from this procedure , it may lose relationships that exist between the different design descriptors. Designs

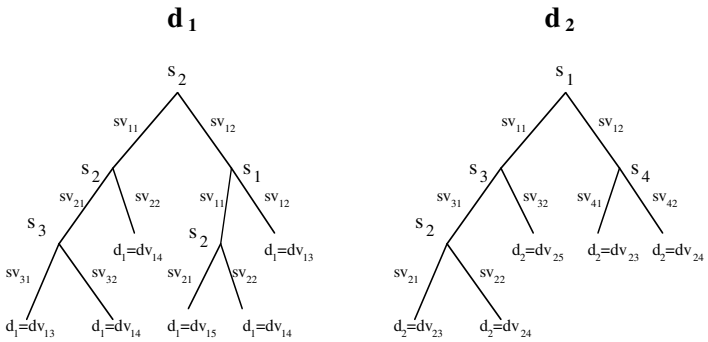

Figure 1–4. Multiple supervised concept learning and synthesis

generated by this approach might all be inadequate, thus requiring redesign. To illustrate, observe the following list of designs:

1. $(s_1 = sv_{11},\ s_2 = sv_{21},\ s_3 = sv_{31},\ d_1 = dv_{11},\ d_2 = dv_{21},\ d_3 = dv_{31})$
2. $(s_1 = sv_{12},\ s_2 = sv_{21},\ s_3 = sv_{32},\ d_1 = dv_{12},\ d_2 = dv_{22},\ d_3 = dv_{32})$
3. $(s_1 = sv_{12},\ s_2 = sv_{22},\ s_3 = sv_{31},\ d_1 = dv_{12},\ d_2 = dv_{21},\ d_3 = dv_{31})$

$\vdots \quad \vdots \qquad\qquad\qquad\qquad \vdots$

that is used to generate three distinct mappings from the specification properties, s_j, to each design property, d_i. If we look at each design property separately, the following predictions for the three objects above might be possible:

1. $d_1 = dv_{11}$, $d_2 = dv_{21}$, $d_3 = dv_{32} \neq dv_{31}$,
2. $d_1 = dv_{12}$, $d_2 = dv_{21} \neq dv_{22}$, $d_3 = dv_{32}$, and
3. $d_1 = dv_{11} \neq dv_{12}$, $d_2 = dv_{21}$, $d_3 = dv_{31}$.

Although the performance level is 66% (i.e., six out of the nine design descriptions are predicted), not one of the designs is completely predicted. If there are important dependencies between design descriptors, then evaluation at a later stage may reveal that one or more of these synthesized (i.e., predicted) designs is infeasible.

To generalize, the inadequacy of supervised concept learning techniques as a means of capturing synthesis knowledge is due to the nature of synthesis. Synthesis is a *many-to-many* mapping (specifications to design description properties). If this mapping is divided into separate *many-to-one* mappings to be handled by supervised concept learning, considerable information may be lost since design descriptor properties

are not independent. Furthermore, synthesis cannot be modeled by a mapping from specifications to classes of designs since such classes are often not known a priori.

3.2 Unsupervised Concept Learning

A learning paradigm concerned with many-to-many mappings is unsupervised concept learning. The approach to synthesis knowledge acquisition by hierarchical (possibly overlapping) unsupervised concept learning is illustrated in Figure 1–5. The principal idea is that specifications and solutions (i.e., design descriptions) are correlated together; specific combinations of specification properties give rise to corresponding combinations of design description properties that satisfy these specifications. A clustering based on this correspondence allows the retrieval of appropriate designs given a new specification similar to an existing one. Ideally, one would form concepts only from acceptable designs, eliminating the need to perform analysis and redesign in the future. In reality, the unsupervised concept learning process is performing an inductive leap (overgeneralization) over the data and cannot guarantee the generation of perfect designs for a new specification. Thus, analysis and redesign stages cannot be eliminated, but ideally the effort expended during these later stages can be reduced.

There is another dimension to synthesis that constrains the type of unsupervised concept learning method that may be used: the output of synthesis is rarely one alternative; usually, several candidates are produced. The unsupervised concept learning method should use a knowledge representation that allows for the generation of multiple alternatives.

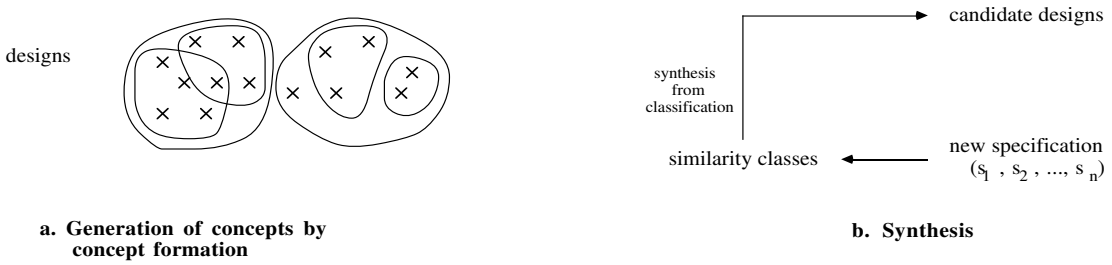

Figure 1–5. Unsupervised concept learning and synthesis

There are two distinct styles of unsupervised learning: batch or nonincremental methods, and incremental, concept formation strategies. Approaches to unsupervised concept learning in design have experimented with both. Although batch processing is not, in general, a valid approach in design, it can provide some insights in the first stages of trying to understand a design domain.

A simple example of using nonincremental clustering is the creation of a metric space (Taura et al., 1989). In this approach, a metric (i.e., distance measure) is used to represent the distance between the function or behavior manifested by designs. The function or behavior is extracted from the artifacts' descriptions by using domain knowledge. A similarity function is used to measure the distance between any two artifacts, arranged in a similarity matrix (see Figure 1–6). Several important dimensions $\lambda_1, \ldots, \lambda_l$ are extracted by performing a principal coordinate analysis over the matrix. The analysis reduces the dimensionality of the description space, which in turn allows a simple clustering. Each cluster is then identified by generalizing the design descriptions of its elements. This description is then used in new situations to construct new objects to satisfy a given functionality expressed by the principal coordinates. Two drawbacks are that this is a batch system and it generates a single-layer classification that may be computationally inefficient for the retrieval of designs that are similar to the new situation. The principles of this approach, however, are similar to those used later in our approach.

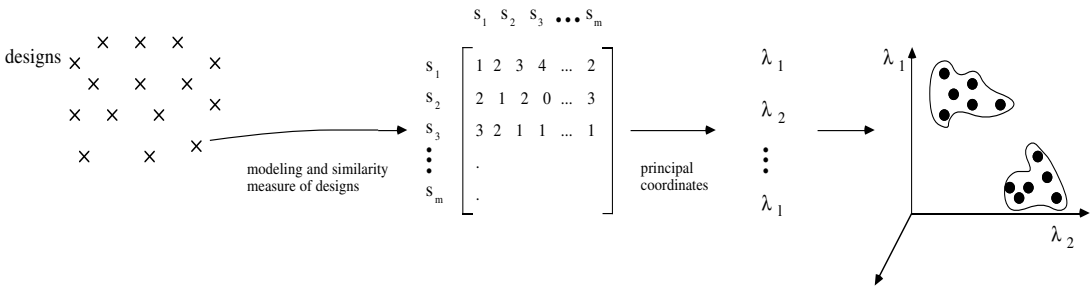

Figure 1–6. Creation of a metric space

A more elaborate form of treating multiple specification properties than those described in the previous section is provided by Lu & Chen (1987). Design examples are described by a list of specification properties and design descriptors ($d_1 = dv_{11}, \ldots, d_n = dv_{n1}, s_1 = sv_{11}, \ldots, s_m =$

sv_{m1}). The specifications are clustered into a finite set of classes by the conceptual clustering program CLUSTER/2 (see Figure 1–7) (Michalski & Stepp, 1983). These clusters serve as the classes that can be used by a supervised concept learning program to generate rules. The product of supervised learning by the program AQ15 (Michalski et al., 1985) are rules that assign the description of a design to the right class, i.e., approximately identify the specification satisfied by the design. Again, these rules constitute evaluation and not synthesis knowledge. Like the previous approaches reviewed, this approach is not incremental; it is also an instance of the class of techniques captured in Figure 1–3.

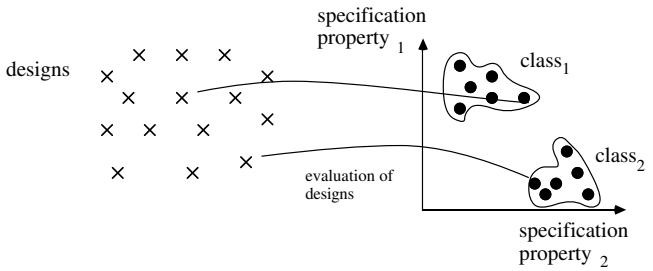

Figure 1–7. Clustering and supervised concept learning

In conclusion, unsupervised concept learning is more suitable than supervised concept learning for the acquisition of design knowledge. This section only reviewed two approaches employing batch learning which, in general, is not optimal for design. The next section describes BRIDGER, a system that incorporates concept formation into design.

4. BRIDGER: An Approach to Concept Formation

This section reviews COBWEB and its application to the generation and use of hierarchical knowledge structures for design. Several of its drawbacks are identified. The section also presents BRIDGER a descendant of COBWEB that addresses these issues.

4.1 COBWEB: An Incremental Concept Formation System

COBWEB is an incremental concept formation program for the creation of hierarchical classification trees (Fisher, 1987). In the context

of design, COBWEB accepts a stream of designs described by a list of property-value pairs. Designs need not be classified as feasible, optimal, or by any other classification scheme. However, any *a priori* classification can be assigned to a design and treated as any other property (Anderson & Matessa, this volume).

A classification is 'good' if the description of a design can be guessed with high accuracy, given that it belongs to a specific class. COBWEB makes use of a statistical function that partitions a set of designs into mutually exclusive classes $C_1, C_2, \ldots, C_n$ (originally proposed by Gluck & Corter, 1985). The function used by COBWEB is category utility (CU):

$$\frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 \; - \; \sum_i \sum_j P(A_i = V_{ij})^2}{n} \qquad (1)$$

where C_k is a class, $A_i = V_{ij}$ is a property-value pair, $P(x)$ is the probability of x, and n is the number of classes. The first term in the numerator measures the expected number of property-value pairs that can be guessed correctly by using the classification. The second term measures the same quantity *without* using the classes. Thus, the category utility measures the *increase* of property-value pairs that can be guessed *above* the guess based on frequency alone. The measurement is normalized with respect to the number of classes.

COBWEB builds the classification hierarchy in the following way. When a new design is introduced, COBWEB tries to accommodate it into the existing hierarchy starting from its top class. COBWEB can perform one of the following operators:

1. expanding the root, if it does not have any sub-classes, by creating a new class and attaching the root and the new design as its sub-classes;
2. adding the new design as a new sub-class of the root;
3. adding the new design to one of the sub-classes of the root;
4. merging the two best sub-classes and putting the new design into the merged sub-class; or
5. splitting the best sub-class and again considering all the alternatives.

If the design has been assimilated into an existing sub-class, the process recurses with this class as the top of a new hierarchy. COBWEB again uses the utility function CU to determine the next operator to apply.

COBWEB designs using a mechanism similar to the one used for augmenting the hierarchy by new designs but allowing only operator 3 to apply. COBWEB sorts the new specification through the hierarchy to find the best host for the new specification. The best host is a leaf node (i.e., an example of an existing design) that is used to complete the description of the new design. This is a very conservative strategy, as it allows only the exact duplication of existing designs.

To illustrate, let us examine how COBWEB designs a new bridge using a hierarchy generated from 60 examples of bridges from the Pittsburgh bridge database. In this domain, a bridge is described using 12 properties of which seven are specification properties (three continuous[1] and four nominal): the *river* and *location* of the bridge, the *period* it was constructed, the *purpose* of the bridge, the *number of lanes*, the *length* of the crossing, and whether a *vertical clearance* constraint was enforced in the design. Five design properties are provided for each design: the *material* used, the *span* of the bridge, the *type* of bridge, the location of the roadway on the bridge (*T-or-D*, meaning through or deck), and the relative length of the span to the crossing length (*Rel-L*).

Figure 1–8 shows the hierarchy generated by COBWEB after learning 60 examples ordered by the date of their construction. The nodes are described using their characteristic properties only. Characteristics are property values that satisfy: $P(A_i = V_{ij}|C_k) \geq threshold$ and $P(C_k|A_i = V_{ij}) \geq threshold$, where *threshold* is a pre-determined fixed value[2] The reference symbol and the number of examples below each node (in parentheses) are also provided. Assume that COBWEB is required to design a *highway* bridge on one of the rivers where *vertical clearance governs*. The following description is a high level interpretation of the prediction behavior of COBWEB. The system starts designing by classifying the known specification and design constraints down the

1. For the purpose of using COBWEB, these properties were discretized. Later, we relax this restriction.
2. Potentially, this value can be learned for each domain. In the experiments described later, this value is fixed to 0.75. Other definitions of characteristic or normative values appear in (Fisher, 1987; Kolodner, 1983; Fisher & Pazzani, Chapter 1, this volume).

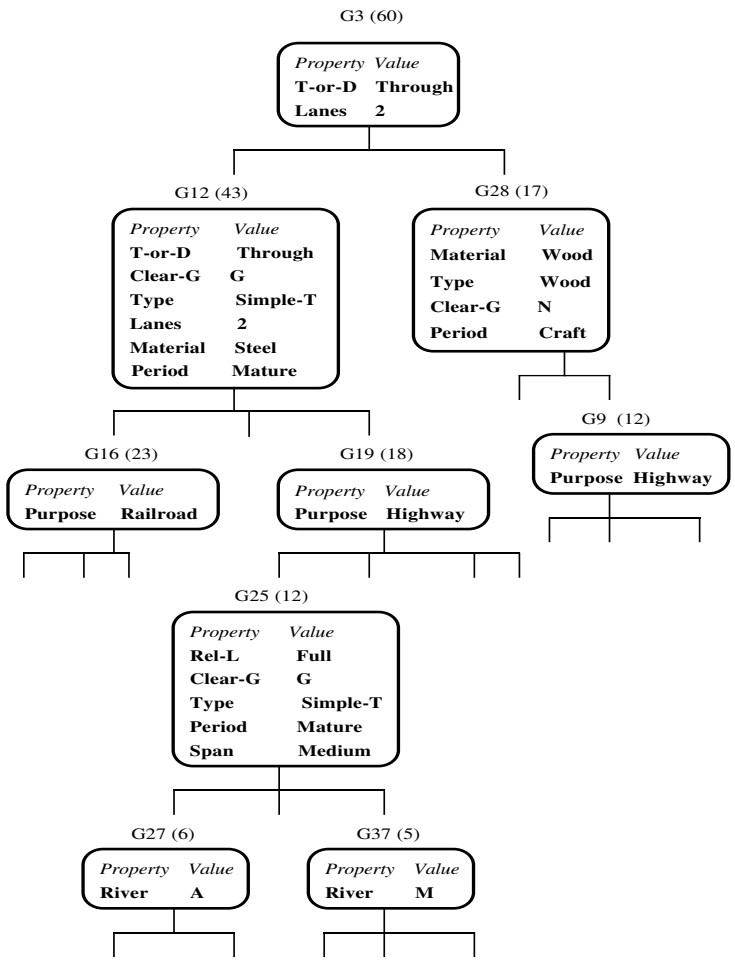

Figure 1–8. Design concepts hierarchy after learning 60 examples

hierarchy. Since vertical clearance governs, the design description is refined using class G12, characterized by clearance governing. Class G19 is chosen next, since it represents *highway* bridges. At this stage the design should terminate and candidate designs (e.g., descendants of G19) should be returned, since all the known specifications have been met. However, the leaf prediction method that we spoke of attempts to find a single leaf node; thus, the design process continues.

Cobweb partially supports the requirements presented in Section 2.2. It allows a flexible use of the hierarchy; any type of specification, partial or complete, can be used to retrieve an appropriate design. However, the system has several drawbacks.

1. Cobweb has a stiff prediction/design scheme. It makes a strong commitment to continue the design process until a complete existing design is retrieved. No generation of new designs or accommodation of subjective judgment is allowed. In addition, leaf prediction is not adequate since it produces only one candidate for each specification. Leaf prediction is also susceptible to noise and may result in unnecessarily high error rates.

2. Cobweb can only manipulate nominal properties. Classit is a descendant of Cobweb that handles only continuous properties (Gennari et al., 1989). Elsewhere, we contrast its approach with the extension to continuous properties implemented in Bridger (Reich, 1990c) and conclude that Classit's strategy is less "natural" and considerably more limited.

3. Cobweb uses only the syntactic measure of CU to guide its learning and prediction. No domain dependent or independent knowledge is used, although if available, this could enhance learning substantially.

4. Finally, performance is dependent upon the order of example presentation. This dependency manifests itself even though two of Cobweb's operators are designed to reduce order effects.

We now turn to an extension of the basic hierarchical classification strategy, which is intended to mitigate the problems that we have listed above, thus yielding a more robust design tool.

4.2 BRIDGER: An Extension of COBWEB for Design

BRIDGER is designed to extend the applicability of the COBWEB approach to design (Reich, 1990b; Reich, 1990c; Reich & Fenves, 1990). Although BRIDGER implements other design tasks such as analysis and redesign (Reich, in press), this chapter concentrates on the synthesis part which will henceforth be referred to as BRIDGER.

4.2.1 PREDICTION SCHEMES

BRIDGER can use several prediction/design methods. We illustrate our system's prediction strategy using Figure 1–8 again.[3] In the same design problem as before, BRIDGER starts designing by classifying the specifications down the hierarchy. The top node of BRIDGER's knowledge represents *through* bridges with *2* lanes. These property-values are assigned to the new design: the first property is a design refinement and the second property is a specification elaboration. Since vertical clearance governs the design, the design description is again refined using class G12. The new design will be a *simple-truss* made of *steel*. Class G19 is chosen next, since it matches the given *highway* specification. At this stage the design terminates, since all the specifications have been met, but the design is only partially specified since the main span and the relative length of the bridge have not been determined. This is expected, since the specifications are abstract. BRIDGER can use refinement strategies to complete the design or retain the abstract design as the solution (Reich, 1990b). For example, BRIDGER can deliver the set of 18 bridges under class G19 as design candidates in a pure case-based approach; it can generate a new prototype from the most frequent property-value pairs in G19 and deliver it with a set of possible variations as in a prototype-based design; or it can deliver a large set of candidates generated from the combination of all the property-value pairs of the 18 designs. In summary, the path traversed during CU-guided classification progressively matches known specifications, and augments the design that is intended to support these specifications.

3. The hierarchy generated by BRIDGER while using continuous properties is slightly different.

4.2.2 Continuous Properties and Constructive Induction

Bridger implements an extension of CU that can handle continuous properties. In its simplest variant the term $\sum_j P(A_i = V_{ij}|C_k)^2$ is calculated as:[4]

$$\sum_j P(A_i = V_{ij}|C_k)^2 \quad \Leftrightarrow \quad P(A_i = \overline{V_i}|C_k)^2 \quad \Leftrightarrow \quad \left(\int_{-\frac{d_i}{\sigma_i}}^{+\frac{d_i}{\sigma_i}} p_i(x)dx \right)^2 \quad (2)$$

where

$$p_i(x) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-(x-\overline{V_i})^2/2\sigma_i^2},$$

$$2d_i = \frac{expected\ range\ of\ property\ values\ of\ A_i}{expected\ number\ of\ distinct\ intervals\ of\ property\ A_i},$$

$\overline{V_i}$ is the mean of the values of A_i in C_k, and σ_i is the standard deviation of the values of A_i. This has been tested extensively in several domains and has proven to be effective (Reich, 1989; Reich, 1990c).

A second extension is based on identifying a mechanism that groups property values into higher-order features. In this extension, continuous property-values are grouped into samples from an assumed distribution, such as the normal distribution used above. In the same way, nominal properties can be grouped to result in a constructive induction ability (Reich, 1990a; Thompson & Langley, this volume). For example two complementing values V_{11} and V_{12} can be combined into a new feature $G_1 = \{V_{11}, V_{12}\}$. This can be viewed as adding an internal disjunct into the description language. Additional values can be accommodated into or deleted from G_1 based on additional information.

Equation 1 is modified to work with such features in the following way:

$$\frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = G_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = G_{ij})^2}{n}; \quad (3)$$

where $G_{ij} = \{V_{ijl}|V_{ijl} \in Dom(A_i)\}$ are new features generated by the constructive induction such that: $\bigcup_j G_{ij} = Dom(A_i)$, $\bigcap_j G_{ij} = \phi$, and $Dom(A_i)$ is the domain of values of A_i. In addition, $P(A_i = G_{ij}) = \sum_l P(A_i = V_{ijl})$, $V_{ijl} \in G_{ij}$.

4. A detailed development of this extension appears in (Reich, 1990c).

Features are initialized to ground properties (i.e., individual property values). The constructive induction operates as an integral part of calculating CU, hence it tries to modify the features each time CU is calculated. The combination of two values is performed only if it increases CU. In principle, the best approach is to try all the combinations of values each time CU is calculated. However, this process is exponential in the number of values. Another approach uses the new example as a trigger for the constructive induction scheme and only allows for small changes in the current features to occur. The features of each of the properties in the new example description can undergo the following changes:

1. the system can leave the features unchanged;

2. the feature that contains the property-value of the new example can be merged with another feature; or

3. the feature containing the property-value of the new example can be divided in one of two ways: (1) divide the feature into two features, one containing the new property-value and the other the remaining values (2) divide the feature into features containing one value each.

4.2.3 SUMMARY

BRIDGER builds on earlier concept formation systems like COBWEB and UNIMEM (i.e., in its use of characteristic values), notably in its exploitation of constructive induction and continuous values. We have experimented with BRIDGER in a number of standard domains, such as the breast cancer domain and the mushrooms domain (Reich, 1990).[5] In these cases, BRIDGER performs comparably, and in many cases better than a number of well-known supervised and unsupervised systems. However, the primary goal of this chapter is to investigate the use of concept formation and design. Thus, we will only detail our evaluation of BRIDGER in design domains.

5. Both domains are available from the repository of machine learning databases at the University of California at Irvine.

5. Evaluation of BRIDGER

BRIDGER's approach to design is similar to the General Design Theory (Yoshikawa, 1981); designs are represented by property-value pairs, where properties describe an artifact. This approach is amenable to preliminary or conceptual design, where major design decisions are made without good quantitative measures of their tradeoffs. In particular, this approach is appropriate to our target domain of bridge design. This approach is different from common approaches to VLSI design (e.g., top-down refinement plus constraint satisfaction (Steinberg, 1988), where design subgoals are matched against components or further decomposed), or some similar mechanical design problems (Ramachandran et al., 1988), where design decisions are tightly coupled.

5.1 Synthesis in Bridge Design

BRIDGER's performance was evaluated using the database of Pittsburgh bridge descriptions that we have been using throughout the chapter for illustrative purposes. Figure 1–9 shows a typical[6] graph of BRIDGER's performance on the Pittsburgh's bridge database. Figure 1–9a describes the performance in a *coverage* experiment. In this experiment the test cases include *all* the examples in the data set. The results show correct prediction accuracy averaged over the five design properties and 50 runs of randomly ordered examples. The achievement of a 78% coverage suggests that the example set is not sufficient to "cover" the domain. Figure 1–9b shows the performance in a transfer experiment where the test examples are only the *unseen* designs. Again the results are averaged over 50 runs. Gradual, but slow, improvement in transfer of knowledge about individual design properties is observed.

Since the database of bridges does not have many examples, the hierarchy generated is not deep enough to allow the emergence of characteristic values for the specification properties. This impedes prediction at an internal node, which we have already noted is a preferred strategy. Thus, in fact, all the predictions of this study are based on leaf nodes. We suspect that this prevents BRIDGER from producing better results.

6. There are several parameters that can be adjusted in the prediction method. For example, the threshold for specifying characteristic values or the parameters of the extension to continuous properties can be changed. Hence we use the term typical.

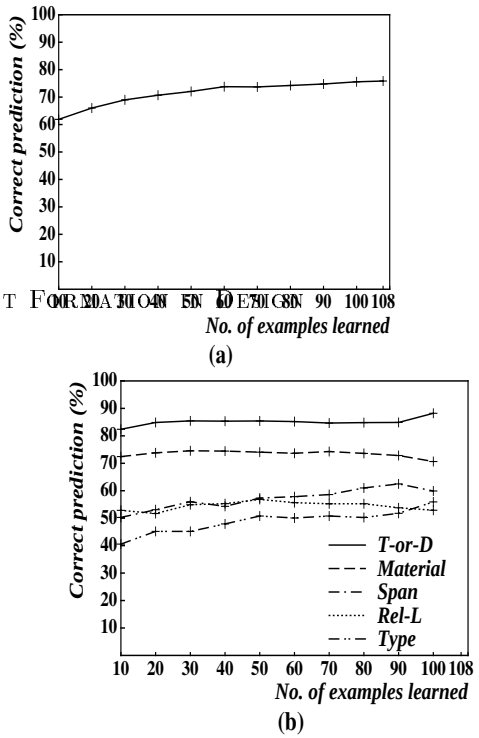

Figure 1–9. Design performance: the bridge domain

In a real domain, examples will be generated and evaluated continuously and will allow BRIDGER to better exploit prediction at internal nodes.

We should also note that the results of the individual 50 runs vary considerably. Table 1–1 shows the best, worst, and the standard deviation of the predictive accuracy of the 50 runs summarized in in Figure 1–9b. The values in the Table are based on the prediction of 8 examples remaining after learning 100 examples. The variability of the results suggests that BRIDGER's performance is dependent on the order of example presentation to a larger extent than expected, even using the split and merge operators. A better understanding of the order effect is an open research issue. We will turn to possible resolutions of this and other issues of prediction shortly.

Thus far, we have looked at how BRIDGER can be used for synthesis, which is our primary goal. However, it is also worth noting that the system can be used for evaluation or analysis. Recall that this is

Table 1–1. Variability of results in performance test

	T-or-D	Material	Span	Rel-L	Type
Best predictive accuracy	1.0	1.0	1.0	0.875	0.875
Worst predictive accuracy	0.333	0.5	0.142	0.25	0.125
Standard deviation	0.177	0.121	0.219	0.152	0.172

a process of mapping design descriptors to specification or behavioral properties. We can intuitively illustrate an evaluation-like process in an imported-automobile database compiled by Schlimmer from (Ward, 1985).

5.2 'Evaluation' in an Imported-Automobile Database

The automobile domain contains 206 examples described by nominal, continuous, and ordinal properties. The task is to predict the price of a car (i.e., a continuous property) from descriptors such as the engine type and the body style. In Bridger, ordinal properties are treated as continuous with the addition of rounding to the nearest integer. The performance test was a $5 \times 10\text{-}CV$ test.[7]

The experiment used only the 160 examples that are completely specified. One set of experiments was made with only the continuous properties so that comparisons could be made against Classit, which only handles continuous properties, and against Ibp, a supervised instance-based learning approach (Killer et al., 1989), which has been used previously in this domain. A second study included complete descriptions, which was used for comparison with Bridger's performance with the partial information. Table 1–2 shows the ability of Bridger to predict the price of a car from the remaining properties compared to the performance of Classit* (our own implementation of Classit) and Ibp. These figures represent the average percent error from the true cost of each test car over the 5 experimental trials. This task also illustrates Bridger's (and Classit*'s) ability to predict continuous properties, a novel idea for concept formation programs. Furthermore, it shows

7. An $n \times V\text{-}CV$ experiment – *V-fold Cross Validation (V-CV)* (Breiman et al., 1984) is performed over n random orderings of the examples and the results are the average of the n runs.

that BRIDGER performs better than CLASSIT*, which was developed to deal with continuous properties, and that BRIDGER slightly improves its performance when it uses the complete example descriptions. IBP performs best, but since it is a supervised program, it can only use its current knowledge to predict the price of a car which is the class property. We might loosely interpret this as a specification property (of a buyer) which BRIDGER predicts from 'design' characteristics. In general, BRIDGER's construction of a many-to-many mapping between specifications and designs provides a flexible knowledge base that supports synthesis and analysis in a uniform manner.

Table 1–2. Performance in the automobile database

	IBP	CLASSIT*	BRIDGER	
			numeric properties only	all properties
% accuracy	11.84	16.10	14.74	14.31

5.3 The bridge analysis database.

This experiment shows how bridger can predict multiple continuous values from a single hierarchy. The bridge analysis database contains 1000 examples of bridges, each described by 26 properties. The bridge examples were generated in the following way. Bridge descriptions were randomly generated by assigning values to the 26 properties. The bridges were analyzed by a program to produce 12 behavioral continuous properties, arriving at a full description containing 38 properties. This data was used to train BRIDGER and test how well can it predict the analysis results without performing the exact analysis.

A learning curve was produced in the following way. BRIDGER was incrementally trained with examples. After each 100 training examples, BRIDGER was tested on the remaining examples. In testing, each prediction of the 12 analysis results made by BRIDGER was checked against the actual analysis calculated by the program before.

The following *average* results from five runs with different random ordering of examples were obtained. The analysis prediction continuously improve when additional examples are learned. Improvements range from a 1000% average error rate after learning 100 examples to a 30% average error rate after learning 400 examples. The prediction of 4 out of the 12 analysis results converge to within 8% from the exact

values after learning 400 examples. One result remains far from the exact analysis (200% mean relative error), but this is caused only by a small number of major errors.

The studies presented demonstrate Bridger's performance on a variety of tasks. As we noted, studies in other domains yield good results relative to other supervised and unsupervised systems, and better illustrate Bridger's constructive induction abilities. To some extent though the results in the bridge domain indicate additional directions for improvement. We briefly discuss two of these now.

5.4 Other Learning Mechanisms of Bridger

Experimental results with the bridge database indicated that Bridger may require considerable data before certain of its prediction-enhancing mechanisms can be exploited, notably the use of characteristic values and intermediate node prediction. To some extent this is a limitation of many strict SBL systems. Additionally, the system demonstrated sensitivity to training order. We have made some initial progress on the problem of exploiting knowledge and mitigating ordering effects during learning.

5.4.1 Knowledge-Driven Learning and Prediction

An important characteristic of the Bridger's classification scheme is that it is declarative. This enables an external process, domain dependent or independent, to inspect it. The ability to benefit from external knowledge relies on the flexible nature of our learning method, which uses weak search methods directed by the category utility function CU. However, our eventual goal is to use CU as a default mechanism in the absence of knowledge, though explicit knowledge can be used to prefer a specific operator (e.g., merging and splitting) and override CU preferences.

Another way of using knowledge is to construct examples that will enhance the performance of the system (Scott & Markvitch, this volume). This approach is essentially an experimentation technique. Work is underway to construct such mechanisms. Finally, another kind of domain knowledge that can be used is knowledge about modeling. For example, such knowledge can suggest important features of objects. Given an ini-

tial description of objects, the knowledge can augment it by additional properties, compress the description of several properties, or erase irrelevant properties. To some extent these abilities are similar to those proposed by Stepp and Michalski (1986) in their CLUSTER/S system (also see Fisher & Pazzani, Chapter 6, this volume). Such knowledge can be augmented by a constructive induction mechanism that generates higher-level features from ground-level properties.

5.4.2 Mitigating Order Effects

Large variance in the results of the Bridge domain indicates Bridger's sensitivity to the order of training examples. Our approach to mitigate this problem is to use knowledge about specifications to reorganize the hierarchical knowledge structure in a process called hierarchy-correction. Figure 1–10 depicts the hierarchy generated from 108 bridge examples. Class G24, which is characterized as a *highway* class, is placed below a large *railroad* class. It might be beneficial to translate this class to a more appropriate branch of the hierarchy.

The hierarchy-correction scheme follows three steps. First, properties that are deemed most critical by a domain expert are manually selected as 'triggers'. Second, the hierarchy is traversed top-down. Each class with a characteristic property value that differs from a characteristic in one of the class' ancestors, is removed along with its subtree from the hierarchy. Third, the examples at the leaves of all the removed subtrees are reclassified into the hierarchy. The process can iterate several times until no change of the hierarchy is obtained. The application of this procedure enhances the predictive accuracy of Bridger.

6. A Framework for Design and Concept Formation

One of the most important aspects of a concept formation or design systems is the *knowledge* provided to the system and its interaction with the system's *control*. The behavior of the system is determined by the controlled application of its knowledge on the input received from the environment. The less knowledge a system has, the more it tends to use exploratory and exhaustive search, while the more knowledge it

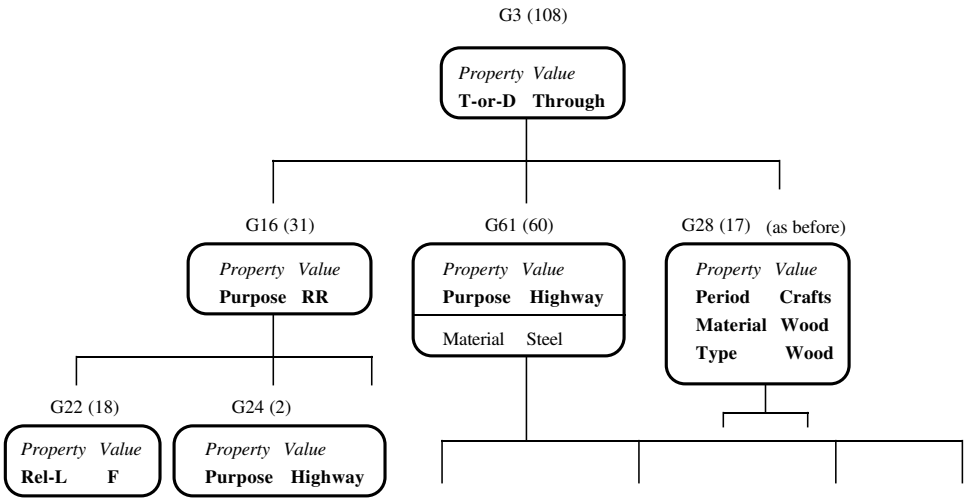

Figure 1–10. Top hierarchy after learning 108 examples

has the more it tends to be conservative and fixed.[8]

Figure 1–11 shows several concept formation and conceptual clustering programs classified along the dimensions of exploration and knowledge (without being precise about the relative location of the systems). The classification of some of these systems is elaborated below.

- Cobweb has a fixed control strategy based on the CU function and rudimentary knowledge about the collection of properties describing objects in a domain. Its control strategy is highly restricted, since it can only make one of a few local changes (e.g., through merging and splitting) to its hierarchy at any one time.

- Cluster/S (Stepp & Michalski, 1986) employs domain knowledge and a goal dependency network to identify the most 'relevant' properties in the formation of clusters. While Cluster/S follows a more extensive search control strategy than a system like Cobweb, it is not amenable to incremental refinement.

- Genesis (Mooney, this volume) is a system that uses domain knowl-

8. The knowledge and control aspects and their tradeoff are similar to Newell's knowledge and search dimensions and their tradeoff stating that the more domain knowledge available the less search deliberation is needed to solve a problem (Newell, 1989).

edge with EBL mechanisms to form concepts, which may be problematic for some types of design. Its control is fixed.

- EXOR (Yoo, Fisher, & Yang, this volume) uses a domain theory to build explanations as in EBL, and clusters over derived explanations. The hierarchical clustering is used to guide the construction of new explanations, much like BRIDGER uses the hierarchy to guide the construction of designs.

- BRIDGER, the learning system, extends COBWEB in that it may use knowledge in its hierarchy correction scheme, and can perform additional learning processes such as constructive induction. Thus, it increases COBWEB's abilities in both the knowledge and control directions.

Similar classifications relative to our framework can be made for design systems. Usually, the systems' desired behavior (e.g., routine design versus innovative design) determines the knowledge and control required.

- 1stPRINCE has qualitative knowledge that it uses to reason from first principles (Cagan & Agogino, 1987). Its control is guided by optimization techniques that are exploratory in nature. These two ingredients facilitate innovative, albeit simple, design behavior.

- AIR-CYL has stiff control composed of known fixed design plans, and limited knowledge representation schemes composed of sets of well understood design alternatives. Considerable compiled knowledge is necessary to formulate these ingredients, but they can only support a certain class of routine design (Brown & Chandrasekaran, 1986).

- Though we have not stressed certain aspects of BRIDGER, the design system, it can use certain exploratory-design and redesign procedures that allow considerable flexibility during design. Some of this flexibility is discussed in Section 4.2.1 when describing the various ways of synthesis. This is in contrast to BRIDGER's learning mechanism which always generate a new leaf node from a new example. Therefore BRIDGER, the design system, is placed high along the exploratory axis. BRIDGER's design procedures are purely syntactic; domain knowledge is only exploited a learning time. Therefore BRIDGER is placed low along the knowledge axis

Figure 1–11. A framework for describing concept formation and design systems

This framework can aid in the selection of learning techniques for a particular design task, but it also shows the gaps in our knowledge about the use of learning in design. The sparseness of examples in the framework suggests that much work remains to be done.

7. Conclusions

Design is a complex task. This chapter concentrates on the synthesis aspect of design: the assembly of candidate designs from an infinite set of alternatives. Synthesis is an ill-structured process with respect to all three characteristics of such problems. The only process that can reduce the complexity of synthesis is the acquisition and reorganization of knowledge. An appropriate knowledge structure guides the search for alternatives in an efficient manner.

Mapping the requirements of synthesis onto the functionality of machine learning techniques reveals that incremental concept formation is an appropriate means for acquiring design knowledge in general. In particular, we present BRIDGER, a concept formation system that can partially automate the acquisition of design knowledge for a class of design

domains where the artifacts' structure is fixed. There are also aspects of the learning/design mapping that we have not stressed. For example, specification properties roughly correspond to operational predicates exploited by EBL systems (Mooney, this volume; Yoo, Fisher, & Yang, this volume). Ideally, learning should render synthesis more efficient by allowing a designing system to construct designs appropriate to the specification (i.e., operational) predicates.

Our experiences with BRIDGER demonstrates that studying concept formation in conjunction with design can benefit the understanding and development of both. Work in design domains has forced the development of several extensions to COBWEB and the results obtained by BRIDGER provides experimental support for a General Design Theory (Reich, 1990b). The study of the two tasks also provides a better understanding of the concept formation appropriate for the acquisition of task-specific design knowledge. We believe that this understanding is further enhanced by a simple framework for classifying concept formation and design systems. This framework motivates and directs our continuing research.

Acknowledgments

This work has been supported in part by the Engineering Design Research Center, a National Science Foundation Engineering Research Center, and the Sun Company Grant for Engineering Design Research. We would like to thank Doug Fisher and Mike Pazzani for inviting us to write this chapter. Doug Fisher, through a dedicated editorial work, has provided many helpful comments on drafts of this chapter.

References

Arciszewski, T., Mustafa, M., and Ziarko, W. (1987). A methodology of design knowledge acquisition for use in learning expert systems. *International Journal of Man-Machine Studies*, *27(1)*, 23–32.

Breiman, L., Friedman, J. H., and Olshen, R. A. (1984). *Classification and Regression Trees*. Belmont, Waldsworth, CA.

Brown, D. C. and Chandrasekaran, B. (1986). Knowledge and control for a mechanical design expert system. *IEEE Computer*, *19(7)*, 92–100.

Cagan, J. and Agogino, A. M. (1987). Innovative design of mechanical structures from first principles. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, *1(2)*, 169–189.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2(7)*, 139–172.

Gennari, J. H., Langley, P., and Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, *40(1-3)*, 11–61.

Gluck, M. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287, Academic Press, Irvine, CA.

Kibler, D., Aha, D. W., and Albert, M. K. (1989). Instance-based prediction of real-valued attributes. *Computational Intelligence*, *5(2)*, 51–57.

Kolodner, J. L. (1983). Maintaining organization in a dynamic long-term memory. *Cognitive Science*, *7()*, 243–280.

Lu, S. C. and Chen, K. (1987). A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision-making. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, *1(2)*, 109–118.

Magued, M. H., Bruneau, M., and Dryburgh, R. B. (1989). Evolution of design standards and recorded failures of guyed towers in Canada. *The Canadian Journal of Civil Engineering*, *16(5)*, 725–732.

McLaughlin, S. and Gero, J. S. (1987). Acquiring expert knowledge from characterized designs. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing*, *1(2)*, 73–87.

Michalski, R. S. and Chilausky, R. L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples: a case study involving soybean pathology. *International Journal of Man-Machine Studies*, *12(1)*, 63–87.

Michalski, R. S. and Stepp, R. (1983). Learning from observation: Conceptual clustering. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363, Tioga Press, Palo Alto, CA.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of AAAI-86*, pages 1041–1045, Morgan Kaufmann, Philadelphia, PA.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: a unifying view. *Machine Learning*, *1(1)*, 47–86.

Mooney, R. J. (1990). Explanation-based learning as concept formation. In Fisher, D. and Pazzani, M., editors, *Computational Approaches to Concept Formation*, Morgan Kaufmann, Los Altos, CA. (in press).

Newell, A. (1989). Putting it all together: final comments. In Klahr, D. and Kotovsky, K., editors, *Complex Information Processing: The Impact of Herbert Simon*, pages 399–440, Erlbaum, Hillsdale, N.J.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1(1)*, 81–106.

Quinlan, J. R. (1989). Personal communication.

Ramachandran, N., Shah, A., and Langrana, N. A. (1988). Expert system approach in design of mechanical components. *Engineering with Computers*, *4(4)*, 185–195.

Reich, Y. (1989). *Combining Nominal and Continuous Properties in an Incremental Learning System for Design.* Technical Report EDRC-12-33-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. (1990a). Constructive induction by incremental concept formation. In Feldman, Y. A. and Bruckstein, A., editors, *Proceedings of the Seventh Israeli Symposium on Artificial Intelligence and Computer Vision*, pages 195–208, Elsevier Science Publishers, Tel-Aviv, Israel.

Reich, Y. (1990b). Converging to "Ideal" design knowledge by learning. In Fitzhorn, P. A., editor, *Proceedings of The First International Workshop on Formal Methods in Engineering Design*, pages 330–349, Colorado Springs, Colorado.

Reich, Y. (1990c). Incremental clustering with mixed property types. Unpublished manuscript.

Reich, Y. (in press). Design knowledge acquisition: Task analysis and a partial imlementation. *Knowledge Acquisition*. An earlier version of this paper appears in the Proceedings of The Fifth Bannf Knowledge Acquisition for Knowledge-Based Systems Workshop, 1990.

Reich, Y. and Fenves, S. J. (1989). The potential of machine learning techniques for expert systems. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 3(3)*, 175–193.

Reich, Y. and Fenves, S. J. (1990). Automatic design knowledge acquisition by concept formation. In Arciszewski, T., editor, *Knowledge Acquisition in Civil Engineering*, American Society of Civil Engineers, New York. (in press).

Reitman, W. R. (1964). Heuristic decision procedures, open constraints, and the structure of ill-defined problems. In Shelly, M. W. and Bryan, G. L., editors, *Human Judgments and Optimality*, pages 282–315, John Wiley & Sons, New York.

Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence, 4(3-4)*, 181–201.

Steels, L. and Van de Velde, W. (1986). Learning is second generation expert systems. In Kowalik, J. S., editor, *Knowledge-Based Problem Solving*, pages 270–295, Prentice-Hall, Englewood Cliffs, N.J.

Steinberg, L. I. (1987). Design as refinement plus constraint propagation: The VEXED experience. In *Proceedings of AAAI-87*, pages 830–835, Morgan Kaufmann, Seattle, Washington.

Stepp, R. E. and Michalski, R. S. (1986). Conceptual clustering of structured objects: a goal-oriented approach. *Artificial Intelligence, 28(1)*, 43–69.

Stone, J. and Blockley, D. (1989). Finding patterns in structural failures by machine learning. In *Expert Systems in Civil Engineering, IABSE Colloquium, Bergamo*, pages 47–56, IABSE, Zurich, Switzerland.

Taura, T., Kubo, S., and Yoshikawa, H. (1989). Generation of design

solutions by a metric space. In , editor, *Proceedings of the Third IFIP W.G 5.2 Workshop on Intelligent CAD*, page , Osaka, Japan.

Thompson, K. and Langley, P. (1990). Managing components in object concept formation. In Fisher, D. and Pazzani, M., editors, *Computational Approaches to Concept Formation*, Morgan Kaufmann, Los Altos, CA. (in press).

Ward (1985). 1985 model import car and truck specifications. In Stark, H. A., editor, *Ward's Automotive Yearbook, Vol. 47*, pages 160–163, Ward's Communication, Inc., Detroit, MI.

Yoo, J. and Fisher, D. (1990). Concept formation over explanation and problem-solving experience. In Fisher, D. and Pazzani, M., editors, *Computational Approaches to Concept Formation*, Morgan Kaufmann, Los Altos, CA. (in press).

Yoshikawa, H. (1981). General Design Theory and a CAD system. In Sata, T. and Warman, E., editors, *Man-Machine Communication In CAD/CAM, Proceedings of the IFIP WG5.2-5.3 Working Conference*, pages 35–57, North-Holland, Amsterdam.