The Knowledge Engineering Review, Vol 7:3, 1992, 183–213

Innovative design systems, where are we and where do we go from here? Part I: Design by association

D. NAVIN CHANDRA

School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA

Abstract

Designing is a skill central to many human tasks. Designers are constantly producing newer and better artifacts, generating innovative solutions to problems in our world. This paper looks at innovation, and research that is aimed at developing theories and methodologies for innovative design. We view design as a process of *association* and *exploration*. These two approaches are fundamental to innovation. The aim of exploration is to generate a large variety of design alternatives by breaking away from the norms by looking in unlikely places, and by relaxing binding constraints. Exploration exposes possibilities that would not normally have been considered, possibilities that may serendipitously lead to innovative solutions. Association, on the other hand, attempts to exploit previous design experiences in a new design context. This is done by recognizing useful analogies that can help in synthesizing parts of a design, recognizing unforeseen problems, and discovering opportunities. This paper is the first part of a two-part paper that presents and discusses a variety of association and exploration methods. This part examines association-based techniques, some of which have been used in actual design systems, and others that point to the solution of some open questions in design research. We develop these ideas by examining connections between design research and other disciplines such as cognitive psychology, artificial intelligence, the history of science, and the creativity literature.

1 Exploration and association in innovative design

Design is the process of producing artifacts that have desired properties and meet a set of functional requirements. Designing is a skill central to many human tasks: architects deal with shape and form to create new buildings, financial planners manipulate investments to design profitable portfolios, and mechanical engineers design functional machines with parts such as gears and cams. Designers are constantly producing newer and better artifacts. In the modern competitive world, they are under constant pressure to turn out new and innovative products quickly. In order to improve the productivity of designers, computer aided design tools have been developed. In the short term, these efforts have focused primarily on automating the more routine and tedious tasks involved in design. Of the two major phases of a design process, *conceptual design* and *detailed design*, tool-building efforts have concentrated mainly on the latter phase. Applications have been limited to tasks such as computer-aided drafting, solid modeling, numerical optimization, simulation, and analysis. Having made substantial contributions towards the development of tools for detailed design, researchers are now focusing their attention on the earlier phase: conceptual design.

Conceptual design is that part of the design process in which: problems are identified, functions and specifications are laid out, and appropriate solutions are generated through the combination of some basic building blocks. Conceptual design, unlike analysis, has no fixed procedure and involves a mix of numeric and symbolic reasoning. Recently, researchers have started developing knowledge-based systems for conceptual design. A majority of these efforts have concentrated on

routine design problems. A design process is deemed routine if it involves a well understood sequence of steps where all decision points and outcomes are known *a priori*—there are no surprises. An advantage of developing systems for routine design is that, as the process is well understood, there is a rich set of design heuristics that can be used to effectively control the search for solutions. Routine design systems can be organized in several different ways. For example, (1) some routine design tasks can be broken into discrete steps, where each step performs some specific design sub-task; (2) in some cases, the problem can be approached hierarchically, where each level in the hierarchy is predetermined; and (3) in other cases, specific heuristics are available for the different parts of the design. These heuristics can be packed into modules, where each module is responsible for one part of the overall design. Using strategies such as these, several systems have been built for various routine design applications. Examples of such applications are: building design (Maher, 1984; Sriram, 1986), circuit design (Tong, 1986a), and the design of mechanical devices (Mittall, 1985; Brown & Chandrasekaran, 1986; Steinberg et al., 1986). Based on the success of these routine design systems, interest was generated in building systems that can handle non-routine (innovative) design problems. This paper surveys and analyses the state of the art in systems that innovate.

The survey concentrates more on the techniques used in these systems rather than what the systems actually did. The survey is based on a reconstructed classification of the various approaches reported in the literature. It goes beyond providing a mere classification of existing methodologies, and presents interesting connections to a variety of other disciplines such as cognitive psychology, artificial intelligence, biology, the history of science, and epistemology. The classification has also helped us identify 'gaps' in current research, thus leading up to ideas for future research directions.

Readers who wish to get a broader sense of the techniques used in Design Automation systems, in general, may refer to the following papers: Nilsson 1980; Stefik 1980; Mostow 1985; Brown 1984; Mittall 1985; Gross 1986; Tong 1986a; Sriram 1986; Tong 1986b; Ullman & Dietterich 1987.

1.1 Basis of the survey

The research efforts and systems surveyed in this article have all been viewed in light of two fundamental approaches to innovate design: *exploration* and *association* (Navin chandra, 1991).

Innovative design can be viewed as a process of *exploration*, a process in which one deviates from the beaten path to generate new solutions to problems—breaking away from the norms, and looking in unlikely places. Exploration exposes possibilities which would not normally have been considered, possibilities which may serendipitously lead to innovative solutions.

In addition to exploration, designers rely on prior designs to generate new designs. Faced with new design situations, designers rely either on their own experiences or on knowledge drawn from sources outside the current problem domain. Drawing ideas from outside requires the ability to recognize useful analogies. We call this design by *association*. The idea is based on the notion of *bisociation*: the 'perceiving of a situation or idea ... in two self-consistent but habitually incompatible frames of reference ... (Koestler, 1964)'. This is an effective way of achieving innovation. The important point is habitual incompatibility, the more different two domains are perceived to be the more innovative the designs that associate the two domains will appear to be.

This part of a two-part article covers design by association. We start with a discussion of the role of association and exploration in innovative design. The paper then concentrates on computer based methods of association including analogy and case-based reasoning. The bulk of the paper presents and discusses techniques for design representation, indexing, matching and synthesis. In each area, we examine current techniques and take a peek at future research directions.

1.2 What is innovative design?

Before talking about tools and techniques, we first try and characterize innovative designs. Earlier definitions of innovative design have tended to finesse over details. For example, innovative design

has been defined as any design that is: new or different or elegant or uses new ideas or is an improvement over its peers (Dixon, 1965). Such a definition is correct, but it does not tell us how to measure 'newness'. It does not tell us how to produce innovative designs. In this section we shed some light on the problem of trying to distinguish between innovative and routine designs. The purpose of trying to define innovative design is to try and identify its characteristics. With a list of such characteristics we can better focus our research efforts. If Design Automation systems that emulate some of these characteristics are built, then such systems can be expected to exhibit innovative behaviour.

The first question to ask is: 'When does a design qualify as innovative (as opposed to routine)?' Ironically, the answer is that there is no absolute measure of innovativeness. It does not make sense to label a design as being innovative, because the perception of an artifact as being novel or not lies not in some inherent property of the artifact but in the eyes of the beholder. For a given observer or a given group of observers (e.g. mechanical engineers) there is a set of design styles they are accustomed to seeing in the artifacts designed by their peers. For example, a civil engineer will view a construction robot (e.g., an excavation robot) as being innovative. On the other hand, a robot designer will view the same construction robot as being yet another application. The robot's sophisticated vision, tactile, and position sensing systems are not new to the robot designer and are part of the *design culture* he works within. For the civil engineer, on the other hand, an excavation robot is novel because robotics is outside his design culture. *A design culture is defined by the common practices, design styles and technologies used by people who operate within the culture. It defines the context they operate within, it's their welthanschauungen* (Navin chandra, 1987). The notion of design culture gives us a datum from which designs can be characterized, where each culture has its own way of viewing designs as being routine or innovative.

We devote the rest of this section to the characterization of exploratory and associational processes in design.

1.3 Design by exploration

Designers working within a culture produce solutions to new problems by using known *techniques* that combine known *technologies*. Techniques and technologies are the two basic components of a design culture. For example, a building designer might combine floor slabs, pillars and walls to come up with new building configurations. These objects are part of the physical technology of the culture. The building design is developed by shaping and composing the available physical parts and materials using standard practices, styles and methods (*technique*). For instance, the creation of bays with regularly spaced pillars and the creation of rooms by erecting walls between pillars is a common and well-known technique of the culture.

Designs that are based on known compositions of techniques (practices and styles) from within a culture will appear mundane. The corresponding design task is one of puzzle solving (Kuhn, 1970), even though individual mundane solutions may be different from their predecessors, they will only be part of current practice. There are many ways in which a design can deviate from the mundane. These ideas are best explained using the notion of a state space. The aim of developing such an explanation is to characterize and specify ways for achieving variety. The various problem spaces are shown in Figure 1*. The outermost region (A-space) is the space of all legal combinations of the technology-base available in a culture. For example, in electronic chip design all possible layouts of logic gates make up the A-space of the culture. Note that in our definition of the A-space, only legal combinations are allowed. Legality may be based on some basic physical laws. For example, two logic gates cannot occupy the same space on the chip.

At the core of the A-space is the space of existing designs (E-space). Surrounding the core is the space of all designs which could be developed using the currently known heuristics, patterns and

*The use of venn-diagrams to visualize the solution space of a design domain was introduced by John Gero (1987, 1990).

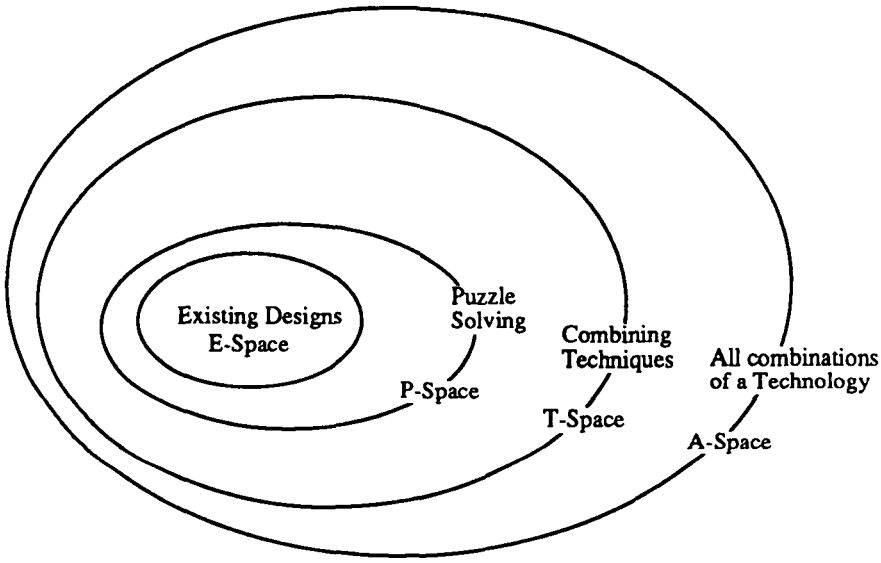

Figure 1 State spaces of a design culture

styles. This is the space generated by what is called Puzzle Solving (P-space). During puzzle-solving, the designer is only responding to new situations using known techniques and technologies. Sometimes, a design agent may go beyond puzzle solving by trying to combine known techniques and technologies in new ways. Let us call this space of design solutions the T-space. Consider an example from the domain of recipe design. A recipe generally has two parts: a list of ingredients (technology) and a method of preparation (technique). One can imaging coming up with a new dish by using the ingredients of one recipe, but using a method that combines the methods of two or more recipes. For instance, one could combine a cake and a cookie recipe. Using the ingredients for a cake, one can use a combination of techniques taken from the cake and the cookie recipes. This produces little cakes that are crispy outside but soft and fluffy inside. A majority of the innovative design systems generate designs somewhere between the T-space and the A-space. The author is unaware of any system that is able to dynamically mix and match the problem solving techniques it knows about†. Finally, a design (or part thereof) may be generated by randomly permuting all available technologies. This process can generate designs that known techniques cannot. Permuting elements of a design culture's technology can generate solutions anywhere in the A-space of the culture.

In this framework, design instances outside the P-space (Fig. 1) will appear innovative to designers who operate within the P-space of the culture. Venturing outside the P-space is called exploration. The simplest exploration process is that of permuting intra-cultural technologies. The A-space of a design culture, however, is so large that random permutations will rarely yield useful solutions. A possible way of approaching this type of exploration is to use a tree-based search technique. The search may be guided by constraints and heuristics that help determine which branches to prune and which ones to expand next. Several variations and extensions of this basic idea are reported in the literature: (Newell & Simon, 1972; Fikes, 1969; Sussman, 1980; Stefik, 1980; Fox, 1983; Mittal, 1985; Gross, 1986).

Following are some desirable characteristics of an exploration technique:

- does not discard good solution paths, even if they initially appear to be inferior to other paths. The idea is that such paths may unexpectedly lead to interesting solutions.

†One area of research comes close—the classifier system concept, as described in the genetic algorithms literature (Holland, 1975), appears to produce solutions in the T-space of the underlying genetic representation. Woodbury has suggested treating search-space heuristics as a genotypic representation of designs (Woodbury, 1989). The idea is to generate new designs by mutating search space operators that are semantically relevant.

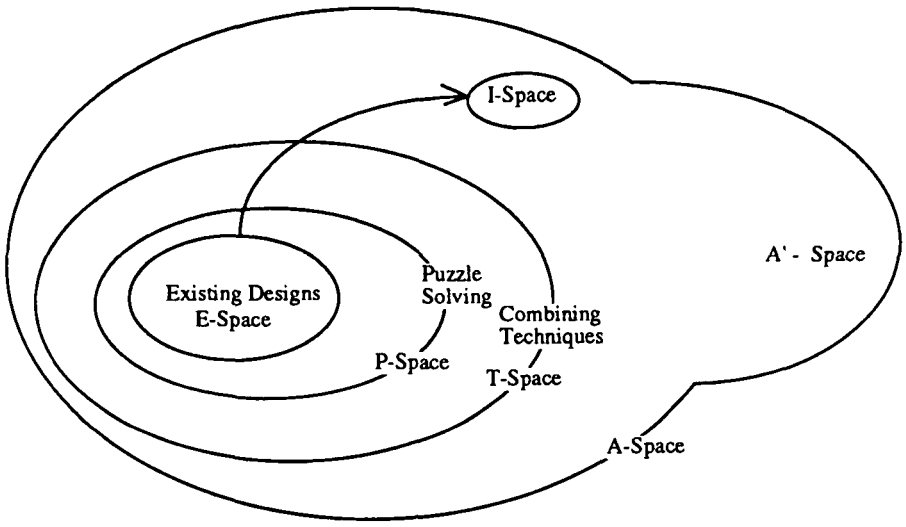

Figure 2 Extending a design culture

- does not generate too many similar solutions. This is a problem with algorithms that are based on a tree-like search technique. Solutions generated at a branching can sometimes be too similar to be considered as separate alternatives. Though quantity is important, it is variety that is vital to innovative design.
- can explore variations of a given design without altering the essence of the initial solution.

1.4 Design by association

If a designer reaches for new ideas beyond the knowledge he normally uses during design, his product will appear different. The further he reaches the more different and novel his design will appear to be. The reason for this phenomenon is that, for a given culture, there are some cultures (or domains) that are perceived as being related, while other cultures are viewed as being far and unrelated. When ideas are transferred between unrelated cultures the resulting designs appear different, perhaps novel. The cross-cultural transference of knowledge includes both technologies and techniques. Importing technologies into a culture increases the number of physical primitives that can be combined to produce new designs. This yields an extended A-space (shown in Fig. 2 as A'-space). For example, when micro-electronics were introduced to the domain of household appliances, the A-space of the appliance domain expanded considerably.

The relatedness or un-relatedness of cultures is determined socially, and is constantly in flux. For example, when ideas about automation were first taken from industrial assembly lines to applications at home (washers, food processors) the transfer was considered very novel. In fact, companies had problems trying to convince home-makers that automation at home is safe and acceptable. Today, advanced electronics is viewed as part of the home appliance design culture. We are now accustomed to programmable ovens and talking refrigerators.

Importing techniques/heuristics from other domains can cause a jump from the current P-space to an Island of solutions (I-space). The I-space is a set of solutions generated by techniques that help the designer look at known technologies in a different light. The I-space can either be in the A-space or in an extended A'-space. For example, the field of fluidics is an application of logic circuit design techniques to the hydraulics design culture. Fluidics is in the I-space of the hydraulics domain. It involves techniques drawn from electronic logic circuit design but does not use the same physical technology (electronic devices). Fluidics technologies include pipes (used as wires), special valves (logic gates), pumps (potential sources) etc. These technologies are all native to the hydraulics culture but the techniques for combination come from outside.

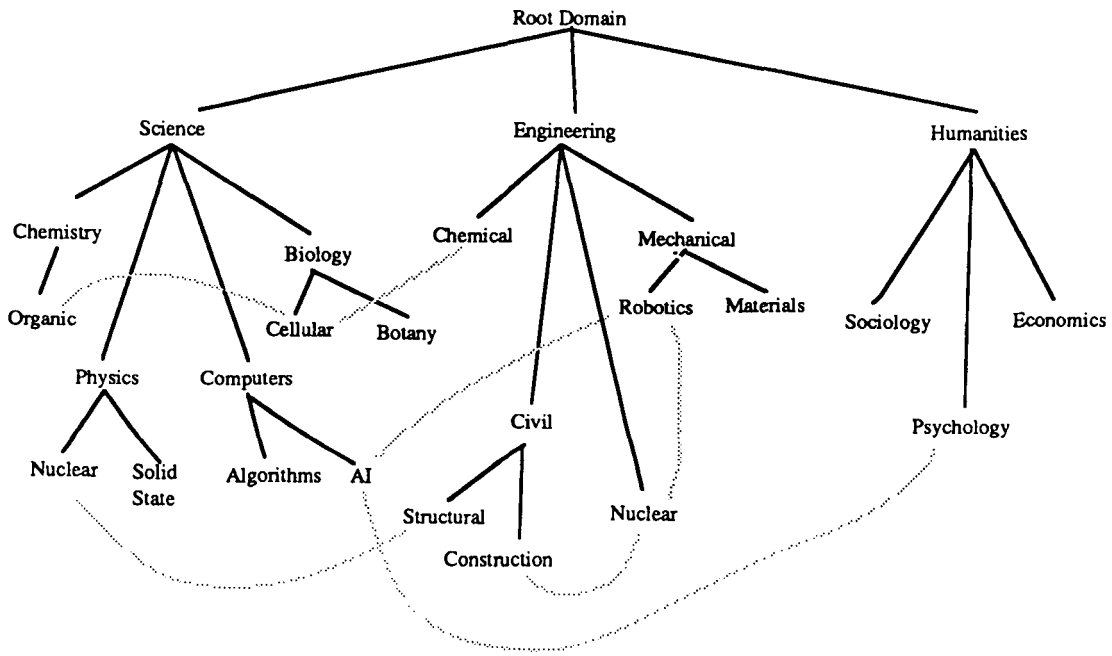

Figure 3 A domain hierachy

Drawing useful analogies during problem solving can help generate solutions outside the P-space. The more 'different' the base and the target of the analogy, the more likely that the resulting solution will appear innovative in both the base and the target cultures. In a computational context, the 'difference' of two cultures can be measured using a hierarchical structure of domains. For example, the tree shown in Fig. 3 shows relationships among domains as an acyclic graph. Cross connections among domains is shown by grey lines. In this framework, the larger the number of arcs one has to traverse to go from one domain to another, the more different (or distant) they are. As more and more cross-cultural designs are generated, the hierarchical structure has to be reorganized to reflect which domains we regard as being close and which ones continue to be unrelated. Such domain graphs are particularly useful in innovative design systems. For example, if one has to choose among two competing extra-cultural strategies for debugging a design, choosing the strategy taken from a 'farther' domain will make the final solution appear innovative. There are merits and demerits of this approach. A demerit is that designs based on ideas drawn from very different cultures are more likely to fail. On the other hand, a novel solution may serendipitously solve the problem more efficiently than anticipated or may help satisfy many more criteria than originally stipulated.

Viewing design as a process of exploration and association gives raise to several specific and probing questions that can be asked of any design system:

- Can the program reason analogically?
- Can it reason at several levels? Can it use principles and results of several domains?
- Does the program have access to knowledge outside it's original design culture? If so, how can it use this outside knowledge. How does it represent and index knowledge.
- Can the system generate alternatives? Can it explore beyond the P-space and look in unlikely places?
- Does it know how and when to relax constraints and drop barriers?
- Will it recognize an innovative solution when it serendipitously stumbles on one?

These issues are open research problems in artificial intelligence and design automation. The rest of this survey reviews work that has been done towards answering the first three questions listed above. The next three questions are addressed in the second part of the two-part article. In this part, we concentrate on associative techniques.

2 Design by association: an introduction

New ideas often have their origins in old ideas. Almost all new designs involve the use of previously acquired knowledge. A designer might acquire such knowledge from his own life experiences, previous designs and from prior education. While designing, people normally draw upon this knowledge to solve new problems. Sometimes, however, a design task may not be solvable with the domain knowledge possessed by the designer. Under these conditions, the designer has to either learn more about the design culture or try to use knowledge from outside the current domain.

Consider the following scenario: a biomedical engineer is given the task of designing a non-surgical method for removing urinary calculii (a calcium deposit in the urinary tract). The task requires innovations drawn from outside the biomedical design culture. The designer has to find some way of reaching the calculii and removing it without surgery. The requirement of finding a non-surgical method reminds him of catheters. This idea is from within his design culture. Now that he has a way of reaching the calculii, he has to find a way of grabbing it. This reminds him of kitchen tongs and he decides to put small clips at the end of the probe. Next, he has to pull out the calculii without scratching the insides of the urinary tract. This can be done by expanding the tract. The work 'expanding' reminds him of a balloon and he decides to introduce a small balloon at the end of the probe. This will aid in pulling out the calculii. This example, taken from a real design problem (Pahl & Beitz, 1984), shows how remindings can occur from within and without the design culture during problem solving. As the designed artifact drew upon knowledge from outside the design culture of bio-medical engineering it was viewed as being different and innovative.

The ability to draw upon knowledge that is analogically related to the current design problem is an important ingredient of innovative design. Experienced engineers have the ability to utilize knowledge gained from previous experiences to provide novel solutions to a wide range of problems. This type of reasoning should be incorporated in programs that attempt to emulate the intelligence of engineers. A program should be able to recognize similarities between a current design task and previous designs. It should also be able to solve the current task by analogy to a prior design. Analogical reasoning is a way of making associations across cultural boundaries. New designs can be generated by synthesizing prior designs from different design cultures, resulting in associations that appear innovative.

The ability to draw useful analogies is considered an important part of creativity. Philosophers and psychologists have studied this aspect of human behaviour for many years. Much of the early work was aimed at describing the symptoms or end products of a hitherto unseen process that has come to be called *illumination*. Illumination is the sudden idea formation that appears to take place during a creative spell. Namely, the 'eureka' phenomenon. The notion of illumination was introduced by Wallas (1926). His model of creativity had four basic phases: preparation (preliminary work), incubation (allowing some time for the ideas to sink in), illumination, and verification (testing). This basic model was expanded and modified over the years (Rossman, 1931; Osborn, 1953). Osborn extended Wallas' model of creativity to a seven step process:

1 Orientation: pointing up the problem
2 Preparation: gathering pertinent data
3 Analysis: breaking down the relevant material
4 Ideation: piling up alternatives by way of ideas
5 Incubation: letting up, to invite illumination
6 Synthesis: putting the pieces together
7 Evaluation: judging the resulting ideas

During the 1950s and 1960s much interest was generated in using models of creativity to develop methods for improving creativity of people in the workplace. Two popular techniques, brainstorming and synectics, were developed. Both these techniques are aimed at helping the problem solver recognize useful analogies. For example, in brainstorming, the idea is to freely analogize and generate as many solutions as possible.

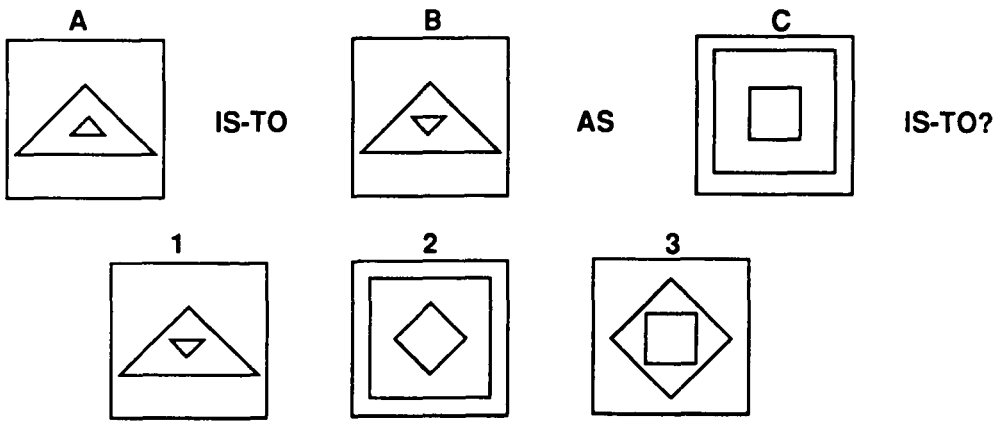

Figure 4 The ANALOGY program

Emerging from such models of creativity are techniques to aid in human creative processes. Much of the work in computer-based approaches to innovative design is based, in principle, on such techniques. Some of the major themes underlying these techniques are:

1 The idea that creativity comes, at least partially, from the ability to ask the right questions, and to use special techniques to transform the questions in interesting ways.
2 Use of analogy and bisociation—the bringing together of seemingly disparate design concepts.
3 The idea that innovations do not arise from a deliberate attempt at being innovative, but by generating a wide *variety* of alternatives and throwing away the bad ones.

2.1 Computer-based analogical reasoning: some fundamentals

Analogical reasoning (AR) involves the use of past experiences to solve problems that are similar to problems solved before: "Analogical problem solving (reasoning) consists of transferring knowledge from past problem solving episodes to new problems that share significant aspects with corresponding past experiences—and using the transferred knowledge to construct solutions to the new problems." (Carbonell, 1986).

Analogical reasoning can be viewed as consisting of four basic processes (Gentner, 1983): (1) *retrieval*: given the target, this process notices and retrieves a potentially analogous state and places portions of the base and target in correspondence; (2) *elaboration*: given the base and the knowledge about the base, derive additional attributes, relations, and complex causal chains involving the base; (3) *mapping*: given base attributes, relations and causal chains, map selected ones over to the target (with modifications); (4) *justification*: given mapped target attributes, relations and causal chains, justify they are, in fact, valid. Modifications are made if needed.

The rest of this section provides a brief history of research in analogical reasoning.

Evans. One of the first interesting programs was called ANALOGY (Evans, 1968). This program could solve simple visio-spatial problems like the one in Fig. 4.

One of the important ideas introduced by Evans was the use of generalizations in the derivation of the transformation rules. For example, in Fig. 4 we can describe the transformation rule for mapping A to B as 'Rotate the inner-most triangle'. The ANALOGY program can generalize the rule to 'Rotate the inner-most object'.

Kling introduced a program zorba, that proved theorems analogically (Kling, 1971). The program is given a theorem to prove. It is then given an analogous base theorem to use to prove the target. The program is interesting, in that, it proves a theorem efficiently by using an analogous theorem and proof, and thereby reduces unnecessarily search. The program could not perform generalizations, as the ANALOGY program did.

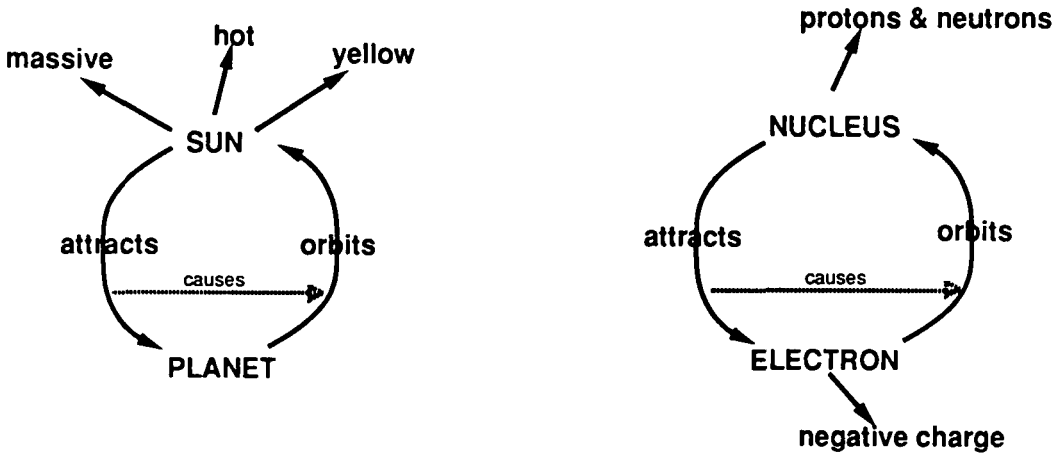

Figure 5 The atom is like the solar system

Winston (1980, 1988) made a considerable contribution to analogical reasoning. Much of his work uses a characteristic representation scheme. Mapping is carried out by matching the links of the base and target representations.

Carbonell, introduced a program that solved problems by combining analogical reasoning and means/ends analysis (Carbonell, 1983a). Given an initial state (base) and a final state (target), the program searches the space of solutions using transformation operators. This approach was called *transformational analogy*. A limitation of transformational analogy was the fact that problems that share some characteristics need not share problem solving strategies. This led to work on derivational analogy (Carbonell, 1983b). The mapping of base to target is done by transferring the base's reasoning process (derivation) to the target.

Structure Mapping, Causality and Explanations. Analogical matching algorithms are usually based on matching relationships among features (Jevons, 1892) has noted that ". . . analogy denotes a resemblance not between things, but between the relations of things . . .". This basic idea has now been put in AI terms (Gentner, 1983; Gentner & Toupin, 1986; Prieditis, 1988). Drawing an analogy requires the ability to match a target problem to a relevant base even if they have very different surface characteristics (Kedar-Cabelli, 1985a). This is done by matching the underlying causal structure of the base and target. The idea draws from the *systematicity-principle* which states that in order to find an analogical match between base and target, it is more important to find common causal relationships among attributes in base and target rather than just common attributes (Gentner & Toupin, 1986).

One of the limitations of early work was the belief that base and target should be matched using the feature sets (Tversky, 1977). This approach is limited compared to the Structure Mapping theory (Gentner, 1983) and the systematicity principle (Gentner & Toupin, 1986). The systematicity principle is used to map base to target by concentrating on the causal relations while regarding all other relations as independent. An interesting example builds the analogy: 'The atom is like the solar system' (Fig. 5).

Various design systems have been built that capitalize on the above ideas. To reason by association, these systems rely on memories of prior designs or other cases that could be relevant to a given design task. Issues relating to the representation, indexing and matching of cases in memory have dominated the literature in recent years.

2.2 Associative reasoning in design systems

In building a design system that exploits associative reasoning techniques, several basic issues need to be addressed. Typically, a design system has a database of prior designs from which it draws

knowledge that is analogically relevant to the current design context. The first issue in developing such a system is the selection or creation of a *representation*. The representation should be able to capture all the salient features of the design that are relevant to the domain. For example, if one is interested only in the kinematic behaviour of devices, one does not have to represent the full physical model of the device. It often suffices to use a abstraction that is relevant to the goals of the system. Once you get all the prior designs represented, they need to be stored in a database. This raises the issue of *indexing*. If one has to recognize similarities among designs, the indexing scheme has to support retrieval that is associative, not just exact matches. This is in contrast to regular databases that are designed to retrieve information that matches a given query. This leads up to the issue of *matching*. Both the representation and the indexing scheme has to support inexact or analogical matching schemes. Various approaches have been proposed in the literature for finding similarities among designs. Some approaches use shared behaviours for matching, others look for common attributes, while some use causal relations. After matching the design cases that are retrieved from storage have to applied to the current design context. This final issue/step is called *synthesis*. One has to extract and apply knowledge contained in the prior design to the new design problem. In many domains the context or goals of the prior design might be quite different from those of the current task. Successful transfer of knowledge requires that the prior knowledge be adapted to the current task.

In the rest of the paper we will examine the above issues in detail. Once again, the issues discussed in the sections to follow are: representation, indexing, matching and synthesis.

3 Design representation

In the past, design systems have relied only on knowledge drawn from within one domain. In order to build systems that can reason analogically from precedents from within and without the current design domain, we need better canonical knowledge representation schemes. In an example presented earlier, we saw how a biomedical engineer was reminded of a balloon while trying to solve a urinary disorder. This means that the balloon precedent has to be represented in such a way that it could be used for very different purposes. One easy way out would be to represent the balloon precedent such that it can be retrieved by a few distinct indices such as: expanding, soft, thin, etc. This will work for some limited cases but does not capture the fact that people can often retrieve precedents to solve problems in ways we had never imagined before. This means that a precedent should be represented such that it can be viewed in many different ways and can thus be used to serve several different purposes.

A popular representation technique used in the machine learning and analogy research is predicate logic. The work on explanation based learning and purpose directed analogy uses this formalism (Kedar-Cabelli, 1985b). People working on design automation systems for VLSI circuits use standard circuit description languages that allow for abstraction and reasoning about functional behaviour. In many engineering domains, however, there is a wide semantic difference between an artifact's representation and its function/behaviour. For example, a stapler can be used as a paper weight, as a nut cracker, as a hammer, its spring-action could be used to launch projectiles such as pencils, open it up and it could be used as a set of crude weighing scales. How does one represent the stapler so that it can be retrieved for many different purposes?

There are a host of questions that arise regarding representation and indexing of cases in the context of a creative and synthetic task such as design. What constitutes a 'design case'? What information should be incorporated? What features could be used as indices? How are these features extracted from the input? How can the various reasoning levels be captured in an expressive and efficient manner enabling a problem solver to move between those levels during the problem solving process? How is numerical and analytical information incorporated into cases? In the rest of this section we will examine representation issues as they relate to design automation systems. Following, is a list of some representation schemes found in the engineering design automation literature (see also [Sycara & Navin chandra, 1989a]):

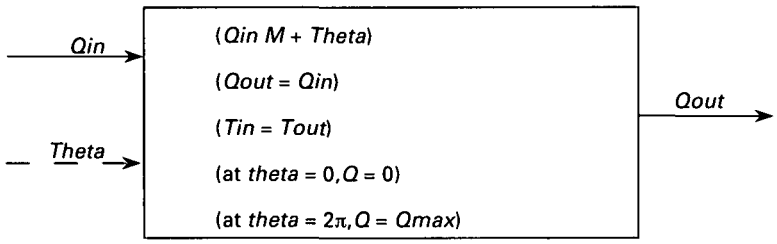

Figure 6 Qualitative device description of a tap

1. Linguistic. Linguistic indices are best suited for direct indexing based on a matching linguistic index. Case attributes provide such indices. For example, a simple household water tap can be indexed in terms of its function, to control water flow; its components, pipe, nozzle, handle, valve and seal; the material out of which it is made, brass; the type of device it is, mechanical; the places where it is intended to be used, kitchen, bathroom, water tank.

Previous research in index determination in the Case Based Reasoning (CBR) literature (Hammond, 1986; Kolodner, 1988; Sycara, 1987) has identified goals as well as object attributes as general classes of features that can be used as indices. Since artifacts always have an associated intended function as the purpose/goal of the artifact, it is clear that the intended functional specification should give rise to a set of related indices. In addition, features that capture the physical description of the device (object features) need to be included.

Linguistic indices may also be used to describe behaviour. This is done in a mechanical design synthesis system called EDISON (Dyer et al., 1986). In the EDISON system, artifacts are represented in terms of parts, spatial relations, connectivity, functionality, and processes. For example, to describe the relative orientation of two objects, EDISON uses predefined notions such as: coaxial, colinear, adjacent, overlapping, etc. Using a set of production rules which know about the behaviour of these different types of connections and spatial relations, the system is able to simulate the working of the artifact.

2. Functional description. Devices can be viewed as black-boxes that take inputs and produce desired outputs. In the physical domain, three types of inputs and outputs have been identified: signals, energy and materials (Pahl & Beitz, 1984). A characterization of the relationships between the input and the outputs is the device behaviour. Device behaviour can be represented at several levels of detail. At the highest level, the behavioural description contains the overall inputs and outputs, whereas at more detailed levels domain principles such as Bernoulli's theorem, Newton's laws and conservation laws could be used.

This kind of representation is used in CADET a behavioural synthesis sytem (Sycara & Navin chandra, 1989b; Navin chandra et al., 1991a). For example, a household water tap takes a material input (water) and outputs the water in response to a signal (open/close). Taking this a step further, a tap takes the input signal *theta*, and the input water flow rate of *Qin* and produces the output flow rate of *Qout*. The temperatures of the input and outputs are *Tin* and *Tout*. The tap may be represented as shown in Fig. 6. The figure shows the tap as a box with the following qualitative relationships: (1) the inflow of water (*Qin*) monotonically increases (*M+*) with the signal *theta*. (2) the inflow is equal to the outflow (*Qin* = *Qout*); (3) temperature does not change (*Tin* = *Tout*). The bottom two statements are qualitative boundary conditions: (4) when *theta* is zero, there is no flow through the tap; and (5) when *theta* is 2π, then the flow is maximum:

3. Representation of physical behaviour. Qualitative states provide a vocabulary for describing a device's physical behaviour. Transitions between the states in the vocabulary are expressed in the causal explanation. The qualitative states can be derived from the geometry through a special type of representation called the *Configuration Space*.

The configuration of a single object is a vector of six parameters, three positions and three orientations, that uniquely define the object's position and orientation in space. Consider a

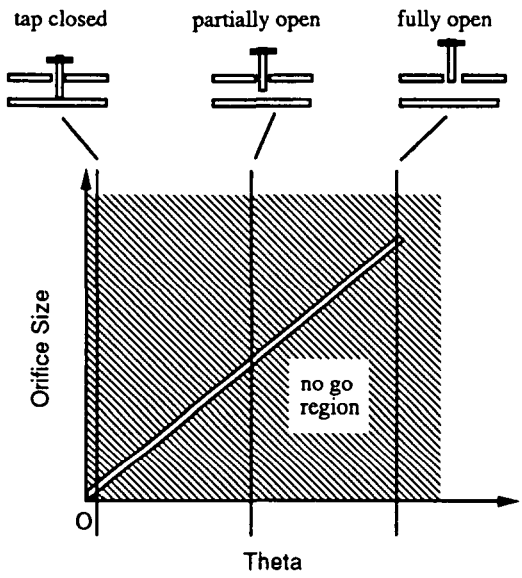

Figure 7 The configuration space and qualitative states of a tap

mechanism with two links. If regarded individually, the two links have a total of two times six, i.e., twelve degrees of freedom. However, because of the fact that two objects cannot overlap in space, some configurations for each link become illegal. The illegal region is also called the *no-go* region. The plot of all *go* and *no-go* regions for any two parameters of a mechanism is the configuration space plot of the mechanism (Lozano-Perez, 1983; Faltings, 1989; Joskowicz & Addanki, 1988). All regions within the configuration space represent qualitative states of the mechanism. The extreme points of regions in the configuration space correspond to the boundary conditions on behaviour.

For example, the tap can take three states that are qualitatively significant: close, partially open and fully open (Fig. 7). These states provide limit cases for qualitatively simulation. The figure shows the three qualitative states and their corresponding boundary conditions. These states are directly derivable from the configuration space, while the configuration space can be derived from the device geometry (Bourne, et al. 1989).

4. Structural features. Finally, one has to capture the visually prominent, structural features of an artifact. There is strong evidence in the cognitive science literature (Holyoak, 1987; Gentner, 1985) that, in a large number of experiments, although subjects rated relational similarity as the basis for judging the soundness of a match, the overwhelming majority of retrievals that occurred most readily were based on surface similarity.

The tap has several structural features (e.g. pipe, nozzle, cylinder) and relations (e.g. cylinder is across the pipe, cylinder is between the input and the nozzle). Structural relations that determine the visual form of the artifact are also used as indices (e.g. the relative positions of the nozzle and the cylinder). In the computer, structural features can be represented using a feature-based solid model (Dixon, 1988).

4 Indexing and retrieval

Indexing precedents in memory is one of the toughest issues facing systems that reason by analogy. As an illustration, try answering the following question: 'Think of ten things you can do with a spent printer ribbon-cartridge'. In answering this question, notice how you can retrieve precedents by using properties of the cartridge as cues into memory. Another question: 'Who is the most famous person you ever met?' In answering this question you have to search memory because it is highly unlikely that you have all meeting-episodes indexed in decreasing levels of 'famousness'. Here is a trace of the process a colleague followed in order to answer the question: 'Where could I have met

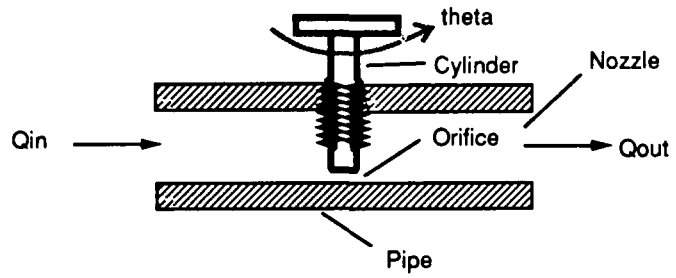

Figure 8 A water tap

famous people . . . Who are famous people, actors, scientists, politicians . . . Who are the scientists I have met? . . . Where could one meet scientists? . . . Whom did I meet at the last AAAI conference? . . . Who are the famous AI people? . . . and so on.' Memory has to be indexed such that precedents can be reached in many ways. In addition, the memory is constantly reorganizing indices and making generalizations (Schank, 1982).

The work of Kolodner on the CYRUS system provides a good indexing mechanism (Kolodner, 1981). CYRUS uses knowledge structures called Episodic Memory Organization Packets (EMOPS). An EMOP can have several episodes under it. The EMOP contains generalized information characterizing its episodes. The individual episodes are discriminated from each other by their differences. When many new episodes are added to an EMOP, then new sub-EMOPS are created. This process eventually forms a tree with the most generalized concept at the root with specific episodes residing at the leaves.

The EDISON system uses this discrimination tree strategy to organize devices (Dyer et al., 1986). In EDISON, devices are organized under a general index and then discriminated by their differences. Indexing is done on function, topology and context of use of the device. For example, a magnet and a suction cup are both methods of semi-permanent connection and are indexed under one node, they are however differentiated by the principles used; namely, magnetism and vacuum.

4.1 The chicken-n-egg problem of indexing

There are several known ways of indexing precedents/episodes based on their characteristics. The CYRUS program indexes episodes based on actual enumerable attributes of the episodes. In an analogical reasoning system, we earlier argued, there needs to be a method by which precedents can be retrieved to fulfil purposes that they were not originally intended for. For this reason, the notion of purpose-directed analogies was introduced (Kedar-Cabelli, 1985b). The approach is based on developing, on the fly, an explanation of why and how a given precedent is analogous to a given goal situation. For example, if one wanted to crack open a nut, and did not have a nutcracker, one could use a stapler instead. The question is, how does one recognize that the stapler can be used as a nutcracker? This can be done by developing, on the fly, an explanation of why the stapler has the characteristics of a nutcracker. This idea raises a very interesting indexing problem. How should the stapler be indexed in memory so that it can be retrieved when one is looking for a nutcracker? We cannot index a stapler under the index 'could also be used as a nutcracker', we cannot anticipate all possible uses of all the cases in memory.

To find relevant cases we need a case indexing mechanism that will allow the retrieval of cases that are analogically related to the current problem. The question is: as one can determine the analogical relevance of a case only after it is retrieved, how does one know which case to retrieve in the first place? This is the chicken and egg problem of memory indexing. Surprisingly, people are very good at this, a 'property of memory that always seemed technically mysterious is its uncanny aptitude for making metaphorical connections, of causing us to recollect things that turn out relevant only after reflection and reformulation' (Minsky, 1982).

One cannot go through memory looking for precedents that can be explained as serving some given purpose. While answering questions such as 'Think of all the things that can be used as a

nutcracker' we use a functional and structural description of a nutcracker to search memory with. A functional description might be based on functions and causal relations among the different parts of the nutcracker. Does this mean that memory should be organized not only by part attributes, but also by function and the types of causal relations among the different parts? Maybe so. Here is a question that illustrates the point about having causal relations as indices: 'Name five things that change colour when you touch or press them?' This question does not index on attributes but on some relation among attributes. A technique for answering questions that require indices that do not exist in memory is required. We have several choices: we can (1) either re-organize memory based on a question, (2) reformulate the question, (3) index memory with many redundant indices and hope for the best, (4) not retrieve the precedent at all, (5) or go through all the precedents in memory trying to explain each one.

This problem can be tackled through an index generation and transformation process. The idea works as follows: Let us suppose we want to find cases that match a given design problem. We can take the characteristics of the current problem and use it as an index into memory. For example, one might want to find a device that can proportionately control the flow in two valves. Let us assume that there are relevant cases in memory, but are not indexed in such a way that they will match the index being used. When the given index fails to find cases, it is transformed using a variety of techniques. For example, the given index may be transformed to 'find a device that causes proportionate motion in opposite directions'. This index may be further elaborated to 'up and down motion', leading to the retrieval of a see-saw as a relevant case. The index generation and transformation method works even when the relevant case is not properly indexed. Instead of having to anticipate all possible indices for a given case *a priori*, transformation techniques allow us to modify the question (index) instead. This is done dynamically, during problem solving.

Index transformation is a promising new approach. Most existing case-based systems use a predefined set of indices to access cases in memory. Such an indexing strategy is limiting since salient features of the current case which could constitute good indices may not directly match the pre-defined index set. Index transformation is a way to change the given salient features of the current problem to match the indices under which previous cases have been stored, thus making accessible to the problem solver previously inaccessible cases. Several techniques for index transformation have been proposed: elaboration (Kolodner, 1984), condensation (Kolodner, 1988), tweaking (Schank, 1986; Kass & Leake, 1988), modification (Murthy & Addanki, 1987), adaptation (Sycara, 1987), and causal decomposition (Navin chandra, 1988).

4.2 Asking the right questions: index generation and transformation

The creativity literature places a lot of importance on the role of analogy and metaphor in problem solving. A majority of the successful techniques are based on index generation and transformation, that is, asking the right question and modifying it in various ways. For example, synectics (Gordon, 1961) uses analogies to solve problems. The synectics process involves: making the strange familiar and making the familiar strange. Simply put, it involves making connections between the target problem and base concepts from conceptually distant parts of memory. To make the familiar strange is to distort, invert, or mutate the everyday ways of looking and responding to problems in an attempt to cause an useful reminding. Gordon has identified four index generation mechanisms for making the familiar strange, each metaphorical in character:

1 **Personal analogy.** Personal identification with the elements of a problem releases the individual from viewing the problem in terms of its previously analysed elements. For example, a chemist might imagine himself to be a molecule, permitting himself to be pushed and pulled by other molecules.
2 **Direct analogy.** This involves the direct comparison of parallel facts, knowledge, or technology. For example, Sir March Brunel solved the problem of underground construction by watching a shipworm tunnelling into timber. The worm constructed a tube for itself as it moved forward, this

led to the classical notion of caissons. To automate such an analogy, one needs to have a good way of representing processes qualitatively.

3 **Symbolic analogy.** Symbolic analogy uses objective and impersonal images to describe a problem. It is not clear yet how to automate the use of imagery and visualization in problem solving.

4 **Fantasy analogy.** This is based on expressing one's wish about what one would like to have. A part of the synectics approach is to ask the question 'How do we in our wildest fantasies desire the artifact to do?'

Techniques such as synectics are aimed at helping people be creative by inducing them to take different views of a problem and by drawing analogies. In order to make interesting analogies one needs to retrieve the right precedents. For which, one needs the right index into memory, such indices come from asking the right questions. Questions are the cues into memory. They provide the index/pattern to search memory with. It is through the process of posing the right questions and redefining them that one can retrieve useful precedents. Several techniques for generating and transforming questions (indices) have been developed. Here are a few:

Wishful thinking (Rickards, 1974). To fancifully think about some goal. The questions may be of the following forms:

"What I would really like to be able to do is . . ."
"If I could break all the constraints . . ."

Boundary examinations (Rickards, 1974). Starting with a statement of the problem, one picks up random phrases in the statement and asks why? For example, the statement: *How to develop the motorway network to allow for gradual replacement of rail by road transport as a consequence of relative lack of flexibility of the former?* is converted into the following questions (with respect to the underlined phrases): (1) *develop*: why develop this at all? Are we solving the right problem?; (2) *motorway replacement*: why motorways only? Why a network?; (3) *gradual replacement*: why only gradual? Why replace? Why not append?; (4) *lack of flexibility*: why is rail not flexible? Can it be made flexible? Questions such as these give us many new ideas as they produce indices to precedents not obvious with respect to the original statement of the problem.

Creativity by brainstorming (Osborn, 1953). Osborn suggests that brainstorming is improved by asking questions about the problem. The purpose of questions is to spur ideation. Here is a generic list of questions one can ask about almost any problem:

1 Can it be put to *other uses*? Are there new ways to use?
2 Can I *adapt*? What else is like this?
3 *Modify*? New twist?
4 *Magnify*? *Minify*? Longer? Larger? Condensed?
5 *Substitute*? Other ingredients? Other power source?
6 *Redefine*? What are the causes and effects? Can rearrange?
7 Can I *reverse*? Turn upside down?
8 *Combine*? Blend? Alloy?

Divergent thinking (Guilford, 1959). Divergent thinking involves deliberate attempts to not follow the beaten path. It requires the ability to rapidly produce ideas. This is related to another strain of thinking called *Ideonomy* (Wall St. Journal, 1 June 1987). Ideonomy is the science of laws of ideas and of the application of such laws to the generation of all possible ideas in connection with any subject, idea or thing. It is by mixing lists of natural phenomena and fallacies, for instance, that many questions can be generated. The inventor of ideonomy, Mr Gunkel, has developed a computer program that helps spew out combinations of ideas.

The underlying principle of all these techniques is that good ideas can be generated by having lots of ideas and throwing away the bad ones. Interestingly, many of the techniques use syntactic

methods for generating ideas. For example, boundary examination and ideonomy use purely syntactic reformulation of the problem or the questions related to the problem. There are few systems that implements some techniques of question formulation and reformulation discussed above. Such systems, though successful, are just scratching the surface of the problem of understanding human creativity:

Question transformation in CYRUS. The CYRUS program (Kolodner, 1980) used interesting techniques of question reformulation to search memory with. It is a question answering program that draws on an episodic knowledge base. The CYRUS program's memory consisted of news stories about one person, Cyrus Vance. It could answer questions about the news stories. For example, the question (adapted from Schank, 1986): *Has your (Cyrus Vance's) wife ever met Mrs Begin*? As CYRUS does not have information in memory to answer the above question, it has to transform the question using some domain knowledge.

Original Question (Q1): Has your wife ever met Mrs Begin?
(Q2): Where would they have met?
(Q3): Under what circumstances do diplomat's wives meet?
(Q4): Under what circumstances do diplomats meet?
(A4): On state visits to each other's countries and at international conferences.
(A3): When they accompany their husbands on these visits.
(Q3a): When did Vance go to Israel?
(Q3b): When did Begin go to the US?
(A3a/A3b): various dates can be retrieved from memory.
(Q3c): Did their wives accompany them on any of these trips?
(A3c): A few trips where this happened is found.
(Q2a): During what part of a trip would the wives have met?
(A2a): During a state dinner.
(A1): 24 May 1977, at a state dinner in Jerusalem. Both wives were present. They probably met.

The question reformulation process converts an un-answerable question into a series of questions that are more relevant to the data in memory. CYRUS's memory, as we discussed earlier in this survey, is in the form of a network of episodes discriminated by their differences. The difference is with respect to the direct attributes of the episodes. Attributes such as topic, participants, location, etc. In the future we will see indexing schemes based on relationships among attributes. Such organizations will be able to answer indirect questions such as 'What are all the things in memory that can be used as an ice-cream-scoop?'

Mutation retrieval. Gero and Rosenman (1989) have proposed a system based on prototypes. Prototypes store design experiences. The retrieval of prototypes is based on the closest match between actual and required behavioural attributes. If the prototype does not satisfy the requirements, then it is modified. If the existing prototypes are not capable of being modified to meet the requirements, other domains are searched for concepts possessing the required attributes. The mutation of prototypes involves the retrieval of other prototypes, based on the closest or at least a good, match between actual and required behaviour attributes.

Question tweaking in SWALE. SWALE is a understanding system that creatively retrieves and uses precedents (Schank, 1986). Given a story, the program starts by checking if the story fits memory, if not, it detects an anomaly, and then searches for an Explanation Pattern* (XP) that can explain the anomaly. Finally it attempts to apply the retrieved XPs. If the XP is not applicable, SWALE tweaks the XP to make it applicable. Finally, if an explanation is generated, it is integrated into memory and generalized.

*A precedent.

Consider the following input to the program: *Swale*, a successful 3-year old race horse, was found dead in his stall a week after winning the Belmont Stakes race.

The program, SWALE starts by trying to detect anomalies. It retrieves a RACE-HORSE script (Schank & Abelson, 1977) from memory which says that horses usually die many years after they retire from racing. SWALE finds a temporal anomaly. When the program detects an anomaly, it starts searching for an explanation pattern (XP) which can explain the anomaly. In order to be creative, SWALE searches both **routinely** and **unusually**. An unusual search is based on using unusual features of the anomaly as indices into memory. For example, the fact that Swale the racehorse is successful and young is used to find XPs with the index: 'death of the successful and young'. The following explanation pattern (XP) is retrieved:

The **Jim Fixx** XP:

1 Joggers jog a lot.
2 Jogging results in physical exhaustion because jogging is a kind of exertion.
3 Exertion results in exhaustion.
4 Physical exhaustion coupled with a heart defect can cause a heart attack.
5 Heart attack can cause death.

After an XP is retrieved, certain applicability checks are made. Next, the program retrieves several themes Swale participates in:

THEME1: Swale often has ACTOR role in the HORSE-RACE theme.
THEME2: Swale often has ACTOR role in the EATS-OATS theme.
THEME3: Swale often has ACTOR role in the SLEEP theme.

The program then tries to substitute themes and asks questions. For example, it finds a match between a RUN belief and the racing belief in the HORSE-RACE theme. This is because it finds a match between the JOGGING script and the RUN script. Finally, the program conjectures that Swale probably died of exhaustion as described in the retrieved Jim Fixx XP. In order to complete the train of reasoning, SWALE goes on to asking the next question: 'Could Swale have had a heart defect?' This question, in turn, is answered by finding additional XPs as described above.

It is through the process of asking direct and indirect questions, followed by tweaking, that SWALE can creatively explain situations. SWALE is one of the few systems that explicitly uses heuristics to adapt precedents to problems at hand. Another example is PERSUADER (Sycara, 1987).

5 Case matching

Issues relating to matching indices to case attributes has been an important part of case based reasoning research (Kolodner, 1980; Riesbeck & Schank, 1989). The simplest form of case matching is based on finding common attributes between the base and target. If a perfect match is not found, partial matches are considered. For example, the ratio of matching attributes to the total number of attributes may be used as a measure of closeness. There are several such heuristic measures that may be used and are often domain dependent. For example, if the attributes have numerical values, then closeness may be measured as the distance between base and target in N-dimentional space, where each dimension corresponds to an attribute in the case. The attributes may also be weighted to signify importance.

In addition to partial matching, one may also use inexact matching. Let's assume that information in the cases are represented as triples. There are essentially three popular types of triples used in design representations:

1 **Object Attribute Value**, for example (table-top material wood)
2 **Object Relation Object**, for example (table next-to sofa)
3 **Attribute Relation Attribute**, for example (density determines weight)

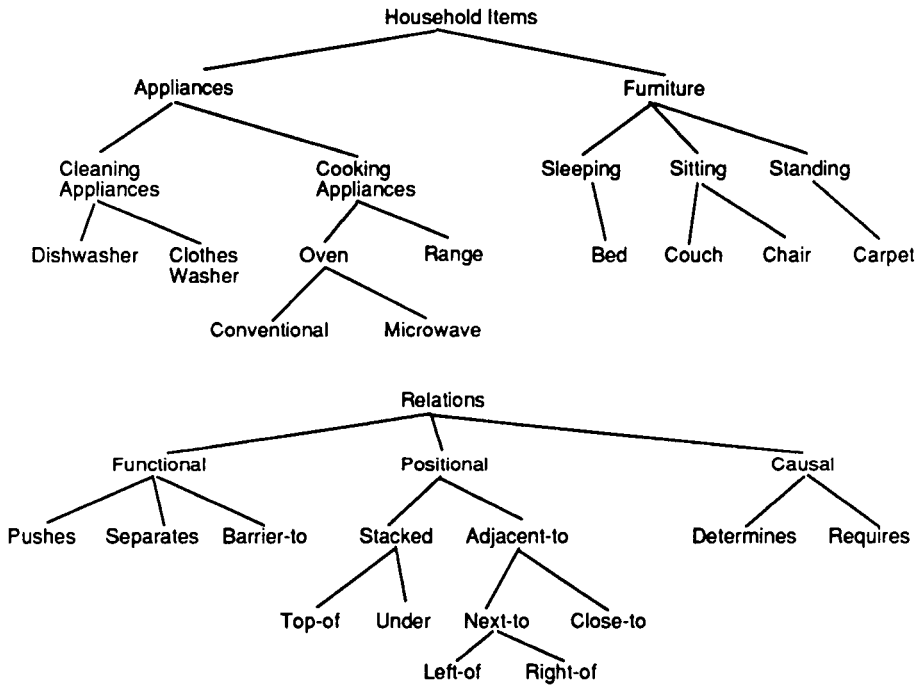

Figure 9 Object hierarchies serve as thesaurus and similarity metric

These data structures can be matched either exactly or through an abstraction. For example, the relation (*stove left-of dishwasher*) could generalize, at the object level, to match the index: (*cooking-appliance left-of cleaning-appliance*), or it could generalize, at the relationship level, to (*stove adjacent-to dishwasher*). Object and concept hierarchies are used to make such generalizations (Navin chandra, 1988). Figure 9 shows how simple it is to create such hierarchies, objects that have fewer links between them are considered more similar.

Higher order matching, such as *structure mapping* (Gentner & Toupin, 1986) can also be achieved with triplet matching. This is done by only matching causal relations in the base and target, and by making generalizations only on objects and attributes, not on relations. This kind of matching strategy yields analogies (Navin chandra et al., 1987).

The nature of the database also plays an important role in determining the results of an associative design system. In our experiments with design automation systems that use analogical reasoning, we have found that innovative behaviour can be attained by introducing the right kind of precedents in the system's database (Navin chandra et al., 1987; Navin chandra, 1988). When we first implemented an analogical matching algorithm, we gave the program a database of previous designs all taken from the same domain. The program consequently produced solutions in the P-space. We later replaced the designs in the database with designs from other domains, we also introduced some non-design related episodes. Using the new database, the program (without any re-coding) started generating unexpectedly interesting solutions! This confirmed our hypothesis (and suspicion) that the perception of a design being innovative is not due to some intrinsic property of the design or the process that generated it, but lies in the eyes of the beholder. When our matching program was using intro-cultural precedents only, the results appeared mundane; but later appeared innovative when a new database was used. The program does not differentiate between the two databases, it performs the same symbolic operations each time. This is one of the most effective approaches (or tricks, if you will) to making a system exhibit innovative behaviour.

6 Design synthesis by association

This section deals with issues relating to how cases get used in synthesizing new designs. Synthesis is the final step in reasoning by association. Synthesis lies at the heart of design, it is the process by

which an artifact is refined or generated. The process by which the artifact actually takes form. In order to perform a synthesis, one has to have things to synthesize. The synthesis process involves first finding what to synthesize, then deciding how to synthesize, followed by an evaluation.

In many systems, the alternatives available for synthesis are known *a priori*. For example, the building design system ALL-RISE (Sriram, 1986) and the mechanical design system MEET (Steinberg et al., 1986). In goal directed systems the programs generate lists of possible candidates for synthesis at each stage of the design process. A design problem is broken into subfunctions and components that fulfil the different subfunctions are retrieved. Conventionally, these components could be retrieved from a global database or local list attached directly to a refinement operator. In an analogical reasoning system the synthesis alternatives could be drawn from within and without the current design culture. Where, each subfunction provides us with a purpose or index to search memory with. After precedent cases are retrieved, parts of the cases are extracted and synthesized into new designs. This transfer of knowledge is one of the least investigated problems in design automation and among the hardest. Work on case-based planning has shown how plan snippets can be retrieved and incorporated in new plans. This work on snippet synthesis has been developed in problem domains where the representation of the case and the solution is of the same type. In mechanical design, however, decisions relating to how certain functions are achieved might be taken at a linguistic or qualitative level. Considerable complication arises from the fact that although a design might be verified to be correct at these levels, simulation at the physical level might fail. We have to find ways in which the problem solver can synthesize snippets at one level of abstraction while making sure the parts will work together in physically correct ways (Sycara & Navin chandra, 1989b). A further complication is that the parts (also called case 'snippets') may not be from the same domain and may need substantial modifications before synthesis. This is the cross-contextual (cross-cultural) synthesis problem.

This section reviews a variety of techniques that are used in innovative design systems. They range from attribute-based matching and synthesis to representations of behaviour, function and causality.

6.1 Attribute based case indexing and retrieval

STRUPLE is a building design system that finds relevant designs using an attribute based similarity metric (Maher & Zhao, 1986, 1987). The program accepts a description of the building to be designed, searches relevant past experiences, and outputs the relevant design vocabulary to be used in synthesis. Experiences are stored in the form of descriptions of building design solutions. The representation includes: *geometric information*: gross area, height, width, aspect ratio, overall shape, and shape irregularity; *loading information*: dead load, live load, wind load, and seismic zone; *primary three-dimensional systems*: tube, tube-in-tube, core, core and suspension, bundled tube; *lateral two-dimensional systems*: rigid frames, braced frames, solid walls, and staggered truss.

STRUPLE finds buildings that are similar to the current building by using a similarity metric. The metric is described by a set of criteria that define what significant common aspects the matching buildings and the current building should share. A matching criterion is a requirement of similarity imposed on a feature of a matching building. For example, the number of stories, the intended use, and the design wind load. The criteria are divided into three classes: required criteria, desired criteria, and no-match criteria. A criterion is defined as required or desired depending on the extent to which it would aid in determining the structural system for the current building. For instance, when a building has over 30 stories, the lateral system design will more likely govern the design, or at least be as important as the gravity system. In this case, wind load is chosen to be a required criterion. If the number of stories of the current building is in the range of 5 to 30, wind load will not be as important a factor and is defined as a desired criteria. When the building has less than 5 stories, wind load will not govern the design, so it is defined as a no-match criterion. After matching buildings are found, knowledge is transferred from the prior designs to the current design problem. The program finds building elements in the prior designs that can be used in the current

design. If there are many alternative elements, STRUPLE orders them based on frequency of use. This way of prioritizing alternatives prefers elements that are frequently used in buildings similar to the current one.

STRUPLE's similarity metric is based on a fixed set of criteria that does not consider the causal rationale involved in the decision process. It is not systematicity based. For this reason the program cannot analogize across domains, but can be very effective if problems are limited to one domain.

Another architectural design system that uses a case-based approach is ARCHIE. The system works in the domain of office buildings (Goel et al., 1991). It also uses a flat, frame based representation of cases like STRUPLE (there is no deep reasoning about shape and form). ARCHIE's contribution lies inits use of qualitative domain models for retrieving cases. For example, it has a model of how various features of an office space (e.g. wall colour, lighting quality) affect the lighting quality of the built environment. Such a model can be used to evaluate a design concept and to retrieve all prior cases where a similar problem was encountered.

6.2 Synthesizing design plans

When solving a new design problem by analogy to a previous case, we might transfer the solution used in the base to solve the target problem. In some cases, instead of transferring the final solution from base to target, it is better to transfer the design plan (process) from base to target. This approach was suggested by Carbonell (1983a). The first method transfers the actual steps performed from base to target. This is called a *Transformational analogy*. The second, improved method, transfers the reasoning process used in the base to the target problem. This is called *derivational analogy* as it uses the underlying reasoning steps in the base and target. To find a match the system first starts solving the target, after some progress is made, the reasoning steps in the target are matched against those of the base. If the match is found, the rest of the plan from the base is transferred to the target. This method, though a desirable capacity for design automation systems, needs improvement. It is not clear how far the target should be solved before the analogy can be drawn and the rest of the plan can be transferred from the base to the target. This issue, among others, is discussed in Kedar-Cabelli (1985c) and Mostow (1986).

The idea of using design plans has been applied to VLSI design in a system called Argo (Huhns, 1987). Argo uses a hybrid mechanism borrowing ideas from MACROPS of STRIPS (Fikes & Nilsson, 1971) and the Explanation-based Generalization method (Mitchell et al., 1986). The Argo program learns operators from solving design problems. Whenever it solves a design problem, Argo creates a tree of the rules used in the solution. The tree is represented as a rule-dependency graph (RDG). Macro-operators are generated by dropping the most specific rules from the RDG. By doing this many times over, Argo regresses through the actual rules of the plan. In so doing, the pre- and post-conditions of the macro-operators end up with variables in their patterns rather than references to specific objects.

While using these operators, the program starts with the most specific macro-op, failing which, it proceeds to use more and more abstract macro-ops. As the system uses generalized operators it will be difficult for it to recover from the wrong application of macro-ops. There is no guarantee that in all domains MACROPS (derived the way Argo does) will work even if all preconditions are tested before application. However, Argo seems to run well in the domain it was built for.

6.3 Qualitative motion synthesis

Mechanical designs can be viewed as being synthesized from conceptual building blocks that perform specific kinematic functions (Kota, 1990). Given spatial relations among the inputs and outputs of a device, it is possible to find a configuration of the basic building blocks that will realize the desired function. We now examine a system that synthesizes mechanisms from parts of previously known mechanisms. The program is able to make associations for synthesis through a qualitative representation that captures the motion of the mechanism.

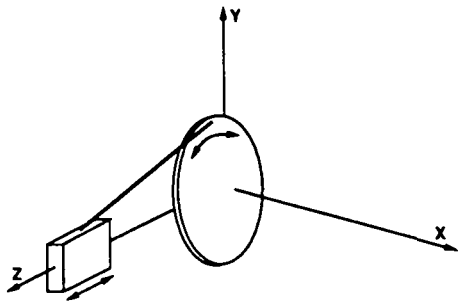

Figure 10 Crank mechanism

The representation is based on matrices that relate input motions to outputs. Consider, for example, the slider crank mechanism (shown in Fig. 10). There are two ways in which one could use the mechanism: one could either rotate the crank and treat the slider movement as an output, or one could move the slider and treat the crank's movement as output. In this case, the crank can be rotated only about the x-axis and the slider can be moved along the y and z axes.

The resulting outputs are represented in an incidence matrix called the Motion Transformation Matrix (MTM). The matrix for the slider and crank mechanism is shown below:

	Rx	Ry	Rz	Tz	Ty	Tz
Rx	0	0	0	0	1	1
Ry	0	0	0	1	0	1
Rz	0	0	0	1	1	0
Tx	0	1	1	0	0	0
Ty	1	0	1	0	0	0
Tz	1	1	0	0	0	0

The column elements are inputs and the rows are outputs. The R's represent rotations about the three x, y and z axes, and the T's represent translations along the three axes. Consider the first column: a crank in the y-z plane can the rotated about the x-axis (Rx), the slider can move anywhere in the y-z plant (Ty and Tz). This relationship is indicated by non-zero elements. The matrix represents the basic notion of a slider and a crank for various orientations of the crank: rotation about x, y, and z axes.

Let us consider a synthesis example. The system is given the task of converting a rotation about the *x-axis (Rx)* to a rotation and translation about some axis a, that lies in the x-zplane. Let us call this vector *Ha*. This specification can be written as follows: $Rx \rightarrow Ha$. The specification can be converted into the following decomposition: $Rx \rightarrow Ty \rightarrow Ta \rightarrow Ha$, which corresponds to the following three matrices:

0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 1 0 1
0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0
0 0 0 0 0 0	0 0 0 0 0 0	0 0 0 1 0 1
0 0 0 0 0 0	0 0 0 0 1 0	0 0 0 1 0 1
1 0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0 0
0 0 0 0 0 0	0 0 0 0 1 0	0 0 0 1 0 1

The decomposition is done using heuristics that know how to convert a vector into an ordered set of ortho-normal vectors. The heuristics are based on the existing building blocks the system knows about. This prevents the system from producing spurious un-realizable alternatives. It does, however, make a closed-world assumption on the primitives it knows about; assuming all new designs are a combination of known primitives. The matrices shown above correspond to various building blocks that can be configured into a design as shown below. The top half of the figure shows the three primitives corresponding to the three matrices. The synthesized design is shown in the bottom half of Fig. 11.

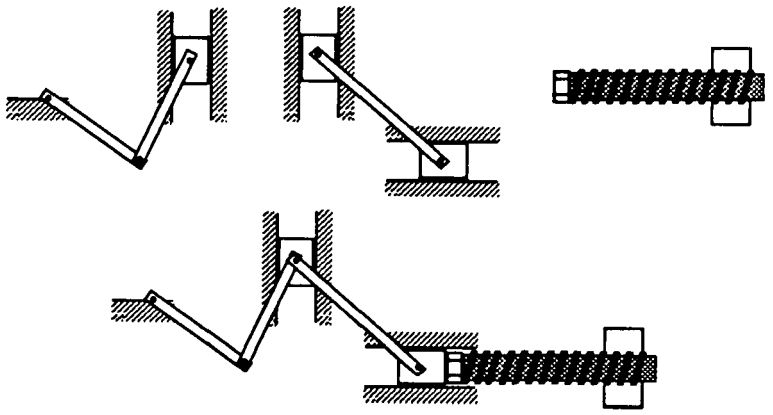

Figure 11

The notion synthesis approach provides a method for recognizing a given behaviour in terms of known primitive behaviours. This is one of the first formalized ways of viewing design as the synthesis of kinematic processes. This approach, however, is not suited to intermittent mechanisms where the connectivity of kinematic pairs change during the operation of the device.

6.4 KRITIK: a deep model-based design system

KRITIK is a deep model based design system (Goel, 1989). It uses a component-substance model that captures the components (e.g., battery, pipe), substances (e.g., water, electricity) and relations (e.g. containment, connection). Behaviour of such systems are represented as graphs of states and transitions. When the system is given a design task, it retrieves the best case and deduces modifications that can be made. Modifications involve changes in relations, substitution of substances, parametric changes of components.

6.5 Qualitative synthesis with interactions

The notion of synthesizing devices from known components is extended beyond basic kinematics in the Ibis system (Williams, 1989, 1990). In Ibis, components are represented as sets of interactions among behaviour parameters of the component. This approach allows one to use any aspect of a behaviour, not restricting behaviour descriptions to just one domain (e.g., qualitative motion synthesis). In other words, Ibis has the ability to make associations across domain boundaries. For this reason, it is capable of producing innovative solutions.

Let us start by examining the representation used in Ibis. A simple pipe, for example, is a device that takes some input flow (*Qin*) and produces some output flow (*Qout*). The flow rates depend on the resistance of the pipe (*R*) and the pressure difference across the pipe (*Pd*). The relationship among these parameters is called an *interaction*. The interaction model of the pipe is shown in Fig. 12.

In addition to primitive devices, the interaction based model is used to capture domain principles. Consider the examples in Fig. 13. A container for fluids can be modelled with two

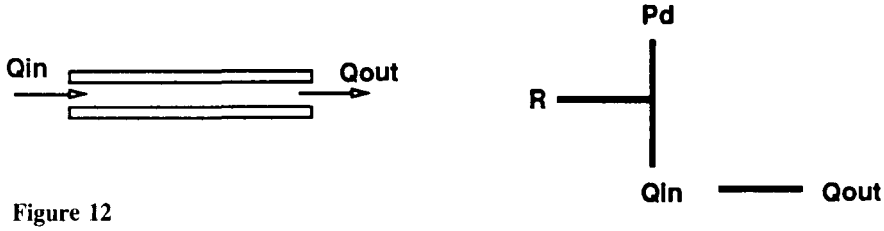

Figure 12

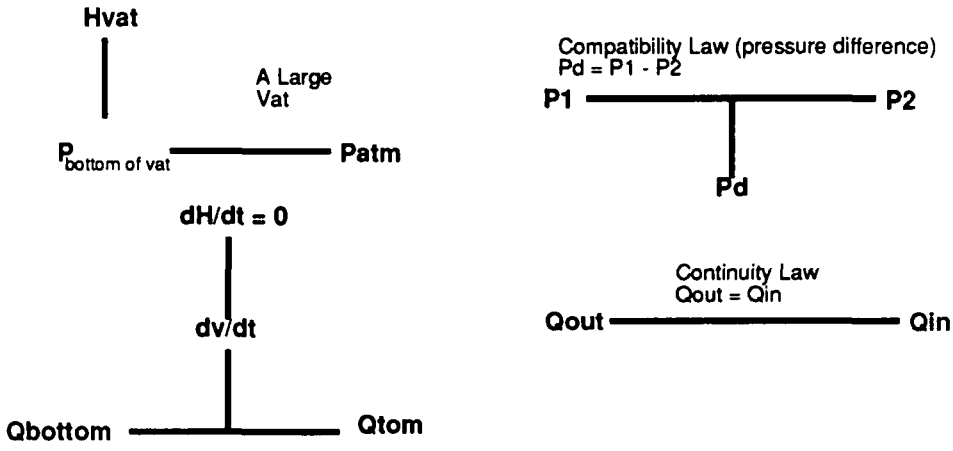

Figure 13

interactions: (1) the rate of change of fluid in the container (dV/dt) is determined by the rate of flow of fluid into the container (*Qtop*) and the rate of flow of fluid out the bottom of the container (*Qbottom*); (2) the pressure at the bottom of the container (*Pbottom*) is determined by the height of fluid in the container H and the atmosphere pressure (*Patm*). The container represented in Fig. 13 is actually a very large *vat* in which the height of the liquid does not change over time ($dH/dt = 0$).

Figure 13 also shows two domain rules. The conservation of mass rule states that the inflow (*Qin*) into and outflow (*Qout*) from a node are equal. The compatibility rule states that the pressure difference at a node (*Pd*) is the difference of the pressures at the node $P1$ and $P2$: $Pd = P1 - P2$.

Consider the following example. Given a large Vat and a Punch bowl, one has to design a device which will maintain the height of the fluid in the bowl even when fluids are removed from the bowl. The specification is given as:

$$Hvat - Hbowl = d(Hbowl/dt)$$

This means that the rate of change of the height of fluid in the bowl is a direct function of the height difference between the bowl and the vat. The bowl and vats are defined as containers. The vat, being very large, has a predetermined parameter: $dH/dt = 0$. In other words its level does not change.

The nodes on the interaction graphs are treated as terminals of the components. The system designs by connecting compatible nodes to produce a working design. The program starts by building a graph of all possible connections among the given device models. This is done by placing *links* between all the terminals that can be connected. For example, all flow rates could be connected to all other flow rates. Figure 14 shows all the known interactions and links among them. The links are shown in fine lines.

The design is generated by finding a path through the interactions and links that will satisfy the specified behaviour. In this case, the system has to find a relationship between the rate of change of height of fluid in the bowl ($dHbowl/dt$) and the difference in the heights of the vat and the bowl. Ibis looks for a path that terminates only on constants (e.g., *Patm* and *Qtop* of the bowl), unless the variable is a desired variable. In this design example, the desired variables are: ($dHbowl/dt$), *Hbowl*, and *Hvat*. The grey line in the figure above represents a path through the given interactions. The resulting design is the pipe attached between the bottom of the bowl and the bottom of the vat.

The interaction approach goes beyond motion synthesis by including higher order relations such as pressure and rates of change. The program also explicitly incorporates physical laws and principles. A major advantage of Ibis is that it can handle feedback mechanism. Ibis, however, has one major drawback. Its ability to solve a problem depends on the syntactic form of the interactions

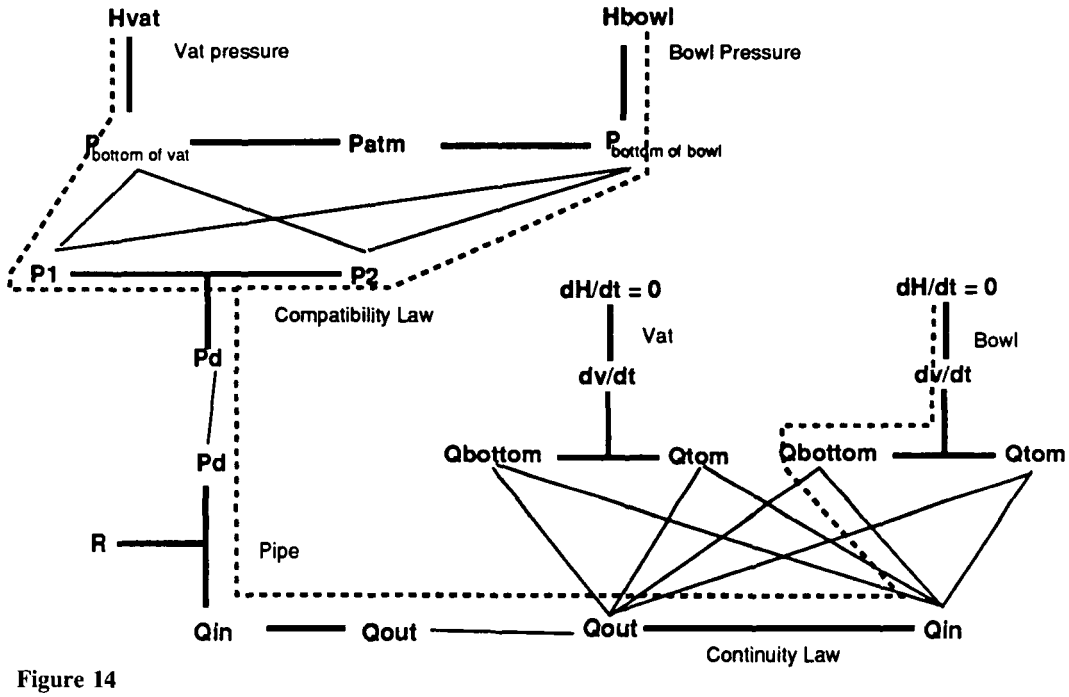

Figure 14

it is given. For example, the bowl problem could have been solved by attaching the *Qbottom* of the bowl directly to the *Qbottom* of the vat. Ibis will not find the solution.

The associative search space can be expanded beyond the immediately available prior designs through a transformation based technique.

6.6 Transforming and indexing on behaviour descriptions

In the design domain, index transformation can be used to find cases that are behaviourally equivalent to a given problem specification (Navin chandra et al., 1991b; Sycara & Navin chandra, 1991). Behavioural equivalence means that one behavioural description should be derivable from the other. We now examine a system CADET, that elaborates given specifications into more detailed specifications. The elaboration technique is based on a qualitative calculus that adds details to a device specification without altering its behaviour. In this way, the system generates alternatives that are behaviourally equivalent to the original specification, but are more detailed and thereby better suited for case retrieval.

For example, consider the design of a device which controls the flow of water into a flush tank. The behaviour can be specified as follows: *as the depth of water (D) in the tank increases, the rate of flow of water into the tank (Q) should decrease*. This statement is expressed as an influence: $Q \xleftarrow{-} D$. Which means that the depth negatively influences the flow rate. This specification may be used as a set of indices to find relevant cases in memory. If there are no cases that directly match the specifications, then it would be useful to consider using parts of several cases, if possible. In this instance, a relevant case could be the hot-cold water faucet shown in Fig. 15. The faucet is specified as a device which allows for the independent control of the temperature and flow rate of water by appropriately mixing hot and cold water. The relevance of the faucet case to the design of the flush tank is not obvious. By extracting portions of the faucet, such as the see-saw part, it is possible to design the flush tank device as shown in Fig. 17.

To accomplish the task of extracting relevant portions of a design, the problem solver must be able to recognize that the flush tank's behaviour can be achieved by combining relevant 'sub-behaviours' some of which are also present in the faucet. In other words, the relevant capability of the problem solver is to recognize shared 'sub-behaviours' among devices that are not given *a priori*. This is done by applying the following transformation rules:

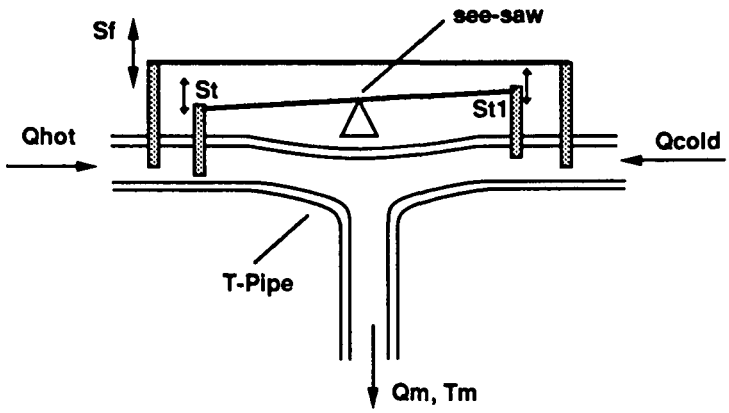

Figure 15 A hot-cold water faucet

Design Rule 1. If the goal is to have x influence z, and if it is known *a priori* that u influences z, then the goal could be achieved by making x influence u.

Design Rule 2. If the goal is to have x influence z and if it is known *a priori* that some two quantities p and q influence z, then the goal could be achieved by making x influence p or q, or both.

The design rules are used to transform goal influences into more elaborate influence graphs. The idea is to hypothesize new influences and to then find cases that may be used to achieve those influences. As new variables are introduced, corresponding new influences are hypothesized. After hypothesizing influences, the case base is used to find prior designs that may embody some physical law or principle that matches the hypothesized influence. Using this approach, CADET often retrieves cases from outside the current design domain that are analogically related to the current design problem. This approach is able to solve design problems by drawing analogies to prior designs and by exploiting physical laws and effects other than those included in the given domain description.

Returning to the flush tank example, the original specification can be transformed as follows: two transformations of the influence, $(Q \leftarrow D)$ yields the following three sets of influences.

1 $Q \leftarrow Var2 \leftarrow Var1 \leftarrow D$
2 $Q \leftarrow Var2 \leftarrow Var1 \leftarrow D$
3 $Q \leftarrow Var2 \leftarrow Var1 \leftarrow D$

Consider the second set of influences. The influence $Var2 \leftarrow Var1$ can be matched to the see-saw influence $X2 \leftarrow X1$ which binds the two unknown variables. The influence $Q \leftarrow X2$ matches the tap. The remaining influence $X1 \leftarrow D$ matches a float (Fig. 16).

The resulting design is shown in Fig. 17. It outputs a list of cases or case pieces with information on how they should be connected.

Through the process of influence hypothesis and matching, the system is able to use physical laws and principles embedded in prior design cases to achieve its current goals. In this way, CADET is opportunistic about the principles it exploits in a design. This is unlike other approaches which assume that all the relevant principles have been identified *a priori*, as in the Ibis system (Williams, 1990). Because CADET hypothesizes influences, it does not limit itself to the given

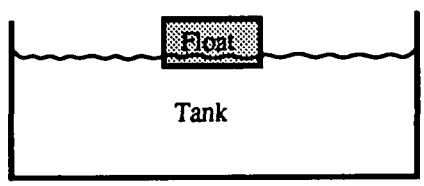

Figure 16 A simple float

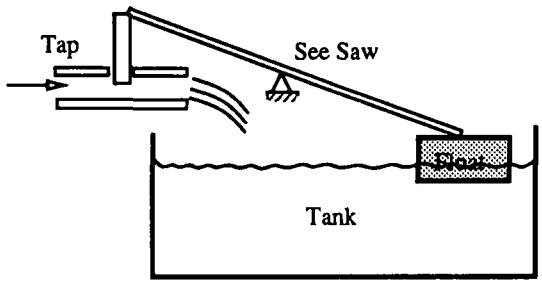

Figure 17 A flush tank

knowledge. Consequently, it can generate elaborations that represent designs that have never been conceived of before. Further, CADET's ability to recognise behavioural equivalences reduce its sensitivity to the form of the problem description.

6.7 Question transformation based on causal explanations

A landscape design system, CYCLOPS, uses a technique called demand posting which, in spirit, is similar to what CYRUS did. It was the first application of the case based reasoning methodology to design (Navin chandra, 1987). The program starts by posting the top-level goal and tries to retrieve matching cases in memory. If this fails, it reformulates the question by using a causal explanation of the problem it is trying to solve. CYCLOPS uses a tree-like structure to keep track of the questions and precedents it uses along the way. Details on the implementation of the above question transformation process may be found in Navin chandra (1988, 1991).

Finding cases relevant to a debugging task involves setting up appropriate cues and finding cases which match the cues. CYCLOPS approaches these two fundamental tasks using explanation-based methods. The hypothesis is that case-based debugging for design problems can be controlled through a process of asking relevant questions and modifying them based on a causal explanation of the bug. These questions serve as cues into memory (Schank, 1986). When a bug is found, a corresponding question is posed to the case knowledge base (CKB): 'Has this, or some similar bug, been seen before? Is there a known way of repairing it?' If a relevant case is not found, the reasons for the bug are used to transform the question. If, for a given bug X, related cases are not found, the debugger then goes on to ask 'What are the causes of X?' 'If it is not known how to eliminate X, can its causes be eliminated?' This questioning process is recursively applied until a relevant case is found. The implemented version of this technique is called *dependency tracing*. Let us consider a simple example: imagine that you have just built a tree-house on the branch of a large tree. However, you notice that the branch is sagging too much, you ask yourself the question: 'Can I think of some way of reducing the sagging of the branch?' Assume no solution comes to mind, the next step is to find the causes of the bug. There are several reasons for the problem: the tree-house may be too heavy, the branch too weak in bending, or the moment arm is too large. Taking any of these causes, a new question is posed: 'Is there a way of making a branch stronger in bending?' This question could retrieve a case about how steel ropes are used to reduce the bending of construction cranes. Steel ropes can be similarly used to solve the 'sagging-branch' bug. A related hypothesis to the dependency tracing idea is that, the better one's explanation of a bug, the better one's chances of finding a solution. In addition, if one cannot explain a new problem, one cannot solve it.

The approach is directly based on Osborn's work on Brainstorming (Osborn, 1953). He suggests that problem solving through questioning (self-interrogation generates remindings that lead to creative ideas. In this book, *Applied Imagination*, he suggests that one of the ways of brainstorming is to redefine the problem question by finding and addressing the causes and effects of the problem. In other words, if you cannot think of a way to solve the given problem directly, try addressing its causes. CYCLOPS implements this question transformation heuristic. An even more comprehensive set of heuristics for problem solving by asking the right questions is presented by Schank (1986,

1988). It is shown that *tweaking* remindings can yield unexplored questions and lead to creative problem solving.

Consider the following example about the cyclops design problem solving system trying to locate a house on a steep slope. The problem solving process is composed of the following steps:

1. Problem recognition. The current state of the design artifact is posted to the case knowledge base for comments. If any case about a previously encountered bug matches, either directly or through an abstraction, it is retrieved and applied. This is shown in the following scenario:

A problem solver is attempting to locate a house on a steep hillside. It decides to place the house directly on the ground. As soon as it does this, it is reminded of a situation in which it had placed a green-coloured wooden block on an incline and the block has become tilted. Its explanation of the situation was that the block was tilted because it was on a sloped surface. Using this explanation it infers that the house will also be tilted. It then uses a domain rule to infer that tilted houses are problematic and that the bug needs to be repaired.

2. Explaining the bug. All rules and cases in the system contain fossilized explanations (Kass & Leake, 1988) which provide the cause of the bug or justify the associated repair strategy. Based on the type of match performed above, the explanation in the case is appropriately modified and copied over the current bug.

The problem solver finds reasons for the bug by going back to the explanation of the green-block case it used earlier. It explains the house's tilt as being due to two facts: (a) the house is on the ground and, (b) the ground is sloped.

3. Retrieving and applying a case. The problem solver first tries to find a case which relates to the bug itself. If such a case is not found, the explanation of the bug is examined to find underlying causes ('sub-bugs'). These new 'sub-bugs' are, in effect, sub-goals to the main goal of repairing the bug. In our example, when the program used the block case to recognize that the house will be tilted, it keeps track of the explanation provided in the case.

The problem solver sets up the goal of answering the question: 'Is there any way of getting rid of the house's tilt?' Using this goal as a cue, it finds a case about how the ground can be excavated to provide a flat site for the house. Let us assume the problem solver is not satisfied with this solution because it generates new bugs relating to soil erosion. On finding no other relevant case, the problem solver uses the causes of the bug to set up two new disjunctive sub-goals: (a) 'Is there any way of not having the house on the ground?', or, (b) 'Is there any way of making the ground not so steep?' While examining the first sub-goal, the problem solver remembers seeing pictures of village homes in Thailand. The pictures showed huts on stilts. The corresponding explanation is that the villagers, who were having problems with flooding and ground dampness, found a way of getting their homes off the ground: they put their huts on stilts! Drawing from this explanation, the problem solver decides to put its house on stilts too.

4. Looking for other bugs. As soon as a repair strategy is applied, the debugger looks for new bugs which might have been generated by the repair. In addition, it can also happen that the bug does not get completely eliminated and has to be repaired again. Sometimes there may be multiple bugs which have to be solved. In our example, let us assume the problem solver finds a case which states that using stilts requires the ground to be firm. A new question (*is ground firm*)? is posted to the CKB. The rest of the problem solving process proceeds like this:

In order to achieve the sub-goal (is ground firm) the problem solver looks for precedent cases which contain the above sub-goal in their descriptions or explanations. It then finds a case about the use of piles to make landfills strong enough to support buildings. One of the sub-goals of using piles is to make the landfill's soil firm. Using this precedent, the problem solver decides to drive piles into the ground. After solving the problem in this way, it re-invokes the process to find whether any new problems are generated. If no problems are found, the solution is deemed correct.

The debugger follows the above steps recursively to find solutions. In an actual run, the program generates many solutions each of which may involve several repair strategies drawn from different cases. These cases can be from a variety of sources: from the domain of the current design problem

(direct match), from a similar domain (abstract match), or from a completely different domain (analogical match). Our current example is simple, but illustrative of the debugger's ability to propose solutions based on past experiences instead of relying solely on domain heuristics.

7 Conclusions

> 'Creativity is not such a mysterious process. It depends upon having a stock set of explanations and some heuristics for finding them at the right time, and for tweaking them after they have been found . . . Search and adaptation of patterns are two of the biggest problems facing AI.' (Schank, 1986)

Innovative design systems should have the ability to consider a wide variety of design alternatives (explore) and the ability to solve design problems by using past experiences drawn from within or without the current design domain. In this paper we have examined the latter—the ability to make associations between a current task and prior solutions. We argued that the farther a system reaches to use knowledge that is outside the current domain of problem solving, the more likely that the final solution will appear innovative. In the examples examined, we have also shown that bisociation can fruitfully occur within a domain. As long as the association is made between habitually different designs, we can expect to produce an innovative combination.

To automate this process, the first issue we have to deal with is representation. It is important that the representation we use is capable of capturing the salient features of the design. The representation should also allow for similarity recognition, it should not limit the different ways in which a design can be viewed. For example, a coffee cup could be simply represented with the symbol 'coffee-cup'. This representation, however, limits the applications of the cup. In contrast, a representation that captures the shape, weight and size of the cup could be used to recognize the cup as a possible projectile, an ice-cream scoop or a receptacle to hold pencils.

Representations are not limited to physical features, they can also be used to capture behaviour or causality. These representations serve as abstractions of the real design. A particular abstraction is chosen to suit the goals of the design task. For example, in debugging, an explanation-based abstraction is most appropriate.

The representations should also be easy to index and match. A wide variety of indexing and matching techniques are being used in design automation systems. The aim of indexing is to provide easy access to relevant cases for a given query. It is relatively straightforward to index cases when one is looking for simple matches between base and target. In systems that reason analogically, the indexing scheme has to allow for matches that are based on similarity. Such similarities are often not at the surface level, but at a deeper more fundamental level. The base can 'look' quite different from the target, but still be relevant. This complicates the indexing task; how does one index a case without prior knowledge of how it will be re-used in future? We called this the chicken-n-egg problem of indexing. Given that we can determine the relevance of a case only after it has been retrieved in memory, how does one retrieve it is the first place? This problem has been addressed by transformation techniques. The idea is to transform the index which is used to retrieve the case from memory. The brainstorming and creativity literature is full of heuristic techniques for transforming the initial question. It is hoped that the transformed question will trigger previously unknown indices. Some of these approaches can and have been used in computer programs.

After precedent cases are retrieved, parts of the cases are extracted and synthesized into new designs. This transfer of knowledge is one of the least investigated problems in design automation and among the hardest. In many design domains decisions relating to how certain functions are achieved might be taken at a linguistic or qualitative level. Considerable complication arises from the fact that although a design might be verified to be correct at these levels, simulation at the physical level might fail. We have to find ways in which the problem solver can synthesize pieces of prior cases at one level of abstraction while making sure the parts will work together in physically correct ways. A further complication is that the parts may not be from the same domain and may

need substantial modifications before synthesis. This is the cross-contextual (cross-cultural) synthesis problem.

This paper has looked at associative techniques in innovative design. We have seen how new designs can be generated by drawing ideas from previous designs, experiences, or objects in the world around us. The basic idea is to find and use relevant prior art that is either directly or analogically related to the current context. We argued that having a large memory of cases, drawn from many different domains, can make the resulting solution appear innovative. The other fundamental ingredient of innovative design is exploration. The aim of exploration is to produce a large variety of design alternatives through an algorithmic, generative approach. The idea is not to waste one's effort in trying to create innovative designs, but to generate lots of alternatives and to simply throw away the bad ones. Computational approaches to systematic exploration of design spaces is the topic of the second part of this two-part paper.

References

Bourne, D, Navin chandra, D and Ramaswamy, R, 1989. "Relative tolerances and kinematic behaviour" In: J Gero, editor, *AI in Design*, Computational Mechanics.

Brown, DC, and Chandrasekaran, B, 1984. "Expert systems for a class of mechanical design activity" In: *Proceedings of IFIP W. G. 5.2 Working Conference on Knowledge Engineering in Computer-Aided Design*, Budapest, Hungary.

Brown DC and Chandrasekaran, B, 1986. "Expert systems for a class of mechanical design activity" In: D Sriram and B Adey, editors, *Proceedings of the First International Conference on AI applications in Engineering*, Computational Mechanics.

Carbonnell, JG, 1983. "Learning by analogy: Formulating and generalizing plans from past experience" In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Tioga Press.

Carbonnell, JG, 1983b. "Deprivational analogy and its role in problem solving" In: *Proceedings of AAAI-83*, pp 64–69.

Carbonnell, JG, 1986. "Deprivational analogy: A theory of reconstructive problem solving and expertise acquisition" In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach Vol 2*, Morgan Kaufman.

Dixon, JR, 1985. *Design Engineering: Inventiveness, Analysis and Decision Making* McGraw Hill.

Dixon, JR, 1988. "Designing with features: Building manufacturing knowledge into more intelligent CAD systems" In: *Proceedings of ASME Manufacturing International-88* Atlanta GA, April 17–20.

Dyer, MG, Flowers, M and Hodges, J, 1986. "EDISON: An Engineering Design Invention System Operating Naively" In: *Proceedings of the First International Conference on Applications of AI to Engineering.*

Evans, TG. "A program for the solution of a class of geometric analogy intelligence test questions" In: M Minsky, editor, *Semantic Information Processing* MIT Press.

Faltings, B, 1989. "Qualitative kinematics in mechanisms" *Artificial Intelligence* (accepted).

Fikes, RE, 1969. *REF–ARF: A System For Solving Problems Related as Procedures* Technical Report, Technical Note 14, Stanford Research Institute.

Fikes, RE and Nilsson, NJ, 1971. "STRIPS: A new approach to the application of theorem proving to problem solving" *Artificial Intelligence* **2** 189–208.

Fox, MS, 1983. *Constraint Directed Search: A case of Job Shop Scheduling* PhD thesis, Carnegie-Mellon University.

Gentner, D, 1983. "Structure mapping: A theoretical framework for analogy" *Cognitive Science* **7**.

Gentner, D and Landers, R, 1985. "Analogical reminding: A good match is hard to find" In: *Proceedings of the International Conference on Systems, Man and Cybernetics* pp 607–613, Tucson, AZ.

Gentner, D and Toupin, C, 1986. "Systematicity and surface similarity in the development of analogy" *Cognitive Science* **10** 277–300.

Gero, J, 1987. Seminar at Carnegie Mellon University.

Gero, J, 1990. "Design prototypes: A knowledge representation schema for design" *AI Magazine* **11**(4) 26–36.

Goel, AK, 1989. *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving* PhD thesis, Ohio State University

Goel, AK, Kolodner, JL, Pearce, M, Billington, R and Zimring, C, 1991. "Towards a case-based tool for aiding conceptual design problem solving" In: *Proceedings of the 1991 DARPA workshop on Case Based Reasoning*, pp 109–120.

Gordon, WJ, 1961. *Synectics: The development of Creative Capacity* Harper & Row.

Gross, MD, 1986, *Design as Exploring Constraints* PhD thesis, MIT.
Guilford, JP, 1959. "Creativity" *American Psychologist* (5) 444–454.
Hammond, KJ, 1986. "CHEF: A model of case-based planning" In: *Proceedings of AAAI-86* pp 267–271, Philadelphia, PA.
Holland, JH, 1975. *Adaptation in natural and artificial systems* University of Michigan Press.
Holyoak, KJ and Koh, K, 1987. "Surface and structural similarity in analogical transfer" *Memory and Cognition* **15** 332–340.
Huhns, MH, and Acosta, RD, 1987. *Argo: An Analogical Reasoning System for Solving Design Problems*, Technical Report AI/CAD-092-87, Microelectronic and Computer Technology Corporation.
Jevons, WS, 1892. *The Principles of Science* McMillan.
Joskowicz, L and Addanki, S, 1988. "From kinematics to shape: An approach to innovative design", In: *Proceedings of the Seventh National Conference on Artificial Intelligence* pp. 347–352.
Kass, AM and Leake DB, 1988. "Case-based reasoning applied to constructing explanations" In *Proceedings of the DARPA Workshop on Case-based Reasoning* pp 190–208.
Kedar-Cabelli, ST, 1985a. *Analogy—From a unified perspective* Technical Report ML-TR-3, Department of Computer Science, Rutgers University.
Kedar-Cabelli, ST, 1985b. "Purpose-directed analogy" In: *Proceedings of the Cognitive Science Society Conference.*
Kedar-Cabelli, ST, 1985c. *Analogy—From a unified perspective* Technical Report ML-TR-3, Rutgers University.
Kling, RE, 1971. "A paradigm for reasoning by analogy" *Artificial Intelligence* **2**(2).
Koestler, A, 1984. *The act of creation* McMillan.
Kolodner, JL, 1980. *Retrieval and organizational strategies in conceptual memory: A computer model* PhD thesis, Yale University.
Kolodner, JL, 1981. "Organization and retrieval in a conceptual memory for events" In: *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.*
Kolodner, JL, 1984. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model* Lawrence Erlbaum Associates.
Kolodner, JL, 1988. "Retrieving events from a case memory: A parallel implementation" In: *Proceedings of the 1988 Case-Based Reasoning Workshop* pp 233–249, Clearwater, FL.
Kota, S, 1990. "A qualitative matrix representation scheme for the *conceptual design of mechanisms" In: Proceedings of the ASME Design Automation Conference (21st Biannual ASME Mechanisms Conference).*
Kuhn, TS, 1970. *The structure of scientific revolutions* University of Chicago Press.
Lozano-Perez, T, 1983. "Spatial planning: A configuration space approach" *IEEE Transactions on Computers* C-32(2).
Maher, ML, 1984. *HI-RISE: An Expert System For The Preliminary Structural Design of High Rise Buildings* PhD thesis, Department of Civil Engineering, Carnegie-Mellon University.
Maher, ML and Zhao, F, 1986. Using experience to plan the synthesis of new designs Technical Report, Engineering Design Research Center, CMU.
Maher, ML and Zhao, F, 1987. "Using experience to plan the synthesis of new designs" In: JS Gero, editor, *Expert Systems in Computer-Aided Design* North-Holland.
Minski, M, 1982. *Learning Meaning* Technical Report, M.I.T.
Mitchell, TM, Keller, RM and Kedar-Cabelli, ST, 1986, "Explanation-based generalization: A unifying view" *Machine Learning* **1**(1).
Mittal, S, Dym, C and Morjaria, M, 1985. "PRIDE: An expert system for the design of paper handling systems" In: C Dym, editor, *Applications of Knowledge-Based Systems to Engineering Analysis and Design* pp 99–116, American Society of Mechanical Engineers.
Mostow, J, 1985. "Toward better models of the design process" *The AI Magazine* Spring.
Mostow, J, 1986. "Why are design derivations hard to replay?" In: TM Mitchell, JG Carbonell and RS Michalski, editors, *Machine Learning—A guide to current research* Kluwer.
Murthy, SS and Addanki, S, 1987. "PROMPT: An innovative design tool" In: *Proceedings of the sixth national conference on artificial intelligence* pp 637–642.
Navin chandra, D, 1987. *Exploring for Innovative Designs by Relaxing Criteria and reasoning from Precedent-Based Knowledge* PhD thesis, MIT.
Navin chandra, D, 1988. "Case-based reasoning in CYCLOPS, a design problem solver" In: J Kolodner, editor, *Proceedings of the DARPA Workshop on Case-based Reasoning* pp 286–301, Morgan Kaufman.
Navin chandra, D, 1991. *Exploration and Innovation in Design: Towards a Computational Model* Springer-Verlag.
Navin chandra, D, Sriram, D and Kedar-Cabelli, ST, 1987. "On the role of analogy in engineering design: An overview" In: D Sriram and B Adey , editor, *AI in Engineering, Proceedings of the 2nd International Conference* Computational Mechanics.

Navin chandra, D, Sycara, K and Narasimhan, S, 1991a. "A transformational approach to case based synthesis" *AI EDAM* **5**(1).

Navin chandra, D, Sycara, KP and Narasimhan, S, 1991b. "Behavioural synthesis in CADET, A case-based design tool" In: *Proceedings of the Seventh Conference on Artificial Intelligence Applications* Miami, FL.

Newell, A and Simon, HA, 1972. *Human Problem Solving*, Prentice-Hall.

Nilsson, NJ, 1980. *Principles of Artificial Intelligence*, Tioga Publishing Co.

Osborn, AF, 1953. *Applied Imagination* Charles Scribner's Sons.

Pahl, G and Beitz, W, 1984. *Engineering Design*, Springer-Verlag.

Prieditis, A, editor, 1988. *Analogica* Pitman.

Rickards, T, 1974. *Problem Solving through Creative Analysis* Wiley.

Riesbeck, CK and Schank, R, 1989. *Inside Case-Based Reasoning* Lawrence Erlbaum Associates.

Rosenman, M and Gero, JS, 1989. "Creativity in design using a prototype approach" In: *Proceedings of the International Conference on Creative Design* pp 207–232.

Rossman, J, 1931. *The Psychology of the Inventor* Inventor's Publishing.

Schank, RC, 1982. *Dynamic Memory: A Theory of reminding and learning in computers and people* Cambridge University Press.

Schank, RC, 1986. *Explanation Patterns: Understanding Mechanically and Creatively* Lawrence Erlbaum Associates.

Schank, R, 1988. *The Creative Attitude: Learning to ask and answer the right questions* Erlbaum.

Schank, RC and Abelson RP, 1977. *Scripts, Plans, Goals and Understanding* Lawrence Erlbaum Associates.

Sriram, D, 1986. *Knowledge-Based Approaches for Structural Design* PhD thesis, Carnegie Mellon University.

Stefik, M, 1980. *Planning with Constraints* PhD thesis, Stanford University.

Steinberg, L, Langrana, N, Mitchell, T, Mostow, J and Tong, C, 1986. *A Domain Independent Model of Knowledge-Based Design* Technical Report AI/VLSI Project Working Paper No. 33, Rutgers University.

Sussman, GJ and Steele, GL, 1980. "CONSTRAINTS—A language for expressing almost hierarchical constraints" *Artificial Intelligence* (14) 1–39.

Sycara, K, 1987. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods* PhD thesis, School of Information and Computer Science Georgia Institute of Technology.

Sycara, K and Navin chandra D, 1989a. "Representing and indexing design cases" In: *Proceedings of the Second International Conference on Industrial and Engineering Applications of AI and Expert Systems* TullahomK, TN.

Sycara, D and Navin chandra D, 1989b. "Integrating case-based reasoning and qualitative reasoning in design" In: J Gero, editor, *AI in Design* Computational Mechanics.

Sycara, K and Navin chandra, D, 1991. "Index transformation techniques for facilitating creative use of multiple cases" In: *Proceedings of the twelth International Joint Conference on Artificial Intelligence.*

Tong, C, 1986a. *Knowledge-Based Circuit Design* PhD thesis, Stanford University.

Tong, C, 1986b. *A framework for organizing and evaluating knowledge-based models of the design process* Technical Report AI/VLSI Project Working Paper No. 21, Rutgers University.

Tversky, A, 1977. "Features of similarity" *Psychology Review* **84**(4), July.

Ullman, DG and Dietterich, TA, 1987. "Mechanical design methodology: Implications on future developments of computer-aided design and knowledge-based systems" *Engineering with Computers* **2** 21–29.

Wallas, G, 1926. *The Art of Thought* Harcourt.

Williams, B, 1989. *Invention from First Principles via Topologies of Interaction* PhD thesis, Massachusetts Institute of Technology.

Williams, B, 1990. "Interaction-based invention: Designing novel devices from first principles" In: *Proceedings of AAAI-90* pp 349–356, Boston, MA.

Winston, PH, 1980. "Learning and reasoning by analogy" *Communications of the ACM* **23**(12), December.

Winston, PH, 1981. *Learning New Principles from Precedents and Exercises: The Details* Technical Report AI Lab Memo 632, MIT AI Lab.

Winston, PH, Binford, TO, Katz, B and Lowry, M, 1983. "Learning physical descriptions from functional definitions, examples and precedents" In: *Proceedings of AAAI-83* August.

Woodbury, RF, 1989. "Design Genes" In: *Proceedings of the International Conference on Creative Design* pp 133–154