The Knowledge Engineering Review, Vol 7:4, 1992, 345–362

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

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 article 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 article is the second part of a two-part paper that presents and discusses a variety of association and exploration methods. This part examines exploration 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 artificial intelligence, evolutionary epistemology, and the automated discovery literature.

1 Design by exploration

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. Innovative designs are not necessarily obtained through some deliberate attempt at producing them, but by generating a wide variety of design alternatives and simply throwing away the bad ones. Consequently, the ability of a system to innovate depends, in part, on how well it can generate diverse alternatives that break away from the norms and governing constraints. Exploration involves relaxing constraints and looking in unlikely places to expose possibilities that would not normally have been considered, possibilities that may serendipitously lead to innovative solutions.

Typically, problem solving techniques that are used in design are set up to narrowly focus their search for solutions and to reduce the effort spent in examining alternatives. For example, in numerical optimization or heuristic search, algorithms are set up to exploit the structure of the given constraints to narrow the space being examined. Many years of research have gone into finding efficient algorithms that are able to reach the goal with minimal digression from the optimal path. However, in an innovative design process, one tries to do quite the opposite. The aim is to explore beyond the bounds set by the constraints, to look in unlikely places, and to experiment with the unknown. Exploration techniques are not a replacement for standard problem solving and optimization approaches, but are aimed at trying to keep the design process a little more flexible. A question that might come up is: "If a given algorithm is known to find the true optimal to a given

problem, then why does one need exploration?" Exploration is useful because, in a real-world design situation the criteria (constraints and objectives) by which a successful design is judged tend to change over the course of design. New opportunities are recognized and older constraints are modified. Often new criteria may enter the design equation thereby changing the original focus and goals.

A designer is forced to innovate whenever he is faced with a problem that cannot be solved in some previously known way. The designer can either explore within his current design culture or could use ideas drawn from other domains. The process of exploration can be either syntactic or context driven; further, it can be either goal or data driven. Whenever a designer reaches a dead end in a design process he can start exploring new alternatives by making syntactic changes (mutations) to the artifact he is designing. He may make such changes in the hope of finding a novel combination that satisfies the given design goals. Such an approach can be extremely inefficient. A designer can also explore new alternatives in ways other than making random syntactic changes. When faced with a design problem, he can perform an analysis of the problem and then make modifications to the artifact in direct response to the results of the analysis. In this case, the designer is reasoning about the nature and direction of exploration, and hence, is using a *context-driven* approach.

Exploration can also be done in a data driven fashion. In this case, the designer has no explicit goals, but is trying to find some regularity or some hidden principle in the alternatives generated by exploration. This process is called *discovery*. Discovery does not necessarily rely on explicit goals for testing alternatives, but uses heuristics to recognize interesting patterns and relationships.

Some desirable characteristics of an exploration technique are:

- 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.
- 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.

In building computer programs that explore, we need:

1 To know when, and in which direction to explore.
2 Methods or operators with which the exploration can be conducted.
3 To resolve the conflict between trying to keep the search space small (for efficiency) and the need to expand the search space (to explore).
4 Ways for using knowledge to recognize promising alternatives during an exploration phase—to know *when* something is interesting.
5 Operators and techniques that extend the search space (the A-space).

In this paper, we examine the four broad classes of exploration methods: (1) Searching for novel combinations; (2) Mutation and Relaxation; (3) Discovery; (4) Analytical Exploration and Invention. The first set of techniques use goal-directed methods to guide the search for solutions. The second set of approaches use relaxation and mutation. They are different from the first set of approaches in that they try to generate alternatives that are closely related to existing solutions. The idea is to mutate existing solutions or to slightly relax a governing constraint to modify the solution. The third set of approaches are data-driven. These systems search the design space, not with the purpose of finding a solution, but to discover a regularity, a pattern, or some concordance in the data points encountered. The fourth set of approaches are goal-directed. They use analytical knowledge and equations about the design to produce new designs. Analytical techniques are very reliable, but may not be able to produce novel solutions like the other exploration methods.

2 First set of approaches: searching for novel combinations

Consider a system that can search the combinatoric space (A-space) of a given set of objects using rules of combination. If such a system can be given a set of criteria to test combinations with, it can be set off on a search for solutions. Given a complex set of criteria and sufficient computer time, such programs could come up with combinations never thought of before. By following a simple generate and test paradigm it is possible to come up with novel combinations (as long as the A-space is static).

One of the first programs in this category was DENDRAL (Buchanan & Feigenbaum, 1978). The program had the task of determining the physical structure of organic molecules. The program takes as input: a mass spectrograph and the chemical composition of the molecule in question. Using rules about how molecules fragment, the program first generates a set of constraints on the possible structure. It then enters a generate and test process where it generates possible structures and tests them. The testing is done by comparing a theoretically derived spectrograph with the actual spectrograph. The test is successful if a good fit is found between the two graphs. A major drawback of this paradigm is that it is very slow. This problem has been alleviated by using special heuristics to search the A-space.

A program that uses heuristics to explore structural models is a discovery system called DALTON (Langley et al., 1987). The goal of the program is to devise a model of chemical reactions that specifies the number of molecules and atoms involved. Given a list of reactions and components of substances, the program uses heuristic operators to search, in a depth-first fashion, through an A-space of possible models. The search proceeds by making assumptions about the number of atoms in a molecule and testing these hypotheses against laws of conservation, combining volumes, etc. Here is a trace of how DALTON figures out the equation for the formation of water by combining hydrogen and oxygen (H and O denote molecules, h and o denote atoms):

1 Starting with the reaction (Hydrogen + Oxygen → Water)
2 Assume one molecule each ((H) (O) → (W))
3 Guess internal structure (single particles are denoted by h and o):
((h) (o) → (W))
4 By the conservation heuristic:
((h) (o) → (h o))—this result was also arrived at by the 18th century chemist Dalton.
5 Using a heuristic about combining volumes, the following is inferred:
((H) (H) (O) → (W) (W)) which leads to
((h) (h) (o) → (W) (W))—the above (#4) reaction is revised using data about combining volumes.
6 DALTON assumes the oxygen molecule is made of two particles as the last equation above does not pass the conservation heuristic:
((h) (h) (o o) → (h o) (h o))
7 The above reaction is found to be wrong by a heuristic that knows how hydrogen reacts in some other reaction. At this point, the system backs up, and tries a different branch.
8 The search proceeds until the following structure is reached:
((H) (H) (O) → (W) (W)), which gives
((h h) (h h) (o o) → (h h o) (h h o)), finally
$2H_2 + O_2 \rightarrow 2H_2O$

Using a very similar technique, a bridge design system, CIDS has been developed (Cheyayeb, 1987). The program works with objects such as beams, arches and cables. Given a set of functional requirements, the program searches its database for structural elements that can satisfy the given conditions. If no single element is found, combinations of elements are considered. The program follows a beam search technique. It uses the number of unsatisfied constraints as a measure of goodness. For example, given the problem of spanning a wide gorge which is too long for a single

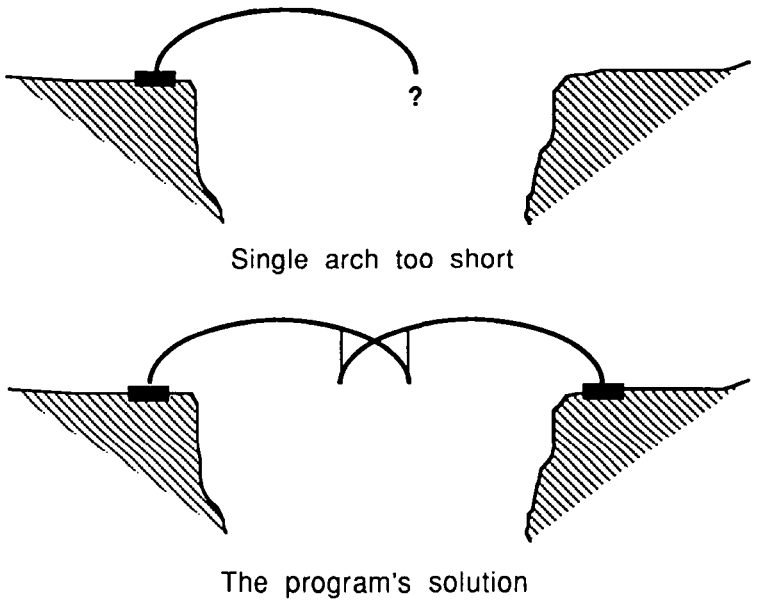

Figure 1 CIDS solves a short arch problem

arch, the program comes up with a solution in the following manner: The program first places an arch and then realizes it needs a support for the arch (Figure 1). There are several ways of supporting an arch; it finds a cable and suspends the arch's free end from the cable. Next, it needs a place to hang the cable from, again, there are several choices: cable, arch or beam. By going through a process of incremental generation, the program solves the problem as shown in Figure 1. The various combinations that the program comes up with are novel.

The use of heuristics for the generation of solutions is generally better than blind combinatoric search methods. Concern, however, has been aired about the completeness of heuristic generators. This problem has been addressed in a layout design system called LOOS (Flemming et al., 1989). The system is based on rules for laying out loosely packed rectangles in a two dimensional space. Though the problem formulation is general, the program has been applied primarily to architectural layout tasks. The rule-based layout *generator* is domain-independent, while the *tester* contains domain knowledge that is used to evaluate the results of generation. The LOOS system operates as a tool that provides support for generation, evaluation, and patching of design alternatives (Coyne & Subrahmanian, 1991). The success of the approach is a direct and literal application of the idea that exploration aids innovation. The program systematically applies rules to generate alternative layouts, while the designer is allowed to experiment with alternative problem spaces and decompositions.

Innovative design systems can be built using fairly simple heuristics as shown above. The apparent inventiveness of such systems is predicated on the fact that the search space is so large that a computer can stumble upon combinations never before thought of by humans. This phenomenon holds true, not only for structural combination systems but for *all* systems that explore while searching for solutions.

3 Second set of approaches: use of mutations in exploration

Exploration by mutation is the most powerful and inefficient approach for computer aided design. Mutation may be used in a *generate and test* framework. Designs can be generated by either mutating existing designs or by mutating random configurations of a technology. The designs can then be tested using criteria that check for feasibility, optimality and functionality. Designs that fail the test are pruned, while others are explored further. By continuously generating a large number and variety of alternatives it is hoped that innovative solutions will eventually be produced. The

idea of using mutation as a heuristic for finding interesting alternatives comes directly from the theory of Natural Selection. The idea is that "as many more designs are generated (by mutation) than can possibly survive pruning; and as, consequently, there is a frequently recurring test of optimality, it follows that any design, if it varies however slightly in any manner profitable to itself or its variations, under the complex and sometimes varying set of constraints and objectives, will (together with its variations) have a better chance of surviving the pruning test, and thus be naturally selected". This quotation is a modified form of the theory of Evolution by Natural Selection as put forward by Charles Darwin in *On the Origin of Species* (Darwin, 1859). The original version from the 1859 text is: "As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it varies however slightly in any manner profitable to itself, under the complicated and sometimes varying condition of life, will have a better chance of surviving, and thus be naturally selected." The idea of drawing an analogy between biological evolution and a human intellectual activity (such as innovative design or scientific discovery) is not new. Even before Darwin's *Origin*, Spencer (1857) drew an analogy between the organic process and the progress of scientific knowledge. Toulmin, a contemporary philosopher, suggests that ideas or concepts which are introduced into the pool of science (by scientists) have to battle for existence and proof of worth (Toulmin, 1967). Popper (1965) proposed a similar view of how new scientific ideas are generated.

Darwin claims that organisms appear to adapt to their environment through natural selection only. This is an interesting and important insight for us. According to the theory of natural selection, organisms are constantly mutating to produce variants. Those variants that are better suited to their environment, survive and pass on their characteristics to their progeny. Interestingly, all such variations are random. It is hard to believe that a random process can indeed generate interesting adaptations. It is hard to believe that a turtle's head shell is only a product of random mutation; but that's only because one is forgetting that the creation of a hard shell is one of possibly millions of other mutations that, being less *fit*, perished. It is for this reason that the approach, though powerful, is inefficient.

There are two types of mutation: phenotypic and non-phenotypic. The first approach is limited to making mutations to the physical structure of design alternatives. This approach tends to produce a very large proportion of nonsensical designs. This problem is addressed in non-phenotypic mutation. The idea is not to mutate the design directly, but to mutate some meta-description of the design. The trick lies in developing a meta-description language that does not get distorted by syntactic mutation. It is also important that there is two-way mapping between all possible designs and the meta language.

3.1 Phenotypic mutation

The simplest mutation method is to randomly re-arrange the features of a given design or the technologies of a given culture. For example, a kitchen layout design system can mutate designs by randomly re-arranging the oven, cabinets, doors, dishwasher, etc. One may wish to make sure that only physically possible mutations are generated, i.e. the dishwasher is not inside the oven. A problem with this approach is its utter inefficiency. Randomly moving physical parts in a car, for example, is unlikely to produce a new meaningful solution. It's like hoping that a monkey jumping on a typewriter will write poetry.

Phenotypic mutation can work in domains where the physical representation being used can yield meaningful results even when mutated. Instead of using completely random mutations, one can get better results by using a pre-defined, domain-specific set of operators that syntactically modify the design. A good example of this approach is the AM system (Lenat, 1976). In its search for mathematical concepts, AM essentially operated as an automatic programming system (Lenat, 1984). It searched the A-space by making mutations to mathematical concepts. It did this by making syntactic mutations to the LISP code that represented the concepts. The program used a set

of rich heuristics for deciding when a concept was interesting or what mutator should be applied to it. For example, the COND clause statement in some piece of LISP code can be generalized by replacing an AND with an OR, which, in essence, is equivalent to relaxing the strictness of the function from a conjunction to a disjunction.

AM explored its A-space by making syntactic changes to LISP code, and by using heuristics to recognize if the mutated concept was interesting or not. The program was able to discover concepts such as prime numbers, addition, subtraction, deMorgan's laws, etc. AM's success led to the belief, at least initially, that pure syntactic mutation of concepts can yield new meaningful concepts. This rather profound idea, unfortunately, cannot be easily applied to domains other than mathematics. The reason AM worked was because it used LISP as an implementation language. In Lenat's own words: "the density of worthwhile math concepts are represented in LISP that is the crucial factor . . . It was only because of the intimate relationship between LISP and Mathematics that the mutation operators turned out to yield . . . useful new math concepts . . . (however, when applied to something else, such as:) generating new heuristics, the technique almost always produced useless new heuristic rules" (Lenat, 1984). When AM mutated small portions of LISP code that made up mathematical concepts, it produced new concepts. It was because LISP and mathematics are so related that mutations on LISP code maintained the semantic equivalence between the code and the mathematical concept it represented. However, when mutations were made on large sections of LISP code that represented heuristics, it became harder to guarantee semantic preservation. This is because the syntax of representations are not normally semantically related to what is being represented.

There is no existing representation of physical designs that will do what LISP did for AM. We cannot represent a kitchen in such a way, that phenotypic (syntactic) mutations to the representation would turn out meaningful new kitchen layouts.

3.1.1 Phenotypic bisociation

Combining two good designs to produce a new one can sometimes produce something better (Koestler, 1964). In the case of natural evolution, parents pass on their characteristics to their children. A child can inherit the best or worst of both parents or some combination thereof. This helps produce variety without wasted search. In a design system, all the alternate paths being considered during search can be viewed as the population of a species. By choosing and combining two good, but different designs can produce an interesting offspring. The simplest technique is *cross-over*. In Genetic Algorithms (Grefenstette, 1987) the bisociation of two strings of bits is achieved by chopping them in half and exchanging opposite halves to produce two new strings. The problem with applying this idea to design is that we cannot find a representation of designs that allow such cross-over. For example, one cannot combine two kitchen designs by cutting one in half and combining it with the half of another. Bisociation should combine the ideas underlying the designs not just their physical representations. We need better non-phenotypic methods.

3.2 Non-phenotypic mutation

What if the monkey was jumping on a typewriter where the keys were ideas and the concatenation of ideas were automatically converted into sentences? Then would the monkey's writing make sense? A design exploration system that works with the ideas underlying designs rather than a representation of the physical design has a much better chance of generating interesting solutions. A means of representing and manipulating design ideas is needed.

I have found design constraints to be a viable means for representing the ideas underlying a design. Constraints are expressions (algebraic or logical) that capture relations among features of a design. For example, the proximity of two objects (kitchen sink and stove) may be expressed as an algebraic inequality that limits the distance between the objects. Constraints are typically used to represent specifications or requirements, but can also be viewed as a means for capturing design ideas. In the former case, constraints are used as criteria to evaluate designs. For a given set of

constraints, there are usually many design alternatives that will satisfy the constraints. The alternatives that one would actually generate during design depend on the algorithms/heuristics being used. To view constraints as ideas, the above process has to be reversed, at least partially.

Assume we want to produce variations (mutations) of a given design. The ideas underlying the design have to be extracted and represented as constraints. Consider, for example, a kitchen layout problem. Assume we start out with a design that has to be mutated. We start by extracting the essential constraints that are embodied in the design. For instance the designer might have placed all essential services in an island in the middle of the kitchen. This idea can be expressed as an algebraic constraint that relates the position of the island to the positions of all the other objects in the kitchen. The set of constraints extracted from the design can then be used to generate new designs using some standard problem solving algorithm. The new designs will be physically different form the original design but will satisfy the same constraints. In this framework, the constraint set can be mutated (by adding, deleting of modifying) before new design alternatives are generated. These new designs will be mutations of the original design. This approach alleviates the problem of having a design representation which will remain semantically intact when it is syntactically mutated. Instead of mutating the design directly we mutate the constraints that are used to generate the design.

The constraints are, taking the biological analogy further, a genotypic representation of the design. If a design is physically mutated, the chances of generating a meaningful new design are slim. For example, randomly adding or erasing lines on a sketch of a kitchen layout will not produce viable new designs. On the other hand, if one mutates the constraints that are used to generate a new design, the resulting solution will at least be a complete design. Hence the analogy between constraints and genes.

Two types of constraint mutation have been implemented and tested in a design system called CYCLOPS (Navin chandra, 1987). The system relaxes constraints and adds new constraints during a design session. In this way, the criteria by which the design is evaluated keep changing during the design process. In CYCLOPS both constraints and objectives are relaxed during problem solving. Relaxing constraints gives rise to new alternatives, some of which may fortuitously lead to improved designs. For example, a landscape designer who relaxes the constraint that "all homes be on slopes less than 8%" to some higher value (say 10%), makes available to him, plots of land that are between 8% and 10% slope. It is possible that some of these new alternatives may provide opportunities such as better soil conditions or a better view. Relaxation overcomes the artificial precision built into a criterion. When we say that the slope should be less than 8%, it does not mean that lots with slightly higher slopes of say 9% or 10% be completely avoided.

In CYCLOPS, the objectives are also relaxed. This is shown in Figure 2. The left half of the figure shows a tradeoff situation between two Objectives O1 and O2. The dots indicate design alternatives. The dominated alternatives are shown (shaded) and the dominating solutions are shown as black dots. The curve passing through the dominating solutions is the Pareto surface. In normal search systems, only the non-dominated designs are considered as possible solutions; all the other inferior designs are eliminated. During exploration, however, we would like to consider

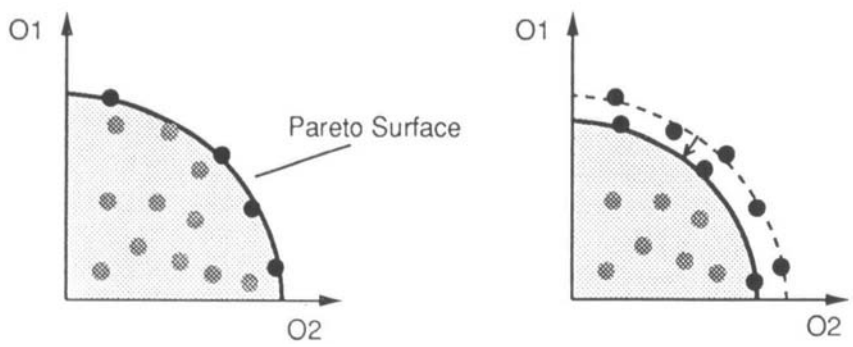

Figure 2 Pareto slip

some of the dominated solutions too. The situation depicted on the left half of the figure can be relaxed by deliberately pushing the Pareto surface towards the origin. In effect, all solutions which lay just below the Pareto surface become available for consideration (right half of the figure). It is in this way that new alternatives can be brought into consideration through objective relaxation. An important property of this technique is that it prefers to surface only those designs that are very close to being non-dominated.

CYCLOPS also changes the constraint set by introducing new constraints during problem solving. It does this by using past experiences (cases). If a new design resembles a previously encountered design case, then the system retrieves the case and uses it. Any prior evaluation stored in the case, favourable or unfavourable, is applied to the new design. If a design alternative has some interesting characteristics, a corresponding new criterion is added to the current problem. The emergence of criteria during the design process can sometimes change the focus of problem solving, and could lead to a solution completely different from what a standard synthesis technique would produce.

Another example of using non-phenotypic mutation has been reported by Lenat (1983). The EURISKO program (unlike AM) made modifications, not to the LISP code that represented concepts, but to the heuristics being used in problem solving. EURISKO's representation of heuristics is frame-based, which provides a functional decomposition of the heuristic. Each function could be independently mutated and the semantic preservation property that AM satisfied fortuitously is re-created by keeping mutations local to functions. This raises some important questions for innovative design. To make random mutation meaningful there should be a strong semantic equivalence between the artifact representation language and the artifact. It is very difficult to achieve this property for physical artifacts.

3.2.1 Achieving semantic equivalence

The highest level of semantic equivalence is in the domain itself. For example, in the blocks world domain, mutation works best if one moves actual blocks around to change their arrangement. Each mutation gives you a new true state of the world. In building computer models that manipulate artifacts we are dealing with two aspects: representation and reasoning. At one end of the spectrum we could have a representation that is semantically so close to the actual domain that any mutation to the representation is not meaningless. At the other extreme, one could use a very abstract representation but make use of the representation in such a way that all actions taken by the control (reasoning) program maintains semantic equivalence. For example, a computer simulation that displays bouncing balls on a screen creates a physical scene using a mathematical representation. The program that sits between the representation and the display device is doing the job of bridging an equivalence between a purely mathematical representation and a simulation we can comprehend. Let us now assume we want to mutate the bouncing balls. A possible mutation might be to cut the balls in half. It is not possible to cut the balls in half by throwing away half the computer code that generates them, but a new function will have to be written to make the change. In fact, this new function will be much harder to write than the original one.

A system that portrays a mix of using a good artifact representation and domain specific mutation operators is a design invention system called EDISON (Dyer et al., 1986). The system is a joining of qualitative reasoning and innovative design strategies. Mechanical devices are defined in terms of parts, spatial relations, connectivity, functionality and processes. The representation is rich enough to allow simulation of the physical devices. The EDISON team has identified the strategies of invention of generalization, analogy and mutation, all of which rely on memory organization, indexing and retrieval. The mutation heuristics that EDISON uses are domain specific pieces of code that mutate in semantically correct ways. For example, Figure 3 shows how EDISON starts with a standard door, mutates it by cutting it in half, finds that the second half has no support, adds supporting hinges, and finally invents the dual bar-room door. Along a different branch of the search tree, EDISON moves the hinges of the normal door from the side to the top of the door and, in effect, invents a trap-door.

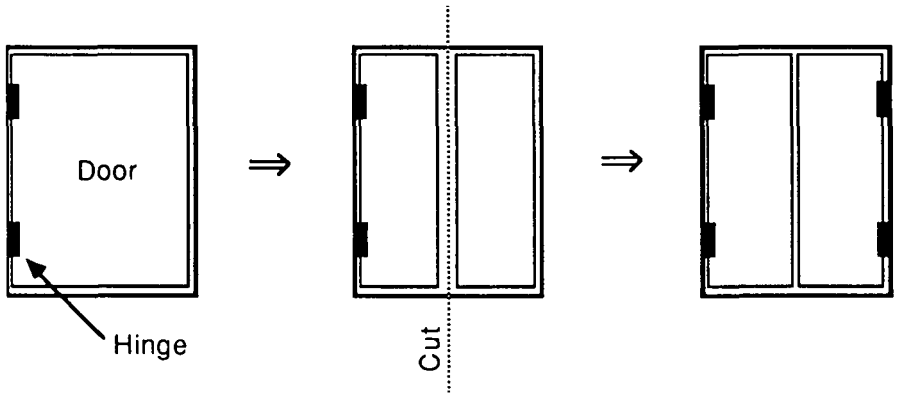

Figure 3 EDISON mutates a door

To build systems that change physical artifacts in meaningful ways we need to develop a balance of the two aspects described above: the representation and the interpretation mechanism that works with the representation.

4 Third set of approaches: discovery systems

Discovery systems are programs that search the A-space not with the purpose of finding a solution, but to discover some regularity, a pattern, or a concordance in the data provided. Discovery programs generally use rather simple statistical or numerical analysis techniques to discover new theories, relationships, etc. These systems rely on the computer's ability to search tirelessly, and are not based on any cognitive model of human thought processes.

Even though discovery systems have a very different purpose than design innovation systems, there are several techniques that we can borrow from the discovery literature. Techniques include the use of exploration heuristics, operators for exploration, and testing techniques that recognize a new discovery. The last issue is of great importance. How does a discovery system know it has made a discovery? There are essentially three methods:

1 The new alternative is better than previous alternatives (e.g., lower weight or higher efficiency).
2 The system can have a global goal to find an interesting pattern, e.g., finding a regular relation among data points in an amorphous dataset.
3 The new alternative may remind the system of an episode from a database of past experiences. If the database has knowledge from a wide range of sources, then it is possible that a reminding will help recognize an interesting solution. A good example is the case of the cardiologist who, in the process of testing new medicines for hypertension, found that one of the drugs, Minoxidil, had an annoying side effect. It caused patients to grow excessive hair. If the doctor was only interested in cardiology he would not have realized that he had accidentally discovered a wonder drug for baldness.

The first technique (above) is commonly used in routine design systems. Pre-defined objectives can easily be used to compare and select design alternatives. We examine the other two approaches in detail below.

4.1 Data driven discovery

Discovery by Bacon Bacon is a quantitative discovery system that finds mathematical relations between given data sets. For example, Bacon can discover Kepler's third law by working with a

given dataset of distances (d) of the different planets to the sun with their corresponding time periods (p). The goal of the program is to find a relation f such that $f(p,d)$ = constant. Let us assume that dataset has both p and d increasing monotonically (shown below, adapted from Walker, 1987):

Planet	p	d
Mercury	1	1
Venus	8	4
Earth	27	9

As the values increase monotonically, a certain heuristic suggests the term (d/p) by calculated. On seeing that d and (d/p) vary inversely, another heuristic suggests multiplying the two to give (d^2/p), by following such heuristics the program finally finds (d^3/p^2) to be constant. Which is Kepler's third law!

Patterns in medical data Another data driven discovery system is a medical data analyser called RX (Blum, 1982). RX works from a database of patient information, and tries to find medical relations between the attributes. It uses statistical techniques and medical domain knowledge in the process. The program uses cross-correlation to hypothesize relations. It then checks the hypothesis for statistical significance. RX, unlike Bacon, has some medical domain knowledge that it uses to build new relations. Medical relations discovered by RX are added into the program's knowledge base and can be used in future iterations.

Some discovery systems use knowledge more extensively, and are not purely data-driven. For example AM/EURISKO (Lenat, 1976) and DAYDREAMER (Muller, 1987) are examples of systems that innovate by both syntactic searching and intelligent use of knowledge and heuristics.

4.2 Memory driven discovery

In order to innovate, it is important not only to have the ability to explore new alternatives, but also to have the ability to recognize innovative alternatives when they are generated. This section discusses how reminding can be used as a way of recognizing and measuring interestingness. Alternatives generated through criteria relaxation or mutation can sometimes cause the designer to be reminded of some previous episode. It is through such a reminding that the designer may recognize an opportunity, serendipitously solve some previous goal, or emerge new design criteria. The techniques mentioned here are all based on making associations between a current design and some prior episode, experience, or design. These kinds of associations play an important role in exploration; they help the system tell when an interesting solution has been found. Memory based approaches for matching and similarity recognition were covered in Part I of this paper. We now show how those techniques could be used for serendipity recognition during exploration.

4.2.1 Serendipity recognition

The following is a list of ways in which systems can be set up to recognize unexpected opportunities.

1. A new state leads to the serendipitous solution of given problem. The DAYDREAMER program displays creative behaviour through the use of a mutation mechanism and a serendipity recognition mechanism. The program takes descriptions of events in the world as input and produces a sequence of events it will perform in an imaginary world. The output represents the program's "stream of thought"—a daydream.

The serendipity mechanism enables the accidental achievement of goals or subgoals. The mechanism responds to states, actions, physical objects, and retrieved episodes. Consider the following scenario of DAYDREAMER operating in the domain of social relations: DAYDREAMER is female and is a great fan of the movie star Harrison Ford. One of the goals of DAYDREAMER is to date Mr Ford. The program is then given an input state: "Harrison Ford is

at the Nuart Theater". In response to this input, the serendipity mechanism recognizes that the episode is relevant to its goal to date Mr Ford. It then starts working through the subgoals of the top level goal. A subgoal of the top level goal is meeting Mr. Ford; a subgoal of meeting a person is to have a conversation with him; a subgoal of having a conversation with somebody is to be in close proximity to that person; a subgoal of being in close proximity to anything is being at the same location as that thing; a subgoal of being at the same location is knowing where the thing currently is; and as Mr Ford is at the Nuart theater, DAYDREAMER decides to go there!

DAYDREAMER plans its dreams with the use of rules that are derived from other episodes. The program uses a pre-compiled network of these rules. The network is essentially a graph where any two rules are connected if the goal of one rule matches the subgoal of another. The program starts with the new state pattern and the goal pattern. It then identifies two rules, one which produces the goal state and another which corresponds to the input state. It then finds a path in the rule graph between the two rules. This kind of search is called *intersection search* (Quillian 1968) and is, in essence, a proof mechanism. To complete the serendipity, DAYDREAMER verifies the path by checking that each subgoal unifies with the antecedent goal of the rule that it is connected to. All unbound variables (if any) are collected and their values can be retrieved from the associated episodes. In this way, new objects or situations from other episodes get used in the dreams the program generates.

The serendipity mechanism in DAYDREAMER provides us with a method for finding a plan for solving a design problem by using a retrieved precedent/episode. The process of finding the connection will lead to the use of several other episodes. One problem with this approach is the need for a good verification technique. Sometimes, the connection between a problem and its solution might follow a path that is bizarre and impractical. For example, a design system that is trying to place a house on a swamp might decide to suspend the house with balloons. An innovative idea, but impractical. One way to check for impracticality is to search the episodic database to find any connection between the generated plan and an unfavourable condition. For example, after generating the balloon solution, the program starts searching the rule graph till it finds a path to a new problem. For example, it might find that balloons are unstable in wind, and that the site is windy and that instable houses are not acceptable. In this way we make the "closed-world" assumption that if none of the rules in the graph can find a problem with the generated design, then it is acceptable.

2. I will know what I want, when I see it! This is equivalent to searching for alternatives hoping a positive reminding occurs. For example, you walk into a department store with the idea of buying a birthday gift for a friend. You are not sure what you want, but you walk around the store hoping to find something suitable.

3. A new episode solves a previous problem. It's a lazy Sunday morning, you pick up the newspaper and start scanning it not looking for any article in particular. You suddenly find something interesting and read the full column. The reason this happens is that the article answers some dormant question or problem or curiosity. While working on one design problem, it is possible that one might find a solution to some other problem. To support such behaviour, we need a mechanism that matches the current input episode to some precedent in memory, where the precedent is some previously unanswered question or incomplete design. This approach is found in systems that automatically scan newspaper articles or electronic mail messages.

4. Emergent criteria. The CYCLOPS design system introduces new criteria as it designs. We have seen this before (section 3.2). We now examine an example from the CYCLOPS' domain: consider an architect who is given the task of locating townhouses on a site. Assume that the site is located near the edge of a cliff, and overlooks a valley. The owner's specifications require that there be a fixed number of units, each with a view of the valley, and all within the budget. These specifications are incomplete. The architect, drawing from his past experiences, might decide to consider additional criteria about noise, access, privacy and safety. Taking these specifications, the

designer might decide to place all the houses along the cliff's edge to give each unit a good view of the valley. While working on this layout, the designer might recognize the opportunity for a mini golf-course in the middle of the site. Seeing this, he might decide to change the specifications and add some more houses to the originally specified number and have them face the golf course instead of the valley. In this way, the designer may arrive at, what *he* believes to be a good solution; even if it requires altering the specifications. "The shape of the design space appears to be determined as much by the search process itself as by external factors . . . The definition of design spaces is therefore dynamic, and the design process involves not only search within such spaces but the acquisition and modification of the knowledge that defines those spaces" (Coyne et al., 1989). This scenario, shows how a designer can emerge new criteria as he goes through the design process. CYCLOPS emulates this behaviour (Navin chandra, 1991).

4.2.2 Discussion on memory directed discovery

We have talked about design systems discovering new ideas/artifacts and finding innovative solutions serendipitously. A question that one may ask about design systems that have this kind of behaviour is: "If the program already has the episode in memory, then how is it's use of the episode an accident? If the program has access to it, then why can't it just use it? Why does the system have to *discover* the episode?" There are two responses to this question:

- *Database size.* Often, the episodic memory is so large that testing all episodes against the current problem is impossible. The main task of controlling programs is to generate the right index to retrieve precedents that may be useful. Sometimes a problem just cannot be solved by episodes that first come to mind. One might solve such problems after thinking about it for some time, or just taking a walk in the park. "Consider an inventor who is blocked in the task of creating a new kind of door, one that allows people through either side simultaneously. The inventor decides to 'quit thinking' about the problem and take a walk. While walking along a river, the inventor sees a water wheel. Suddenly a solution comes to mind: the revolving door . . . The water wheel could not be accessed directly . . ." (Dyer et al., 1986). If the designer had tried to access memory for all relevant episodes, he probably would have never found a solution.
- *Learning to use new criteria.* Before starting a design process a designer might list down a set of criteria. These criteria represent what the designer thinks is important prior to the design process. During the process of design new criteria emerge through a reminding process. One might recognize problems in the design which had gone un-noticed earlier. After a set of criteria are emerged and used, they become part of the designer's experience.

The next time the designer comes around to a similar problem his list of prior criteria will be longer and he will appear to have more expertise. The prior criteria for a design problem is dictated by the design culture one is operating in. As one gains experience in designing and one solves problems by analogy to episodes outside the current design culture, the new episodes become part of the design culture. For example, the design of a structural part by analogy to the structure of honeycombs was a very innovative idea when it was first proposed. Today, however, the honeycomb structure is a part of the mechanical engineer's design culture, and is mentioned in introductory engineering design text books. Let us try a thought experiment to show how criteria emergence is helpful. Imagine yourself to be a designer who has the task of designing a hot/cold water tap. Think up some criteria that you might impose on the design. Write them down on a piece of paper before you continue. . . . Pause . . . I am sure you came up with a few good criteria, here is an excerpt from the list of an expert designer (Pahl & Beitz, 1984):

- to fit household basins, convertible for wall fitting
- light operation (for children)
- self-flushing (no deposits)

- maximum throughput 20 L/min at 20 bar pressure
- trademark prominently displayed
- handle/knob not heat over 35°C, even under prolonged use
- no sharp edges
- easy maintenance, can use standard spare parts
- and so on . . .

The experts' list is longer and has some concerns that you probably missed. Interestingly, however, many of the expert's criteria seem to be fairly obvious: surely you want the tap to be operable by children; surely no sharp edges; surely it should be maintainable . . . These requirements are *not* alien to you, but you did not invoke them because they just were not indexed right. It is for this reason that a system that draws new criteria from a database of prior design is said to "discover" (or emerge) criteria.

5 Fourth set of approaches: invention by analysis

Analysis is the fourth and final class of exploration techniques we will discuss. The philosophy underlying these techniques is to use knowledge about physical laws and principles to analytically derive suitable modifications for a design. Modifications, unlike mutations, are generated specifically for the problem at hand, and hence, are very reliable.

In a neo-Darwinian framework, analysis represents the most advanced form of evolution; where a system evolves not by natural selection, but by assessing its own shortcomings and by making modifications to correct the problem. Analytic approaches, however, are not as general as classical mutation, and cannot be expected to make large creative leaps. Designs, before and after modification, often tend to retain the same basic form.

5.1 Modification operators

PROMPT is a design system that debugs by performing a qualitative analysis of design problems (Murthy & Addanki, 1987). The program performs an analysis on the equations that describe the behaviour of the artifact being designed. After the qualitative analysis is completed, modification operators are used to correct the design. One of the design examples that PROMPT has been tested on is beam design. Consider the following scenario: the program is given the task of designing a rod that can bear high torsional loads. The program uses the specifications to retrieve a metal rod from the database. The rod is then tested for torsional stress under the specified loading conditions. The program finds the rod is over-stressed. The program then analyses the equation for torsion $Ts = KR^4/L$ and decides to increase the radius of the rod. This modification, however, violates the weight constraint of the rod. The program, through some more analysis realizes that the torsional stress borne by the rod is higher on the periphery and lower towards the centre of the rod. PROMPT finally decides to redistribute mass from the centre of the rod to the periphery. In this way, PROMPT invents a "hollow torsional member".

PROMPT's ability to analyse problems in such detail is based on a large knowledge base called the "Graph of Models" (Addanki & Davis, 1985). This database uses a graph structure to store physics knowledge about principles such as: bending loads, buckling, torsion, vibration, etc. Using this knowledge, the program can reason about an artifact from first principles and can derive its behaviour from its structure.

PROMPT uses modification operators to eliminate problems. It uses two types of operators: operators that it derives directly from the behavioural equations of the artifact, and operators that are pre-stored in the system. Examples of derived operators are redistribution of mass by removing mass from regions of low stress and adding mass to high stress regions, changing material distribution, and some simple shape changes. Examples of predefined operators are shape

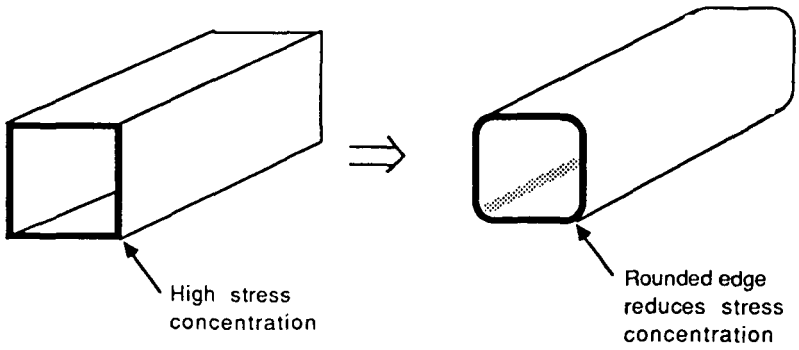

Figure 4 Example of a shape change operator

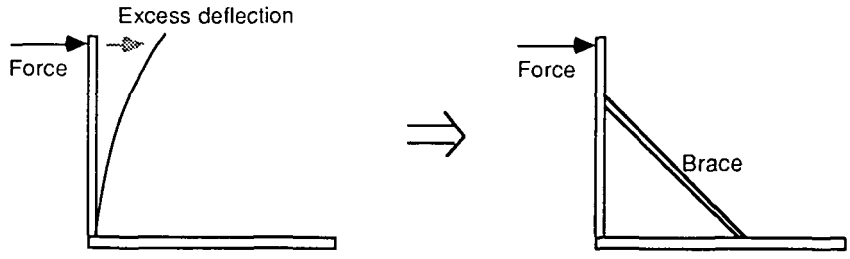

Figure 5 Example of a topology change operator

changing operators and topology changing operators. Here are some examples: Figure 4 shows how a simple shape change in a box beam can change stress concentrations at the edges.

Figure 5 shows how a strut can be used as a structural brace. The PROMPT system can use these modification operators to generate new solutions.

5.2 Non-routine design by dimensional variable expansion

1[st]PRINCE is a non-routine design system that generates innovative structural designs by reasoning from first principle knowledge (Cagan, 1990). Design knowledge is input to 1[st]PRINCE in the form of equations pertaining to the domain of interest as a symbolic optimization problem, which represents a good initial design prototype. The system then uses qualitative techniques of monotonicity analysis as derived from Karush–Kuhn–Tucker (KKT) conditions of optimality to select a critical integral. The design space is then expanded by what is called the "Dimensional Variable Expansion", which essentially divides the integral into a set of smaller ranged integrals. Based on the following definitions 1[st]PRINCE has a set of rules which allow it to manipulate the design space.

Definitions

1 The monotonicity of a continuously differentiable function $f(x)$ with respect to variable x_k is the algebraic sign of $\partial f/\partial x_k$.
2 A constraint $g_i(x) \leq (\geq) 0$ is active at x_o if $g_i = 0$. Otherwise it is inactive.
3 A positive variable x_k is said to be bounded above by a constraint if the variable is maximum when the constraint is active. It is said to be bounded below if the variable is minimum when the constraint is active.

Based on these definitions, Papalambros and Wilde (1988) give three rules of monotonicity analysis which define well-constrained problems optimization problems without overconstrained cases:

Rule 1. If the objective function is monotonic w.r.t. a variable, then there exists at least one active constraint which bounds the variable in the direction opposite of the objective.
Rule 2. If a variable is not contained in the objective function then it must be either bounded from both above and below by active constraints, or not actively bounded at all.
Rule 3. The number of non-redundant active constraints cannot exceed the total number of variables.

In 1[st]PRINCE, the design space is expanded by splitting a critical integral into a summation of smaller ranged integrals as:

$$\int_0^x X = \int_{10}^{x_1} X_1 + \int_{x_1}^{x} X_2$$

This division creates the new dimensional variable x_1 and the new vectors of variables X_1 and X_2. The division takes place because monotonicity analysis determines that x_1 might have a critical effect on the objective function. This division does not take place indefinitely. 1[st]PRINCE uses inductive inference to identify trends in the constraint activity, and uses that information to extrapolate the current design space to the final design space.

5.2.1 A structural design example
Consider a circular solid beam under a transverse load as shown in Figure 6. This design problem is input to 1[st]PRINCE as the following optimization problem:

$$\text{(a) } W = \int_0^L \pi\rho r_1^2 dx = \pi\rho L r_1^2$$

$$\text{(b) } \sigma = \frac{Mc}{I} = \frac{4Px}{\pi r_1^3}$$

$$\text{(c) } \tau = \frac{PQ}{It} = \frac{4P}{3\pi r_1^2}$$

$$\text{(d) } \tau \leq \tau_y$$

where ρ is the mass density, x is the distance along the beam of length L, P is the transverse load, M is the bending moment (equal to Px), c is the maximum distance from the neutral axis (equal to r_1) I is the moment of inertia, Q is the first moment, and t is the width.

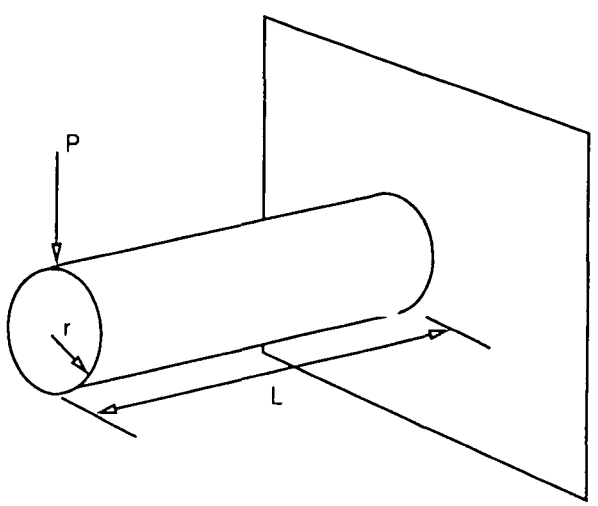

Figure 6 A circular beam under transverse load

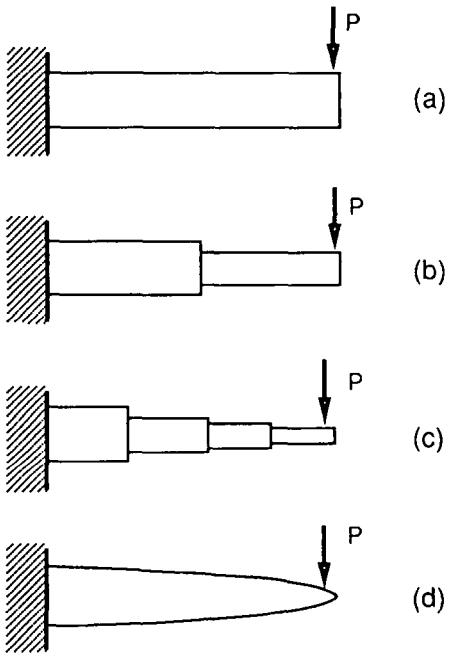

Figure 7 Inductive inference of the optimal beam

Consider the optimization of W based on the transverse load only. The integral (a) can be divided into two integrals as $W = \int_0^L \pi\rho r_1^2 dx = \int_0^{x_1} \pi\rho r_1^2 dx + \int_{x_1}^L \pi\rho r_1^2 dx$. Solving the optimization problem for x_1 1stPRINCE gets $x_1 = L/2$ (Figure 7(b)). Further division of the integral leads to Figure 7(c). At this point, 1stPRINCE inductively infers the trend and comes up with the final design, as in Figure 7(d).

6 Conclusions

In order to innovate one must have an open mind. One must be willing to relax one's constraints to examine off-beat alternatives. Innovation comes from one's ability to break the rules, to look beyond the norms, and to avoid mind-sets. In a computational framework, exploratory behaviour can be achieved either by mutation-based approaches or by purposeful analytic approaches. Exploratory behaviour has to be achieved by striking a balance between the need to keep the search space narrow and focussed, and the need to expand the search space in order to uncover hidden solutions.

An exploratory process, by its very nature, will tend to produce a very large proportion of alternatives that are useless and even nonsensical. Systems that explore, spend a lot of time looking down paths that eventually lead to dead ends or uninteresting solutions. We also pointed out that innovative designs are not necessarily obtained through some deliberate attempt at producing them, but by generating lots of alternatives and simply throwing away the bad ones. Consequently, the ability to explore depends on the diversity and variety of the alternatives generated by exploration. It is not enough to have a system that produces vast numbers of very similar designs.

A successful exploration should have an end. If the system is supposed to be looking for an innovative solution, then it cannot have any prior notion of what that solution looks like. So, how will the system know that it has found something innovative when it stumbles upon the alternative during exploration? This problem can be addressed by using a memory of prior designs, experiences, and episodes. For example, if one can associate a given alternative to some successful prior design, then one can judge the given design as also having favourable characteristics. Association can also bring new criteria from other designs and domains. The ability to recognize such opportunities can lead to serendipitous innovation. Hence, it is not sufficient to evaluate the

results of exploration with only a pre-defined set of design criteria. In reality, criteria emerge and goals change during the design process.

We reviewed four strategies for exploring beyond the solution space of a given design culture. The first set of approaches adopted a generate and test paradigm with special heuristics to guide the search for novel combinations. The success of these approaches is predicted on the fact that the search space is so large that the computer can stumble upon combinations never before thought of by humans. We then examined natural selection by mutation as another way of exploring the A-space. The third set of approaches were discovery systems that are aimed at finding regularities, patterns or an unexpected concordance in the data provided to it. The important contribution of discovery systems are methods that help alert the system when it makes a discovery. Finally, we examined approaches that reason about physical laws and principles to analytically derive appropriate modifications (mutations).

In this two-part paper we have argued that *exploration* and *association* are the two fundamental ways in which design automation systems can innovate. We have discussed a variety of exploration and association methods, some of which have been used in actual design systems, and others that point to the solution of some open questions in design research.

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. Design by 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. Exploration and association go hand in hand. Generally speaking, the process of exploration generates a wide variety of alternatives that can be evaluated by association to prior experiences. Associational methods can also be part of the exploration process. New design alternatives can be generated by drawing knowledge from analogically related prior designs and experiences.

Innovation is not as mysterious as it is made out to be. We are now beginning to understand some of its underlying processes. We have uncovered some components of those processes that can now be incorporated in computer aided design tools which can help designers explore alternatives and exploit relevant prior art.

References

Addanki, S. and Davis, ES, 1985. "A representation for complex domains". In *Proc. 9th Int. Joint Conference on Artificial Intelligence.*

Blum, RL, 1982. "Discovery and representation of causal relationships from a large time-oriented, clinical database: The RX project". In: Lindberg, DAB (ed.), *Medical Informatics, Volume 19.* Springer-Verlag.

Buchanan, BG and Fiegenbaum, EA, 1978. "Dendral and Meta-Dendral: Their application dimension". *Artificial Intelligence* **11** (1) 5–24.

Cagan, J, 1990. Innovative Design of mechanical structures from first principles. PhD thesis, Univ. of California.

Cheyayeb, F, 1987. A framework for engineering knowledge representation and problem solving. PhD thesis, MIT.

Coyne, RE and Subrahmanian, E., 1991. "Computer supported creative design: a pragmatic approach". In: Gero,J and Mahler, M (eds.), *Modeling Creativity and Knowledge Based Design.* Lawrence Erlbaum.

Coyne, RD, Rosenman, MA, Radford, AD, Balachandaran, M and Gero, JS, 1989. *Knowledge-Based Design Systems.* Addison Wesley.

Darwin, C, 1859. *On the Origin of Species by Means of Natural Selection.* John Murray.

Dyer, MG, Flowers, M and Hodges, J, 1986. "EDISON: An Engineering Design Invention System Operating Naively". In: *Proc. 1st Int. Conference on Applications of AI to Engineering.*

Flemming, U, Coyne, R, Glavin, T, Hung, H and Rychener, M, 1989. *A generative expert system for the design of building layouts EDRC 48-15-89.* Technical Report, Engineering Design Research Center, Carnegie Mellon University.

Grefenstette, JJ, 1987. *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* Morgan Kaufman.

Koestler, A, 1964. *The Act of Creation.* McMillan.

Langley, P, Simon, HA, Bradshaw, GL and Zytkow, JM, 1987. *Scientific Discovery—Computational Explorations of the Creative Processes*. MIT Press.

Lenat, DB, 1976. *AM: An artificial intelligence approach to discovery in mathematics as heuristic search*. PhD thesis, Stanford University.

Lenat, DB, 1983. "EURISKO: a program that learns new heuristics and domain concepts, The nature of Heuristics III: program design and results". *Artificial Intelligence* **21** (1,2).

Lenat, DB, 1984. "Why AM and EURISKO appear to work". *Artificial Intelligence* **24** 269–294.

Muller, ET, 1987. *Daydreaming and computation: a computer model of everyday creativity, learning, and emotions in the human stream of thought*. PhD thesis, Univ. of California.

Murthy, S and Addanki, S, 1987. "PROMPT: An innovative design tool". In: *Proc 6th National Conference on Artificial Intelligence*. 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, 1991. *Exploration and Innovation in Design: Towards a Computational Model*. Springer-Verlag.

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

Papalambros, P and Wilde, D, 1988. *Principles OF Optimal Design*. CUP.

Popper, K, 1965. *Conjectures and Refutations; The Growth of Scientific Knowledge*. Harper & Row.

Quillian, RM, 1968. "Semantic memory". In: Minsky, M (ed.), *Semantic Information Processing*. MIT Press.

Spencer, H, 1857. "Progress: its law and cause". *Westminster Review*.

Toulmin, S, 1967. "The evolutionary development of natural science". *American Scientist* **57**.

Walker, MG, 1987. "How feasible is automated discovery". *IEEE Expert* 69–82, Spring.