Zeitschrift: IABSE surveys = Revue AIPC = IVBH Berichte

Band: 14 (1990)

Heft: S-45: Knowledge-based expert systems: past, present and future

Artikel: Knowledge-based expert systems: past, present and future

Autor: Garrett, James H. Jr.

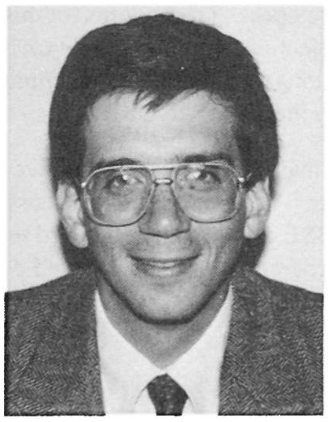
DOI: https://doi.org/10.5169/seals-933

Knowledge-Based Expert Systems: Past, Present and Future

Systèmes experts basés sur la connaissance: passé, présent et avenir

Wissensorientierte Expertensysteme: Gestern, heute und morgen

James H. GARRETT, Jr
Assist. Professor
University of Illinois
Urbana, IL, USA



James Garrett, born in 1961, got his BSCE, MSCE and Ph. D. from Carnegie Mellon University in Pittsburgh, PA. He joined the faculty at the University of Illinois in 1987 and has been performing research in the areas of standards processing, object-oriented building modeling and neural networks.

SUMMARY

Knowledge-based expert system (KBES) technology was presented to the international structural engineering community in the early 1980's and experienced an exponential growth of application to Civil Engineering problems in the latter part of that same decade. The intent of this paper is to discuss what has happened since Fenves, Maher and Sriram described this technology in IABSE Periodica in 1985, with emphasis on what transpired at a recent colloquium titled Expert Systems in Civil Engineering held in Bergamo in 1989. During that colloquium, there were many interesting presentations made by various members of the international community on applications of this technology to various Civil Engineering problems. The current state of application will be summarized in this paper along with several state-of-the-art technologies that were discussed, or alluded to, within the colloquium.

RÉSUMÉ

La technologie des systèmes experts basés sur la connaissance (KBES) a été présenté à la communauté internationale du génie civil au début des années 1980, et a subi dès lors un développement exceptionnel dans des applications en génie civil. Cet article passe en revue l'évolution depuis la publication de Fenves, Maher et Sriram dans les PERIODICA AIPC en 1985. Il traite aussi du Colloque tenu à Bergame en 1989 sur le thème des systèmes experts en génie civil, au cours duquel de nombreuses applications pratiques ont été présentées. L'article résume l'état actuel des connaissances dans ce domaine, tel qu'évoqué ou détaillé lors du Colloque.

ZUSAMMENFASSUNG

Die Technik wissensbasierter Expertensysteme (KBES) wurde der Internationalen Baufachwelt in den frühen 80er Jahren vorgestellt und erlebte eine exponentielle Zunahme der bauspezifischen Anwendungen in der zweiten Hälfte des Jahrzehnts. Der vorliegende Beitrag möchte die Fortschritte diskutieren seit der Beschreibung der Technik durch Fenves, Maher und Sriram in den IVBH PERIODICA 1985. Dabei liegt das Schwergewicht auf dem 1989 in Bergamo veranstalteten Kolloquium «Expert Systems in Civil Engineering», wo unter internationaler Beteiligung viele interessante Anwendungen auf unterschiedliche Probleme des Bauwesens vorgestellt wurden. Der gegenwärtige Stand der Techniken und ihres Einsatzes, wie sie auf dem Kolloquium diskutiert oder erwähnt wurden, ist nachfolgend wiedergegeben.

1. Introduction

In 1985, an article appeared in IABSE Periodica that described an emerging technology for solving ill-structured problems using heuristic search techniques known as knowledge-based expert systems (KBES) [11]. In that article, the authors described 1) the nature of knowledge-based expert systems, 2) the range of applications possible with KBES technology, and 3) the implications of this technology in education, practice and research.

Judging from the technical literature, both within the United States and abroad, much has happened in applying KBES technology to various problems within civil engineering since that article was published. The purpose of this paper is to "stand back," and review what has happened in the intervening years since 1985 and to discuss what the future appears to offer for KBES developers. The impetus for this paper was a colloquium recently held in October of 1989 in Bergamo, Italy on the subject of *Expert Systems in Civil Engineering* [1]. During that colloquium, a wide range of applications was presented along with some newer techniques for knowledge representation, inference, and machine learning. This paper is also meant to serve as a quasi-summary of that colloquium.

In the first section, a review of the main points made by Fenves, et al., is conducted so as to build an historical foundation for the rest of the paper. The next section discusses the breadth of applications within structural engineering described within the IABSE colloquium. The third section briefly describes a few emerging techniques, such as object-oriented modeling/programming, model-based or qualitative reasoning, analogical or case-based reasoning, machine learning, and neural networks, that are beginning to be applied in KBES development.

2. Knowledge-Based Systems - A Brief Look Back

Although the concept of knowledge-based systems is actually "old" by today's software time-scale (it came on the computer science scene in the early 1970's), it has taken the engineering community several years to understand and grasp the potential of this technology. In 1985, Fenves, et al., authored a paper within the IABSE Periodica describing the nature of, possible application of, and implications of this technology[11]. At that time, the technology was predominantly a rule-based technology, where they described the main components of a KBES as:

- a knowledge-base — a collection of IF-THEN rules that represent pieces or "chunks" of decision-making knowledge that are applied during the development of a solution to a problem;
- a context or working memory — a formal data structure for representing the initial information known about the problem as well as the evolving solution being developed through application of the rules in the knowledge base; and
- an inference engine — a pattern-matching program for applying the "chunks" of knowledge in the knowledge base to the problem (and its partial solution) described in the context.

A knowledge-based system has several key qualities that differentiates this form of computer program from its procedural counterparts that all stem from its separation of knowledge and its usage: transparency of the knowledge applied, transparency of the solution being developed, and a capacity for incremental development.

Because the rules are developed and represented as separate "chunks" of knowledge, each hopefully specifying the preconditions under which it is applicable, those pieces can be individually observed and understood thus leading to the claim of transparency of knowledge applied. In a procedural program, the pieces of knowledge used are inextricably intertwined with the control of these pieces of knowledge and thus one must look at the entire program to get a sense of the knowledge within.

Because the inference engine of a knowledge based system searches over this set of rules and keeps track of what rules match the context at what points in the solution process, it is possible to reconstruct the evolution of the solution process thus leading to the claim of transparency of solution. A procedural program normally does not keep track of the path it followed during the solution of a problem and only issues its final answer.

Finally, because the knowledge that is represented within the knowledge base can be both general and much more specific and one can rely on the inference engine to "figure out" which to apply, it is possible to develop a knowledge base that consists mostly of general rules in the early stages of system development and is then augmented with more specific rules as the knowledge acquisition process matures. Thus, the system will perform at a reasonable level of proficiency, but not expertly, in the early stages of development. It will not be capable of more expert problem solving capabilities until the more specific, exceptional pieces of knowledge have been added to its knowledge base. This represents the concept of incremental growth capability. Most procedural programs do not allow for more than one level of problem solving knowledge.

In their paper, Fenves, et al., stated that the range of application of knowledge-based systems was quite broad and spanned from derivation-type problems (e.g., diagnosis, monitoring, and interpretation) to formation-type problems (e.g., planning and design) [11]. At that time, they mentioned that knowledge-based systems were still in their infancy and listed several prototypical systems developed within civil engineering domains that spanned the derivation-formation spectrum, such as: HI-RISE[27], a system for the preliminary design of high-rise buildings; TRALI[50], a system for the design of signalization for traffic intersections; HOWSAFE[26], a system for assessing the capacity for safety within a construction firm; and CONE[32], a system for interpreting cone penetrometer data.

In the years between the publication of Fenves' paper and the IABSE Expert System colloquium held in Bergamo in October, 1989, the technology has reached a greater level of application as more people became aware of its capacity to solve ill-structured problems requiring the formalization and application of heuristics. In a book published in 1987 by the American Society of Civil Engineers, edited by Maher [28], a large number of civil engineering applications of knowledge-based technology are described and the potential for KBES applications within all aspects of civil engineering is clearly illustrated. The next section describes some of the most recent applications of this technology presented at the IABSE Expert Systems colloquium. Many of these applications illustrate that the technology is moving from the stage of "skepticism and discovery" to "acceptance and employment". Many of the systems are now being developed by industrial members of the civil engineering community and are very much intended for use in their daily operations.

3. Knowledge-Based Systems - Bergamo 1989

As an illustration of the breadth of application of this technology within the field of structural engineering, this section summarizes the applications reported at the 1989 IABSE Colloquium on Expert Systems in Civil Engineering [1]. The applications can be classified into several areas: 1) systems for aiding in the design of structures; 2) systems for aiding in the analysis and damage assessment of structures; and 3) systems for aiding in the construction, operation and maintenance of structures.

3.1. Systems for Design

Several systems were described for assisting the civil engineer in one of the primary aspects of his profession — design. The following are brief summaries of a few of the design-oriented KBES presented in Bergamo:

- EKSPRO — A system for designing energy efficient buildings. Using knowledge-based technology, this system integrates a 3-D CAD environment with rule-based representations of governmental regulations for materials, HVAC, lighting, and occupational health and user-defined constraints. The result of using this system is a building design that meets the requirements embodied within the knowledge base of the system. The system is currently being used by several firms within Denmark [34].
- A system for cost effective design of glue-laminated structural components. Using formal knowledge representation techniques for the design standards for these glue-laminated members, the system determines the most cost-effective design of a 2-D structural frame that meets the specifications given the clear height dimensions of the building, its geographical location, and current material prices [48].
- WINDLOADER — A system for ensuring the correct interpretation and usage of the Australian wind and building codes. This system represents the information within the codes using rule-based technology, and helps the user of these codes determine which rules are applicable and which are irrelevant. The system is currently being used in design offices throughout Australia [44].
- ARCHPLAN - A system that assists in the development of the conceptual design of high-rise office buildings. Given as input the site description, program of usage of the building, budget, and geometric restrictions on the building, ARCHPLAN produces a three-dimensional description of the space decomposition within the building, including the function for each of these spaces, e.g., mechanical, office,

circulation, etc. The knowledge is stored in either algebraic form or as heuristic rules. The system is currently integrated in a much larger, vertically integrated building design environment [42].

In addition to the above described applications, several researchers described more general models of design: 1) Bento, et al., described a model that uses frames and first order logic for describing the state space transition of design [3], and 2) Maher described a a software environment, called EDESYN, for the development of engineering design systems that consists of five main modules: a design knowledge base, a synthesis algorithm, a design context, a user interface, and a knowledge acquisition facility [29]. Several other researchers described formal models of engineering design standards: 1) Garrett described an object-oriented model (section 4.1. describes the term object-oriented programming) for the representation and automated usage of the information and logic found within engineering design standards [14], and 2) Feijo, et al., described a model based on first order logic for the representation of a design standard [10].

3.2. Systems for Analysis

Several systems were described for evaluating the performance, or damage, of a structure from a variety of perspectives, such as structural, fire-resistance, and earthquake resistance. The following are brief summaries of a few of the analysis-oriented systems presented:

- A system for fire vulnerability analysis. As the authors describe, the assessment of fire vulnerability must consider environment, people, goods, and the building itself. This system first performs the task of identifying hazards and their likelihood of occurring, and then for the identified hazards that are likely to occur prescribes suitable protection measures and assesses the financial ramifications of such procedures [6].
- A system for seismic risk evaluation and retrofit. Again, this system has two components: a diagnosis component in which the seismic risk of a building is assessed; and a debugging or design component that helps the designer develop risk abatement procedures. This system makes extensive use of an object-oriented model of the structure and its possible behaviors for performing qualitative reasoning at different levels of detail about the seismic risk of a building [4].
- AMADEUS — A system for the assessment of earthquake damaged buildings. This system is intended to assist during a post earthquake period in which many buildings must be quickly, and correctly, assessed for safety. It provides detailed instructions to novice inspectors (which is definitely the case during an emergency) for the tasks of surveying and evaluating the extent of structural damage to masonry structures [33].
- A system for assessing the damage state of a reinforced concrete bridge deck. This system applies fuzzy production rules based on fuzzy set theory to deal with the inherent uncertainty and fuzziness in inspection reports. The system first assesses possible states of damage, then diagnoses the cause of these apparent damage states, and then predicts the remaining life left in the concrete deck [13].

In addition to the above systems which are aimed at a specific type of analysis, several other researchers described more general applications of emerging AI technologies to general finite element analysis techniques: 1) Rehak and Baugh describe an object-oriented approach to finite program development which allow developers of such programs to describe elements and analysis techniques at a much higher, more meaningful level of abstraction, leaving the details of implementation and representation hidden within the objects they are manipulating[37]; and 2) Steimer and Forde describe the application of knowledge-based programming techniques for solution process control and optimization within the finite element method using the KBES to select and apply appropriate analysis computations [46].

3.3. Systems for Construction, Operation and Maintenance

Several systems were described for providing assistance during the construction stage or in the operation/maintenance stage. The following are brief summaries of a few of the construction management-oriented or maintenance-oriented KBES presented in Bergamo:

- An advisory system for site managers. This system has been developed to help site managers supervise "all incoming and outgoing information, the costs and to a certain point the technical problems of a site." The system helps the site manager identify other analysis and decision support tools that should be brought to bear on identified important decision points. The system has been developed using an object-oriented approach within three different modules: cost, resource, and administration [16].

- A system for managing structures in service. This system applies knowledge-based techniques for organizing relevant knowledge of, and providing advice to responsible agents on, several aspects of in-service management, such as: monitoring, evaluation, maintenance, and modification of existing structures. A prototype of such a system for a small part of the whole domain was developed to provide assistance to engineers having to make a decision about cracks appearing in steel structures. The prototype was developed in a simple KBES shell, but the author acknowledges that to attack the larger, overall system, hybrid knowledge representation and reasoning capabilities are necessary [45].
- A system for assisting in the technical inspection of waterproofing on flat roofs. This system was developed due to the high amount of damage incurred in buildings for which improper inspection of the flat roof waterproofing system was performed. Hence, the objective in building this system was to capture the necessary knowledge for performing an adequate inspection of such systems. The bulk of the knowledge used in this application is of a regulatory nature and hence a model of design regulations was also developed as a part of this project [35].
- REPCON — a system for assisting in the repair of concrete structures. This system is intended to assist the structural engineer in assessing the state of deterioration within a concrete structure and to propose different repair procedures. Given a description of the various components within the building, including any visible symptoms of damage, the system then diagnoses the cause of this damage. After performing this diagnosis, knowledge is applied to determine which repair procedure is most appropriate for the diagnosed cause of damage[38].

In addition to the above applications, Comerford, et al., reported the development of a KBES methodology for interpreting signals from instrumentation used in monitoring. The method brings together data from instrumentation, qualitative descriptions of the site, history of construction, constructor, etc., and heuristic knowledge in order to properly interpret the state of the system being monitored. This system was demonstrated via two prototypes: 1) the first being used to interpret signals from a non-destructive test of a concrete pile to determine the integrity of that pile; and 2) the second being used to interpret the data emanating from hundreds of sensors on an earth dam to detect such things as global drifts [8].

4. Knowledge-Based Systems - Emerging Methods and Technologies

Several successful applications of the KBES technology were described in section 3. However, while developing these systems, and many others like them, the KBS community has come to realize several important limitations of the knowledge-based technologies of the last decade:

- knowledge is not homogeneous in structure and many different knowledge representation paradigms (frames, rules, procedures, formal logic, neural networks, etc.) are needed in order to adequately represent the problem solving knowledge brought to bear on a particular problem;
- knowledge is not homogeneous in level of abstraction and knowledge at various levels of detail is needed in order to develop preliminary, high-level solutions to problems, as well as more detailed solutions, at various stages in the problem solution process;
- knowledge acquisition is an enormously difficult, time-consuming, error-prone task, is the most important task in developing a knowledge-based system, and is one area in which tools for assistance are most needed; and
- most current knowledge-based systems are only static "snapshots" of the knowledge used in problem solving and must be made capable of self-modification over time and as more experience is gained.

Hence, KBS developers have looked toward researchers in AI and other fields of Computer Science to help them solve these limitations. As we enter the 1990's, several methodologies and technologies are receiving a lot of attention and represent some of the advancements that will eventually find their way into mainstream knowledge-base system development: object-oriented knowledge representation, model-based (qualitative) reasoning, cased-based reasoning, machine learning and neural networks. The following sections describe each of these emerging technologies in more detail and describes how each are being, or might be, applied to solve civil engineering problems.

4.1.Object-Oriented Programming

Object-oriented programming methodologies are now being used for organizing and representing various forms of knowledge, where within an object, one can represent and apply both rules and procedures in ways that were impossible, or at least extremely difficult, in former strictly rule-based paradigms. One can represent relationships between concepts that were formally possible, but much more cumbersome, or impossible within the pure rule-based paradigm. The inherent hierarchical structure of object-oriented representations has also made the maintenance of large rule bases much more tractable.

The basic building block of an object-oriented representation is the *object* — a modular, self-contained collection of descriptive attributes and the methods (procedural or rule-based) for manipulating those attributes. Representation in an object-oriented environment first requires the description (declaration of attributes and methods) of the general types of objects that populate the domain (class objects), and then requires the generation of instances of the class objects to describe the particular entity being modeled.

In object-oriented representations, everything is an object. Objects can represent concepts, physical objects, processes, etc. In all cases, the object possesses a set of attributes and relationships, both of which are represented as slots in the object. Attributes represent descriptive pieces of information about the object and may be represented by either a static value or a method, where a method describes how the value of an attribute is computed. Methods may also cause side-effects in other objects. Relationships, also represented using slots, represent links to other objects. For example, an object may have a "next-to" slot that is filled with the name of another object representing the fact that the two objects are next to each other. Fig. 1. shows the typical structure of an object and an example of a **w-shaped-section** object. In that example, the first slot is a relationship, the second through seventh slots are attributes with static values, and the eighth slot is an attribute with a method for computing its value.

ObjectName
- SlotName: SlotValues
  - *FacetName: FacetValue*
  - *FacetName: FacetValue*
  - *FacetName: FacetValue*
- SlotName: SlotValues
  - *FacetName: FacetValue*
  - *FacetName: FacetValue*
  - *FacetName: FacetValue*

w-shaped-section
- is-a: hot-rolled-steel-section
- length:
  - *units: feet*
- depth:
  - *units: inches*
  - *if-changed: depth.if-changed.attachment*
- flange-width:
  - *units: inches*
- flange-thickness:
  - *units: inches*
- web-thickness:
  - *units: inches*
- moment-of-inertia:
  - *units: inches*^{4}
  - *if-needed: moment-of-inertia.if-needed.attachment*
- moment-capacity: w-shaped-section.ultimate-moment.method
  - *units: inch-kips*

Figure 1. Example of an Object

Other objects can access the attributes and relationships of an object by *sending a message* to the object that "owns" the attribute or relationship. In addition to having a value or method, the slots of an object may also have self-descriptive information, such as permissible range or type of value. This information is stored in *facets* that are associated with the slots. The attribute slots of **w-shaped-section** (shown in Fig.1.) all have facets for storing the units associated with the value placed in the slot. A special type of facet, called a procedural attachment or demon, watches a slot value and executes a method when that value is added, changed, or erased. For example, in Fig. 1. the slot "depth" of **w-shaped-section** has a procedural attachment that is activated when the value of depth is changed; this procedural attachment nullifies the value stored in the slot "moment-of-inertia." In addition, the slot "moment-of-inertia" has a procedural attachment that is activated when a value is needed and none exists; the value of "moment-of-inertia" will be re-computed, after having been nullified, the next time the value is requested. Thus, the procedural attachments help maintain consistency between the slot values within an object or set of objects. This event-driven nature of object-oriented environments is especially well-suited for maintaining consistency between concept instances present within the context (working memory) of the knowledge-based system. This type of behavior is difficult to achieve in a monotonic, purely rule-based environment.

A common practice in object-oriented programming is to develop templates for types of objects, commonly called *class* objects. These class objects usually possess attributes, attribute values, methods, and procedural attachments that are common to several more specific objects. If these more specific objects are themselves templates for other even more specific objects, they are called *subclass* objects. Objects that represent a specific instance of a class or subclass object are called *instances.* Instances are *children* of subclasses (or classes), subclasses are children of classes (or other subclasses), classes are *parents* of their subclasses and instances and subclasses are parents of their instances or other subclasses. These parent-child relations are important because in object-oriented programming environments, children automatically *inherit* attributes, attribute values, methods, and procedural attachments from their parents. For example, all instances of the object **w-shaped-section** (shown in Fig. 1.) will inherit the slot names "length", "depth", "flange-width", etc., and their associated facets from this class object. Through inheritance, it is possible to represent information at an appropriate level of object generality and have all more specific subclasses and instances of objects inherit that information, thus reducing redundancy and improving consistency. For example, **w-shape-section** actually inherits the slots "length" and "depth" from its parent object, **hot-rolled-steel-section** (not shown in Fig. 1.); these two attributes are stored in **hot-rolled-steel-section** because all subclasses and instances of hot-rolled steel sections have a length and depth attribute. In fact, these attributes may be stored at an even higher level within the hierarchy, such as within the parent of **hot-rolled-steel-section.**

Hence, the key ideas of object-oriented representation are that objects possess attributes and methods, can inherit attributes and methods from other objects, and communicate with each other (i.e., get data or execute an object's method) only by sending messages.

While, the most popular way to represent domain-dependent behavior is with production rules, they alone are inadequate for representing domain object definitions and their static relationships. It is common today to use an integrated approach to knowledge representation, where objects represent the natural structure of the physical objects and abstract concepts within the domain that are to be reasoned about, and rules are used to represent the decision-making knowledge (i.e., the derivation of object attributes that are conditional in nature). Objects provide a powerful foundation for a rule-based inference by providing:

- a powerful language for describing domain objects that can be used within rules,
- a set of inference mechanisms (inheritance and ValueClass/cardinality checking) that can automatically reach a lot of conclusions that are needed by a rule, such as class membership and default values, and
- a powerful and flexible language for defining the rules themselves.

Because the information contained in an object's slot is available to inference mechanisms, a rule could reason about the characteristics of an object by referring to its slot values. For example, the following rule is meant to apply to all subclasses and instances of the object **hot-rolled-steel-section** (symbols with a ? denote rule variables and x.slotname denotes the slot of object x) :

```
IF    ?member is-one-of hot-rolled-steel-section
AND   ?member is-part-of ?main-structural-system of ?building
AND   ?building is-one-of high-rise-office-building
THEN  ?member.insulation = sprayed-on-concrete
```

Note that this rule does not have to be written for all classes and subclasses of hot-rolled-sections, but rather can use the class membership hierarchies present with the object-oriented description of the domain. The class membership hierarchy also provides a convenient and natural mechanism for partitioning, indexing, and organizing a system's massive collection of production rules. Production systems, like conventional programs, need to be organized into small, easily managed modules (groups over about 50 rules become difficult to manage and maintain). A group of rules can be associated with a class of objects by having the set stored in the slot of an object and invoked from within that object either by a method or procedural attachment.

Many people have wrongfully concluded that object-oriented programming is nothing more than the development of programs using subroutine calls to a core set of subroutines. While this statement might have a grain of technical truth, it misses point. Object-oriented programming is a methodology for structured programming in which the object is the mechanism of structuring, not the procedure. The advantages of of an object-oriented representation are: 1) the methodology greatly aids in structuring the knowledge; 2) it enables rules and procedures to be more generic making the knowledge-base easier to understand (one can write rules based on a class of objects and have them apply to all subclasses); 3) it compartmentalizes the knowledge, thus reducing the complexity; 4) through graphical display of attributes and relationships, the knowledge base is more understandable; 5) because of the highly structured nature of the knowledge, it is much more easily maintained; 6) it facilitates knowledge acquisition through concept refinement; 7) the taxonomic relationships among objects (parent-child relations) enable descriptive information to be shared among multiple objects using inheritance (getting information about child from parent frame); and 8) through procedural attachment, constraints between frames can be automatically maintained.

4.1.1.Applications of Object-Oriented Programming

Many researchers are investigating the usage of object-oriented programming techniques to civil engineering problems. As stated previously, Rehak and Baugh have developed an object-oriented approach to finite element program development [37]. In Rehak and Baugh's system, the concepts and subconcepts, such as stiffness matrices, material models, loads, etc. are all represented at several levels of abstraction using the concepts of classes and subclasses. The finite element program developer is able to concentrate at the level of abstraction that is appropriate for describing the finite element relationships (Rehak states that the finite element method as originally described was only about a page of mathematical equations, but current implementations of that method are hundreds of thousands of lines long) and can keep the details of implementation of such concepts hidden within the class objects with which he is expressing his relationships.

Garrett is applying object-oriented programming techniques to the development of a formalized, modular model of engineering design standards and specifications[14]. In that model, all aspects of the standard are represented as objects and the behaviors of these objects are hidden within their class descriptions. The advantage of using an object-oriented approach is again that the details of modeling a standard are hidden from the standard modeler; he need only know what kind of object he is trying to represent, selects the appropriate class object, represents the unique attributes for that object, and then inherits all evaluative behavior from the class description. In addition, if the set of objects are insufficient for the piece of specification he wishes to model, he may create a new object (likely a subclass of an existing object) and use it without affecting the operation of the rest of the objects in his model. This flexibility yet ease of use is the hallmark of most object-oriented software environments. The description of the above two projects is only meant to be an illustration of the possible applications of this methodology; it is acknowledged that many other researchers working in this area have not been mentioned in the interest of space.

4.2.Model-Based Reasoning

Model-based reasoning, also known as qualitative reasoning, provides developers of knowledge-based systems with a tool for assisting both knowledge acquisition and reasoning at multiple levels of abstraction. Model-based reasoning is based on the way in which humans initially develop causal models for, or initially reason with, complex systems — they do not start with the detailed descriptions of the causal relationships about the components of a complex system, but rather, reason qualitatively about the way in which these components interact to get a sense for how the system will behave. Hence, model-based reasoning involves building a qualitative model of a complex system, consisting of a set of identified subcomponents and their qualitative causal interrelationships (i.e., A causes B to increase), and then using this model to: 1) qualitatively reason about effects given observations about existing symptoms (causes); 2) qualitatively reason about possible causes given observations about existing conditions (effects); or 3) developing more detailed causal relationships between causes and effects that can be used to predict values of quantities instead of just qualitative trends thus guiding the knowledge acquisition process. When the more detailed causal relationships are known, the qualitative model is not discarded, but remains as a more abstract description of the system which can be used to reason about the system in earlier stages of system development when many of the system details are yet to be determined. The existence of both qualitative and detailed descriptions of causal relationships thus provides the multi-level reasoning capability described as the second deficiency of current knowledge-based systems.

Harandi and Lange have developed mechanism for describing, and reasoning with, qualitative domain models of complex systems that consists of two primary constructs: *components* and *causal flows* [21]. In their qualitative modeling environment, a complex system is decomposed into its components, which may be high-level subsystems that are further decomposed into smaller subsystems. To the top level of the model, this complex subsystem appears to be a component, while to the lower level of the model, the subsystem appears to be a complex system itself. The detail to which a system is decomposed into components depends on for what the model is to be used. After the system has been decomposed into components, causal flow lines are defined between the components at the same level within the model. Each flow line is assigned a type from a causal type hierarchy that describes what is flowing over the causal flow line. Harandi and Lange use as an example a pump (component) which has two inputs (type fluid and power) and one output (type fluid). In addition to components and causal flows, each component possesses a *causal equation*, which describes the relationship between causal inputs and outputs. Initially, each causal flow is assumed to only possess knowledge about its change in quantity (i.e., +, -, 0, or ?) and not is actual value and the causal equation is written in terms of these derivatives. The default causal equation is a simple linear addition of the derivatives of the inputs — e.g., power (+) + fluid (+) = > fluid (+). However, at later stages of model development, flows may actually possess, and causal equations may be expressed in terms of, actual quantities. Even with the default causal equation based only on linear relationships of change, one can discern what the effect on the output of fluid will be when the power remains unchanged, but the amount of input fluid is increased. Note, however, that this simple model cannot be used to determine the effect when one input is increased (e.g., power = +) and one input is decreased (e.g., fluid = -).

A more complex qualitative model is then built by combining the components via their causal flow lines. By assigning a type to each causal flow, and then restricting connection between components to have compatible types, the model can be automatically refined and consistency of the model can be enforced. For example, a water reservoir that possesses an output flow of type water can be connected to a a pump at its input flow of type fluid, which causes the pump to become a water pump and its output causal flow to become type water. This is possible because water is a type of fluid. However, if one tries to connect a water reservoir to a gasoline pump, the output flow of the reservoir (type water) is not compatible with the input flow of the pump (type gasoline), because although water and gasoline are both fluids, they reside along different branches of the fluid hierarchy and are thus incompatible. The modelling environment prevents incompatible connections. Thus, in addition to such a model being useful for performing diagnostic causal reasoning, such a reasoning system would also prove extremely useful in the aggregation, or composition, type of design activity. Once the set of components have been properly connected, it can be used to reason from causes to effects or from effects to causes. When an expert reviews the results from either of these reasoning exercises, he may discover inconsistencies, or the inability of the model to reason from opposite changes in inputs (+,-), and at that time may modify the causal equation for a component. In this way, the model is refined and thus assists in knowledge acquisition, too.

4.2.1. Application of Model-Based Reasoning

Because engineers are primarily concerned with the design and diagnosis of complex systems, it should come as no surprise that model-based reasoning is a very useful tool for assisting in engineering decision making. As an example of how this tool is being used in civil engineering, we again turn to one of the papers presented in [1]. Calvi, et al., developed a system for seismic risk evaluation and retrofit that was founded on a qualitative model of a structure[4]. This system has two modules: a diagnosis module in which the seismic risk of a building is assessed by reasoning from cause (seismic loading) to effect (dynamic response); and a debugging or design module that helps the designer develop risk abatement procedures by using qualitative reasoning from effects observed within the building to possible causes (i.e., parameters within some of the components of the building). Once the components within the building that should be modified to affect changes in the response have been identified through qualitative reasoning, the possible modifications are evaluated, detailed, and then propagated through the qualitative model to determine their actual effect on structural response. This system makes extensive use of an object-oriented qualitative model of the structure and its possible behaviors for performing qualitative reasoning, both from cause to effect and in reverse, at different levels of detail. This application is a very good example of how qualitative model-based reasoning can be employed in developing an effective and efficient engineering decision-making tool.

4.3.Case-Based Reasoning

Case-based reasoning, also known as reasoning by analogy, provides developers of knowledge-based systems with an alternative to the traditional distillation of knowledge into a set of static rules. Case-based reasoning is based on the way in which humans with lots of experience sometimes solve problems, i.e., by determining which of the myriad of past experiences is closest to the current problem and modifying the former solution to develop a solution to the current problem. Hence, the case-based reasoning paradigm takes a large collection of previous experiences, called cases, and determines which case is most similar to the current case. Having determined which case is most similar, and having identified which aspects are similar and which are different, the case-based reasoning system then transforms the past solution into one that is viable for the current situation. Having the ability to reason about the similarity of previous cases (and their solutions) with the current case being addressed and then to develop an analogous solution offers a dynamic problem solving paradigm that begins to address the fourth limitation of current knowledge-based systems.

Shank states that "reminding is a crucial aspect of human memory that has received little attention from researchers on memory [43]." However, in order for any memory system to be useful, it must facilitate the retrieval of past episodes within our memory that remind us of the current situation we are encountering. This is one of the primary problems being addressed by cased-based reasoning researchers: "how does a current situation remind us about the past situations we have encountered?" Shank identifies several different ways in which reminding occurs: 1) when a physical object reminds us of another physical object (a ball reminds us of the earth); 2) when a physical object reminds us of an event (a signed baseball reminds you of the time you met Babe Ruth); 3) events can remind you of a physical object (being at a commencement exercise reminds you of the diploma you possess); one event can remind you of another event in the same domain (a professional hockey game may remind you of the hockey game you played as a child) ; and an event can remind you of another event in a different domain (a commencement exercise may remind you of the thesis defense that preceded that commencement) [43]. Shank states that "reminding occurs as a natural part of understanding new situations in terms of previously processed situations. ... We use what we know to help us process what we receive [43]." He goes on to state that there are two primary ways in which an expert system could be developed: 1) recording the rules that the expert developed from his vast number of experiences; or 2) recording and indexing the experiences on which the expert drew his rules and develop rules for identifying and modifying these past episodes to provide solutions for new situations — a case-based approach. The two main issues in the latter approach are: 1) how are the past episodes represented and indexed so that one can be appropriately reminded of a past episode by a current situation; and 2) how does one go about reminding oneself of past episodes given a current situation? [43] In other words, how does one go about performing a case-based inference?

Kolodner describes three steps in performing a simple case-based inference: 1) recall a previous case that is similar to the current case; 2) focus on the appropriate parts of that previous case as it relates to the current situation; and 3) use the appropriate parts of the previous case to derive an appropriate solution to the current case [25]. Howard states that the main components of a case-based reasoning system that begins to mimic the way experts make use of their vast collection of past experience are as follows: 1) a representation for a case that captures both the "what" and the "why" behind past problems and their solutions and a collection of case instances known as the case-base; 2) some mechanism for determining the similarity between the current problem to be solved and the cases in the case base (Howard refers to this as *access* and *mapping*); and 3) some mechanism for transforming the solution or solution strategy found in the similar case to the current problem, referred to as *analogous inference* [23]. Accessing a previous case involves probing past memory, stored as cases, to find the case that is most similar to the situation currently being addressed for which some decision is required. In fact, more than one case may be employed in developing a solution to the current problem; different cases may be similar to the current situation in different ways and together the subset of past cases provides enough past knowledge to solve the current problem [25].

In order to make effective use of past experiences and to determine the degree of similarity between past cases and the current situation, it is not sufficient to only represent the final design solution. What is needed about past cases are: the goals and subgoals of the past problem, the constraints imposed on that problem at the goal and subgoal levels, the solution derived for each of these goals and constraints, and the rationale or plan behind the solution derived for the goals and subgoals. It is usually the case that the goal hierarchy, constraints and rationale are represented using an object-oriented representation such as that described in section 4.1.; Thompson is currently working on an object-oriented methodology for representing design rationale [47]. If one only had the final design solutions represented, without decomposition or rationale, the case-base could only be applied when the current design problem is almost identical to one of the cases. However, with past cases and their rationale described, it is possible to define what similarity among past cases and current situation means. Kolodner defines a past case to be similar to a current situation if the past case has many of the same goals and most important constraints [25]. Howard states that the definition of similarity must vary with the class of problem being considered; for some problems it may be necessary to define similarity based on basic attributes while for others it may be necessary to define similarity based on derived quantities, such as performance based properties [23]. There is no currently agreed upon definition of similarity and Howard is likely correct in saying that the definition should be tailored to the class of problems being solved.

Once a past case has been identified as being similar to the current case under consideration, the next step is to determine what about the past case can be used in solving the current problem. This is what Howard refers to as mapping and what Kolodner refers to as focussing on appropriate parts of the case [23][25]. As stated before, Kolodner uses goals and constraints as the means by which parts of the past case can be identified as applying to the current case. The overall problem solver places a goal on a blackboard and the memory component retrieves a similar case as it relates to that goal. The system then focuses its attention on that goal and how that goal was achieved in the past case. In this way, a solution for a current problem is pieced together from past cases. However, once a particular case has been identified as applying to a particular goal or subgoal within the current problem, one step yet remains — what Howard refers to as the analogic inference and what Kolodner refers to as the case-based inference [23][25].

There are several ways in which the successful solution of a goal from a past case can be employed in the development of a solution to a similar goal in the current case [25]: 1) the solution used in the past case can be directly transferred to the new case; 2) the solution used in the past case can be modified for the new case based on the differences in constraints and subgoals between the past and current case; and 3) the method by which the solution in the past case was derived can be transferred directly to the current case and used to derive a solution for the current case. Hence, using the design representations and plans of past cases, a design plan and solution for a new design problem can be constructed.

Case-based reasoning systems provide an alternative to the traditional, static rule-based systems. One can easily see that as the case-base is built up with descriptions of past experiences encountered and successfully, or unsuccessfully, resolved, the performance of the system will drastically improve — a characteristic of systems capable of learning.

4.3.1. Applications of Case-Based Reasoning

Several researchers have investigated the usage of case-based reasoning to civil engineering problems. Howard and Rafiq have investigated the application of case-based techniques to the design of structural components in a system called RESTCOM [36]. In their system, Howard and Rafiq use a representation of past component designs that captures cross-section, boundary conditions, length, material properties, loading conditions, and external constraints. With this representation of their past design cases, they are able to search for cases that match the case under current consideration, where a match may be exact, almost exact, or a partial match with some of the attributes and conditions. From the description of their system, it did not appear that they performed any transformation on the past case found to be similar, but rather used method (1) as described by Kolodner.

Daube and Hayes-Roth have investigated the use of case-based reasoning to structural component *redesign* with a system called FIRST [9]. In their system, they not only require the representation of the problem conditions and final design description, but also require the representation of design plan (i.e., the set of goals and subgoals and the steps performed in the solution of these goals and subgoals). In addition to its representation of design plans, FIRST also represents its domain knowledge in a semantic network; all concepts and components dealt with by the system in the past are placed somewhere within this hierarchy. Given this semantic hierarchy, FIRST is able to determine the semantic relationship between the past case and the current case and uses this relationship in modifying design plans from a past case for use on the current case. They illustrate how the redesign plan for increasing the moment of inertia of a rectangular beam can be analogically transformed into a plan for increasing the moment of inertia of a circular beam, by realizing the the x and y dimensions of the rectangular cross-section are analogous to the diameter of a circular cross-section.

Zhao and Maher have investigated the use of case-based reasoning in assisting in the development of preliminary designs of high-rise buildings [49]. Their system, called STRUPLE, takes as input a high-level description of the building (which includes such information as the location, budget, gross area, dimensions, shape, number of stories, intended floor use, design live load, design wind load, etc.), and searches a case-base of past design experience and determines the systems and subsystems that should be considered during the synthesis of a preliminary design of the structure. Note that it does not perform the preliminary design, but rather focuses the synthesis module on a few plausible design subsystems. The past cases used by STRUPLE contain general information (name, city and state, etc.), geometric information (gross area, height, width, aspect ratio, etc), architectural specifications (number of stories, intended floor usage, etc), loading information (dead load, live load, etc), primary 3-D systems used (rigid frames tied together, core, tube, etc), primary lateral 2-D systems used (rigid frames of steel or concrete, braced frames of steel, etc.), floor system type (beam and slab, waffle slab, etc.), and the foundation system type (piles, caissons, etc.). The matching process in STRUPLE is conducted by defining three types of matching criteria: required criteria (such as same design live load), desired criteria (such as same state), and no-match criteria (such as wind load if number of stories is less that 5). After a set of similar past cases are identified, they are ranked according to how well they match the current set of design specifications. The systems and subsystems used in the set of buildings having a high ranking are what STRUPLE returns for use in a design synthesis routine. Each system or subsystem is given its own ranking depending on how many times it appeared in the matching past cases and the ranking of that past case. Hence, a collection of feasible design systems and subsystems along with a ranking indicating their preference is supplied to a design synthesis routine for use in synthesizing preliminary designs for a structure. STRUPLE illustrates a very effective use of case-base, or analogical, reasoning.

4.4.Inductive Machine Learning

Learning is a generalized term denoting the way in which people (and computers) increase their knowledge and improve their skills. Machine learning has been a goal of AI researchers since beginning of AI, where researchers strived to understand the process of learning, and to create computers that can be taught rather than programmed.

There are essentially four major forms of machine learning as described in [5]: inductive learning, where concepts or rules are learned from both positive and negative examples of that concept; analytic learning, which makes use of a fairly rich underlying structure of the domain and goes by the name of analogic or cased-based learning (described briefly in the previous section; genetic algorithms, which bases its strategies on mutation-based concepts; and connectionist learning, i.e., neural network learning algorithms which are discussed separately in the next section. Within this section, only inductive learning is briefly discussed.

4.4.1.Induction

One of the types of machine learning described above, induction, has received a lot of attention and presents valuable solutions for the third and fourth limitations of current knowledge-based systems. Induction is the process of learning how to perform a task by being presented with examples of how it should behave. One of the key ideas that separates inductive learning from connectionism, or neural networks, is that the knowledge obtained through induction is explicitly represented, modifiable, and explainable. This is not the case with neural networks, which will be discussed in section 4.5.

As an example of induction, consider trying to teach the system how to classify a poker hand as being a FLUSH (this example is summarized from [7]). The objective is to generate a rule (in the rule space), that correctly classifies all examples of a FLUSH (in the instance space).

- The instance space is the space of all possible 5 card poker hands, with an element of that space looking like:

  ((2 clubs) (3 clubs) (5 clubs) (jack clubs) (king clubs))

- The rule space is the space of all predicate calculus expressions that could possible be produced from the following primitives:
  - o the variables c1, c2, c3, c4 and c5, which represent the five cards in the poker hand;
  - o predicates suit and rank, which identify or test the suit and rank of cards c1-c5;
  - o any necessary free variables;
  - o the possible suit values: clubs, diamonds, hearts, and spades;

- the possible rank values: ace, king, queen, jack, and 10–2;
- the conjunction operator (AND); and
- the existential operator (THERE EXISTS).

- Hence, a predicate calculus expression that describes the concept of a flush would look like:

```
THERE EXISTS c1, c2, c3, c4, c5 :     SUIT(c1, x) AND SUIT(c2, x) AND
                                      SUIT(c3, x) AND SUIT(c4, x) AND
                                      SUIT(c5, x).
```

An inductive learning program must search these two spaces as follows:

- the program selects a training instance from the instance space and asks whether it is an instance of the desired concept (is it a flush?);
- the program interprets this instance and its classification into a form that can be used to guide the search of the existing set of rules in the rule-base;
- using this instance, a set of plausible rules (called candidates) is generated from the available primitives and existing rules in the rule-base;
- these plausible candidates are then tested against other examples of flushes in the instance space;
- those plausible candidates that work remain in the rule-base and those that do not are removed from the rule-base.

Note that this two space model of learning assumes a closed world. In other words, the instance space includes all possible instances and the rule space is capable of describing all possible rules with the available primitives. This closed world assumption allows the learning programs to locate the desired rule by progressively excluding rules that are known to be incorrect.

The most common inference process used for learning from examples is *generalization*. We say that rule A is more general than rule B, if rule A applies in all situations that rule B does and then some more. For example, "5 cards that are all of the same suit is a flush" is more general than "5 cards that are all hearts is a flush." Often description A is more general than description B because description A places fewer constraints on any relevant situations for which the rule applies. The representation chosen for the rule space is important and must be able to support generalization. For example, predicate calculus supports generalization by permitting the following: 1) turning constants into variables, 2) dropping conditions, and 3) adding conditions, to name a few of the operations.

For example, suppose we have the following two rules in our rule-space (again, example is summarized from [7]) :

Rule 1:
SUIT(c1, spades) AND SUIT(c2, spades) AND
SUIT(c3, spades) AND SUIT(c4, spades) AND
SUIT(c5, spades) = > FLUSH(c1, c2, c3, c4, c5)

Rule 2:
SUIT(c1, clubs) AND SUIT(c2, clubs) AND
SUIT(c3, clubs) AND SUIT(c4, clubs) AND
SUIT(c5, clubs) = > FLUSH(c1, c2, c3, c4, c5)

From these two, we could generalize by replacing constant values with a variable, x.

Rule 3:
SUIT(c1, x) AND SUIT(c2, x) AND
SUIT(c3, x) AND SUIT(c4, x) AND
SUIT(c5, x) = > FLUSH(c1, c2, c3, c4, c5)

Rule 3 is thus more general than either rule 1 or rule 2. To illustrate dropping conditions, suppose we have the following rule in our rule-space:

Rule 1:
SUIT(c1, spades) AND RANK(c1, 3) AND
SUIT(c2, spades) AND RANK(c2, 5) AND
SUIT(c3, spades) AND RANK(c3, 7) AND
SUIT(c4, spades) AND RANK(c4, 10) AND
SUIT(c5, spades) AND RANK(c5, king) = > FLUSH(c1, c2, c3, c4, c5)

By dropping the RANK constraints on the cards, a more general rule can be created.

Rule 2:
SUIT(c1, spades) AND
SUIT(c2, spades) AND
SUIT(c3, spades) AND
SUIT(c4, spades) AND
SUIT(c5, spades) = > FLUSH(c1, c2, c3, c4, c5)

Rule 2 is more general than rule 1, which can still be further generalized by replacing the spades with a variable x. To illustrate adding conditions, suppose we have the following rules:

Rule 1:	RANK(c1, jack) = > FACE(c1)
Rule 2:	RANK(c1, queen) = > FACE(c1)
Rule 3:	RANK(c1, king) = > FACE(c1)

From these, we could generate a more general rule for describing a face card:

Rule 4:
RANK(c1, jack) OR RANK(c1, queen) OR RANK(c1, king) = > FACE(c1)

Thus, the way in which the plausible candidates are generated is by looking at existing rules and existing instances, which look like very specific rules, and generalizing this set of "rules" into a more general rule either by changing constants to variables, adding constraints, or removing constraints. There exist much more complicated mechanisms for more efficiently performing this generalization and issues such as representation of instances and rules influence the efficacy of this process, but the basic objective remains the same — to build a general set of rules for classifying a concept from a set of concept instances and is the heart of inductive machine learning. There are other mechanisms of machine learning other than induction, but their description is beyond the scope and objective of this paper.

There are currently several problems with induction systems. Some of the rules induced are incomprehensible. A knowledge engineer still has a very big role in inductive learning by instructing the system as to what attributes are important and what attributes are not. Inductive systems will tend to over-generalize. Inductive systems require that the attributes selected by the knowledge engineer for inductive reasoning be independent of each other. In other words, learning systems cannot decide for themselves what attributes are important and which are not. If the attributes are not filtered by a knowledge engineer, the noise might be so great (in the form of unusable, or silly unrealistic rules) that the program will never perform efficiently and at an expert level. The rules induced from a set of cases is very dependent on the scope of the test cases. If the test cases do not represent a broad set of situations, the rules will be incomplete and possibly over-generalized. As is also true with neural networks (described in section 4.5.), the programming of a learning system lies in the selection of test cases or the breadth of the experiences fed into the learning systems. Hence, inductive learning systems will only work for domains where a large body of experience exists, and will not learn any principles not embodied in the experiences presented to them.

4.4.2. Applications of Inductive Learning

The concept of machine learning is quite enticing, in that one can develop a set of classification rules from a corpus of example classifications, rather than through the arduous task of manual knowledge acquisition between a knowledge engineer and the expert. Several civil engineering researchers are investigating the usage of machine learning techniques within the civil engineering domain. Reich and Fenves are investigating automated concept formation by using a set of historical bridge designs from the Pittsburgh Area and trying the derive rules for classifying a given design or for completing a design given the specification using the general rules derived from the historical set of bridge design data [12]. Arciszewski has been investigating the use of machine learning techniques for 1) developing shallow models from a corpus of stimulus-response examples that can be used to quickly reason about a domain when such reasoning is necessary such as at the preliminary stages of design, and 2) to provide learning capabilities to existing knowledge-based expert systems such that these systems can improve their behavior, i.e., learn better rules, as they experience more problems [2].

4.5.Neural Networks

Neural networks, a computation and knowledge-representation paradigm, have been inspired by the neuronal architecture and operation of the brain. Rumelhart, et al., argue that the reason the brain is capable of high performance in natural information processing tasks, such as perception, language understanding, motor control, etc., is that:

- these tasks require the simultaneous consideration of a large number of pieces of information and constraints, and
- the neuronal architecture, consisting of billions of neurons that are interconnected with a fan-in and fan-out on the order of about 1,000 to 100,000, is ideally suited for the relaxation-type computation necessary for the simultaneous consideration of the above mentioned multitude of information and constraints [41].

Thus, neural networks are ideally suited for solving certain types of engineering problems that require a mapping (i.e., pattern association) from a large, distributed set of input features to a large-distributed set of output features. Many problems in which the input represents some form of an analog signal (e.g., displacement, sound frequency, etc.) and must be mapped to some meaningful quantity representing the meaning of that signal are what neural networks are best suited to do. For example, neural networks are being used to interpret the analog signals emanating from sophisticated sensing equipment, such as computerized engine diagnostic systems[30].

This neurally inspired computation and knowledge representation paradigm consists of a model of an individual neuron, called a processing unit, that is instantiated thousands of times, with many connections existing between these various instances. The basic model of an individual neuron was developed in 1943, by Warren McCulloch and Walter Pitts and still remains at the heart of most neural networks [31]. Given that the physiological structure of a neuron looks like that shown at the top of Fig. 2., McCulloch and Pitts put forth the following model of a neuron (see bottom of Fig. 2.):

- each neuron has an activity, x_i, that is the sum of inputs that arrive via weighted pathways,
- the input from a particular pathway is an incoming signal, S_i, multiplied by the weight, w_{ij}, of that pathway,
- w_{ij} represents synaptic efficacy — the amount of post-synaptic potential transmitted to soma j if a spike occurs at soma i, and
- the outgoing signal from neuron i is computed as $S_j = f(x_j)$, where $f(x_j)$ is binary threshold function of the activity, x_j, in that neuron.

The McColluch-Pitts neuron can also have a bias term, Θj, which acts as a form of threshold. In the McColluch-Pitts neuron model, the signals, S_i, are restricted to take on the values of 0 and 1, as is obvious from the output function $f(x_j)$.

Each processor in the network thus maintains only one piece of dynamic information (its current level of activation) and is capable of only a few simple computations (adding inputs, computing a new activation level, or comparing input to a threshold value). A neural network performs "computations" by propagating changes in activation (i.e., level of stimulation) between the processors, which may or may not involve feedback and relaxation. For example, Figure 3. illustrates a neural network that is capable of detecting if the two patterns being presented at its designated input units are inverses of each other. This computation is performed by propagating activation from the designated input units through the network to the designated output units. Each unit is a McColluch-Pitts unit, where a threshold, Θ, is indicated next to each unit (other than input units) and the weights for each connection are indicated by the boxed number lying on the connection.

From the example shown in Fig. 3., it can be seen that what a neural network "computes" is very much affected by the topology and strengths of the connections between its processors, which should also be obvious from the McColluch-Pitts description of activation computation (see Fig. 2.). However, the McColluch-Pitts model of a neuron assumed that the weights were static and did not address the issue of dynamic adaptation of these weights, known as "learning."

Rosenblatt built on top of the McColluch-Pitts model with the intent of giving self-organization capabilities to networks of these units through localized, dynamic weight modification [39]. His work, while highly criticized for dealing with only simple two-layered networks which were incapable of learning certain types of functions, did pave the way for more modern research into learning mechanisms for multi-layer networks, such as backpropagation [41]. Without such capabilities, any complex computation by a McColluch-Pitts style neural network would require the manual setting of all connection strengths, which would be impossible. Hence, a key characteristic of neural networks is their capability of self-organization or "learning". Unlike traditional sequential programming techniques, neural networks are trained with examples of the concepts to capture. The network then internally organizes itself to be able to reconstruct the presented examples. Several other interesting and valuable characteristics are: 1) their ability to produce correct, or nearly correct, responses when presented with partially incorrect or incomplete stimuli; and 2) their ability to perform a limited amount of generalization from the cases on which they are trained. Both of these latter characteristics stem from the fact that a neural network, through self-organization, develops an internal set of features that it uses to classify the stimuli presented to it and return the expected response.

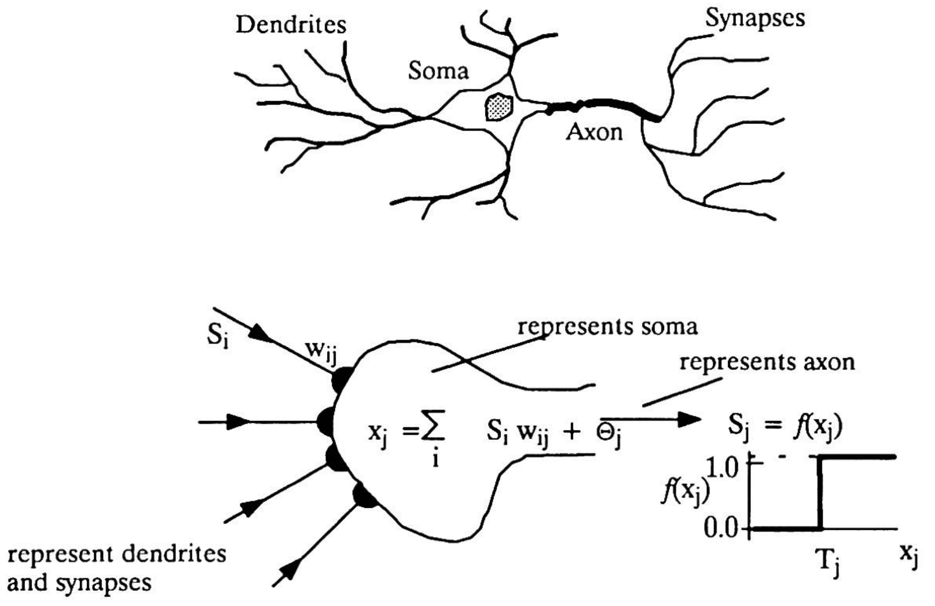


Figure 2. — The McCulloch-Pitts Model of a Neuron

In the early stages of learning, the immature connection strengths cause the actual response of the network to be quite different from that expected. Error is defined as a measure of the difference between the computed output pattern and the expected output pattern. Most learning mechanisms strive to reduce this error for all associations (input/output pattern pairs) to be learned through an iterative, localized weight modification strategy that serves to reduce the sum of the squares of errors for each pattern pair.

Rumelhart, et al. [41], provide an excellent description of the basic anatomy of neural networks, which they divide into seven basic aspects: 1) a set of processing units, 2) the state of activation of a processing unit, 3) the function used to compute output of a processing unit, 4) the pattern of connectivity among the processing units, 5) the rule of propagation employed, 6) the activation function employed, and 7) the rule of learning employed. The network topology (i.e., the number of nodes and their connectivity), and the form of the rules and functions are all variables in a neural network and lead to a wide variety of network types. Some of the well known types of neural network described in [41] are: Competitive Learning ([19], [40]), the Boltzmann Machine [20], the Hopfield Network [22], the Kohonen network [24], and the Backpropagation network [41]. The Backpropagation network is given its name due to the way that it learns - by backpropagating the errors seen at the output nodes. Although there are many other variations of neural networks, backpropagation networks are currently being considered by most neural network application developers.

4.5.1. Self-organization and the Backpropagation Rule of Learning

Several mechanisms for imparting self-organization to these multi-layer networks have been developed and are described in [41]. One of the main characteristics of these learning mechanisms is whether learning occurs in a *supervised* or *unsupervised* fashion. Supervised learning means that the expected output is included in what the network is to learn. Unsupervised learning means that the network is not told what it is to learn about the input with which it is presented and must, on its own, discover regularities and similarities among the input patterns. One form of supervised learning, developed by Rumelhart, Hinton, and Williams, is called the *generalized delta rule* [41] and is the learning mechanism used in backpropagation neural networks. All backpropagation networks, which use the generalized delta rule for self-organization and derive their name from the need to backpropagate error, have the same general architecture shown in Fig. 3..

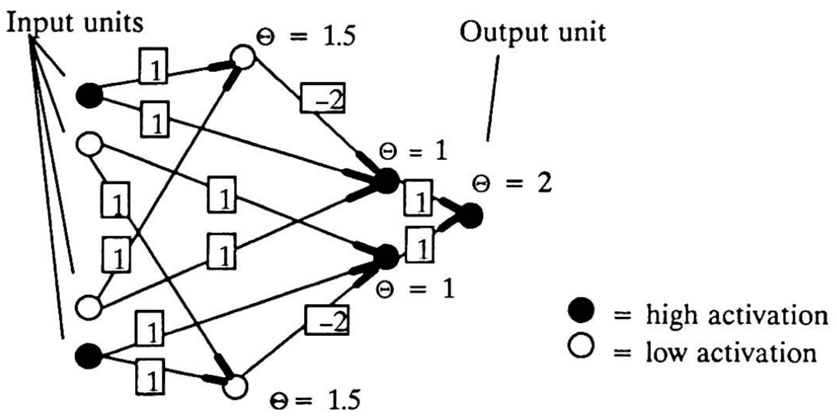


Figure 3. — Neural Network for Inverse Detection

The training (i.e., supervised learning) of a multi-layer backpropagation neural network, via the generalized delta rule, is an iterative process. Each step involves the determination of error associated with each unit and then the modification of weights on the connections coming into that unit. Each presentation of one training case and subsequent modification of connection strengths is called a *cycle*. Each cycle actually involves three sub-steps: 1) for the training case to be learned, the network is presented with the input pattern and then propagates the activation through to the processing units; 2) the error at the output units is then backpropagated back to the hidden processing units; and 3) connections coming into the hidden units then modify their connection strengths using this backpropagated error.

A set of cycles, made up of one cycle for each training case, is called an *epoch*. The training process for a network may require several hundreds or thousands of epoches for all of the training cases to be "learned" within a specified error tolerance. The generalized delta rule basically performs a gradient-descent on the error space consisting of Euclidean norms of the errors of all patterns presented during training. Convergence is not guaranteed. Convergence depends in some ways on the network architecture, its capacity and the amount of informations it is to learn. This relationship is as yet not well understood and is the subject of current research.

4.5.2. Applications of Neural Networks

Neural networks are particularly well suited to problems in which the input patterns from which inference must be made is either spatially or temporally distributed, such as the signals from some form of sensor. For example, Sejnowski and Gorman had great success in training a neural network to classify the origin of sonar signals [18]. The field of civil engineering has many problems in which the learning capabilities and the distributed problem-solving behavior of a neural network can be productively applied. Several such areas that are currently under investigation are: 1) the modeling of complex materials, where the neural network is used to discern the material relationships present within a corpus of material test data [17]; and 2) the classification of randomly generated groundwater transmissivity fields for use in identifying difficult design conditions and thus efficiently designing a reliable remediation strategy [15]. Other fruitful areas deserving of further research are: 1) the interpretation of non-destructive evaluation sensor data; 2) adaptive control of civil engineering systems such as HVAC and active structural damping systems; and 3) the interpretation of satellite photogrametry data. One point that must be emphasized at this time is that neural networks are not envisioned by this author as replacing the symbolic representation and inference techniques previously mentioned. Rather, the author envisions neural networks as providing yet another tool for employment in solving certain types of problems not amenable to traditional symbolic representation and inference. If one has an acquired set of symbolically represented heuristics, or a set of algorithmic procedures, that adequately solve a problem, then a neural network approach is likely not the best approach to solving that problem. However, if the problem is of the type where a vast amount of data must be processed in a short period of time and the "experts" that do this processing have great difficulty in describing what they do, then a neural network approach may prove viable and should be investigated.

5. Summary

The purpose of this paper was three-fold: 1) to briefly look back at where we were a few years ago regarding the application of knowledge-based system techniques; 2) to observe where we are in our current application of these techniques; and 3) to briefly describe some of the newer techniques being applied, or soon to be applied, in civil engineering problem solving. Many of the systems currently being developed are still using technology that is 20 years old. However, many others are being developed that incorporate various forms of machine learning, use more descriptive, flexible models of the domain, and make much greater use of past performance for improving their performance on new problems. While knowledge-based systems are still in their "formative years", the prospect of having a system that 1) acts as a knowledgeable colleague capable of providing advice when asked, 2) grows in capability as it experiences various problems and their solutions, and 3) serves to record in much more detail the evolution of problem solutions, is still very exciting and deserving of more research attention.

6. Acknowledgements

The author would like to thank Professor Steven Fenves of Carnegie Mellon University and Dr. Ian Smith of the Ecole Polytechnique of Lausanne for reviewing this manuscript and providing valuable editorial commentary, and Professor Jorg Schneider of the Swiss Federal Institite of Technology Zurich for encouraging the development of this manuscript.

7. References

[1] *Proceedings of the IABSE Colloquium on Expert Systems in Civil Engineering*, IABSE, Bergamo, Italy, 1989.

[2] ARCISZEWSKI, T., "Inductive Learning in Civil Engineering", in [1], pp 23-31.

[3] BENTO, J., B FEIJO, and P. DOWLING, "Knowledge-Based Design of Steel Portal Frames for Agricultural Buildings", in [1], pp 281-290.

[4] CALVI, M., A. PEANO and P. SALVANESCHI, "Expert System for the Diagnosis and Rehabilitation of Building Structures", in [1], pp 151-162.

[5] CARBONELL, J. G., "Introduction: Paradigms for Machine Learning", *Journal of Artificial Intelligence*, Number 40, 1989.

[6] CHARLES, S., J. KRUPPA and D. CLUZEL, "Expert Systems for Fire Vulnerability Analysis", in [1], pp 163-174.

[7] COHEN, P.R. and E. FEIGENBAUM, "Chapter D. — Learning from Examples", The Handbook of Artificial Intelligence, William Kaufmann, Inc, Los Altos, California 1982.

[8] COMERFORD, J., J. H. MARTIN, D. I. BLOCKLEY and J. P. DAVIS, "On Aids to Interpretation in Monitoring Civil Engineering Systems", in [1], pp 219-229.

[9] DAUBE, F and HAYES-ROTH, B., "FIRST: A Case-Based Redesign System in the BB1 Blackboard Architecture," *Proceedings of the AAAI-88 Case-Based Reasoning Workshop*, 1988.

[10] FEIJO, B., P. DOWLING and D. L. SMITH, "Incorporation of Steel Design Codes into Design Automation Systems", in [1], pp 393-402.

[11] FENVES, S. J., M. L. MAHER, and D. SRIRAM, "Knowledge-Based Expert Systems in Civil Engineering", *IABSE Periodica*, Number 4, 1985, pp 63-72.

[12] FENVES, S. J., "Expert Systems: Expectations versus Realities", in [1], pp 1-19.

[13] FURUTA, H. and N. SHIRAISHI, "An Expert System for Damage Assessment of Bridge Structures using Fuzzy Production Rules", in [1], pp 197-206.

[14] GARRETT, JR., J. H., "Object-Oriented Representation of Design Standards", in [1], pp 373-382.

[15] GARRETT, JR., J. H., GHABOUSSI, J., WU, X., and RANJITHAN, R., "Applications of Neural Networks to Civil Engineering", to appear as a chapter in the monograph entitled *Expert Systems in Civil Engineering: Knowledge Representation*, R. Allen (ed.), American Society of Civil Engineers, 1990.

[16] GEHRI, M., "Development of an Advisory System for Site Managers", in [1], pp 117-128.

[17] GHABOUSSI, J., GARRETT, JR., J. H., WU, X., "Knowledge-Baserd Modeling of Material Behavior with Neural Networks", submitted for publication in the *Journal of Engineering Mechanics*, 1990.

[18] GORMAN, R. P. and SEJNOWSKI, T., J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, Vol. 1, pp. 75-89, 1988.

[19] GROSSBERG, S., "Adaptive pattern classification and universal recoding: Part 1. Parallel development and coding of natural features detectors." *Biological Cybernetics* 23, 121-134, 1976.

[20] HINTON, G. E., SEJNOWSKI, T. J., AND ACKLEY, D. H., "Boltzmann Machines: Constraint satisfaction networks that learn." *Tech. Rep. No. CMU-CS-84-119*, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1984.

[21] HARANDI, M. T. and LANGE, R., "Model-Based Knowledge Acquisition", Chapter 4 in *Knowledge Engineering – Vol 1. Fundamentals*, H. Adeli (ed.), pp. 103-129, McGraw-Hill, New York, 1990.

[22] HOPFIELD, J. J., "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences*, USA, 79, 2554-2558, 1982.

[23] HOWARD, H. C., WANG, J., DAUBE, F., AND RAFIQ, T., "Applying Design-Dependent Knowledge in Structural Engineering Design," (AI EDAM), Vol. 3, No. 2, pp. 111-123, 1989.

[24] KOHONEN, T., *Self-organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

[25] KOLODNER, J. L., "Extending Problem Solving Capabilities Through Case-Based Inference," in Proceedings of the Case-Based Reasoning Workshop, sponsored by DARPA, May, 1988.

[26] LEVITT, R. E., "HOWSAFE: A Microcomputer-based Expert System to Evaluate the Safety of a Construction Firm", in *Proceedings of the ASCE Conference on Expert Systems in Civil Engineering*, 1986.

[27] MAHER, M. L. and S. J. FENVES, *HI-RISE: An Expert System for Preliminary Design of High Rise Buildings*, Technical Report, Department of Civil Engineering, Carnegie Mellon University, 1984.

[28] MAHER, M. L., ed., *Expert Systems for Civil Engineers: Technology and Application*, ASCE, 1987.

[29] MAHER, M. L., "Synthesis of Structural Systems", in [1], pp 261-270.

[30] MARKO, K. A., JAMES, J., DOSDALL, J., MURPHY, J., "Automotive Control System Diagnostics using Neural Nets for Rapid Pattern Classification of Large Data Sets," in Proceedings of the International Joint Conference on Neural Networks, Vol. II, pp. 11-16, Washington, D.C., June 18-22, 1989.

[31] MCCULLOCH, W. S., AND PITTS, W., "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics* 5, 115-133, 1943.

[32] MULLARKEY, P. W., S. J. FENVES and D. A. SANGREY, *CONE: An Expert System for Interpretation of Geotechnical Characterization Data from Cone Penetrometers*, Technical Report R-85-147, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA, 1985.

[33] PAGNONI, T, C. GAVARINI and Z. TAZIR, "AMADEUS: a KBS for the Assessment of Earthquake Damaged Buildings", in [1], pp 141-150 .

[34] PAU, L. F. and S. S. NIELSEN, "Architectural Design for Energy Savings and Thermal Comfort", in [1], pp 33–46.

[35] POYET, P. and B. DELCAMBRE, "Noe: Expert System for Technical Inspection of Waterproofing on Flat Roofs", in [1], pp 175–188.

[36] RAFIQ, T., "Similarity in Structural Component Case Bases," unpublished Engineer's Degree Thesis, Department of Civil Engineering, Stanford University, 1989.

[37] REHAK, D. R. and J. W. BAUGH, "Alternative Programming Techniques for Finite Element Program Development", in [1], pp 363-372.

[38] REINHARDT, H. W. and M. SOHNI, "Expert System for Repair of Concrete Structures", in [1], pp 189-196.

[39] ROSENBLATT, F., *Principles of Neurodynamics*, Spartan, New York, 1962.

[40] RUMELHART, D. E., AND ZIPSER, D., "Feature Discovery by Competitive Learning." *Cognitive Science*, 9, 75-112, 1985.

[41] RUMELHART, D. E., MCCLELLAND, J. L., and the PDP Research Group, *Parallel Distributed Processing - Volume 1: Foundations*, MIT Press, Cambridge, MA, 1986.

[42] SCHMITT, G., "Architectural Pre-Processor for Engineering Expert Systems", in [1], pp 291-302.

[43] SHANK, R. C., "Reminding and Memory", Chapter 2 in *Dynamic Memory - A Theory of Reminding and Learning in Computers and People*, Cambridge University Press, 1982.

[44] SHARPE, R. and B. MARKSJO, "Expert Systems for Engineering Codes", in [1], pp 383-392.

[45] SMITH, I. F. C., "Knowledge-Based Systems for Structures in Service", in [1], pp 129–140.

[46] STEIMER, S. F. and B. W. R. Forde, "Knowledge-Based Finite Element Analysis", in [1], pp 355–362.

[47] THOMPSON, J. B. and S. C-Y. LU, "Design Evolution Management: A Methodology for Representing and Utilizing Design Rationale," part of an unpublished annual report from the Knowledge-Based Engineering Systems Research Laboratory, Department of Mechanical Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[48] ZAGAR, Z., "Knowledge-Based System for Automatic Design of Glued Laminated Structures", in [1], pp 239–248.

[49] ZHAO, F AND MAHER, M. L., "Using Analogical Reasoning to Design Buildings," *Engineering with Computers*, Vol. 4, pp. 107–119, 1988.

[50] ZOZAYA-GOROSTIZA, C. and C. HENDRICKSON, *An Expert System for Traffic Signal Setting Assistance*, Journal of Transportation Engineering, Number 2, Volume 113, 1987.