Spring 1992

Reasoning by Analogy in a Multi-Level System Architecture for the Design of Mechanisms

Ghassan F. Issa
Old Dominion University

REASONING BY ANALOGY IN A MULTI-LEVEL SYSTEM ARCHITECTURE FOR THE DESIGN OF MECHANISMS

by

Ghassan F. Issa
B.E.T. August 1983, University of Toledo, Ohio
B.S.E.E. November 1984, Tri-State University, Indiana
M.S. December 1987, Old Dominion University, Virginia

A Dissertation submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May, 1992

Approved by:

Dr. Stewart Shen (Director)

Dr. Meng-Sang Chew

Dr. Michael Overstreet

Dr. Christian Wild

Dr. Nageswara Rao

Table of Contents

List of Tables

List of Figures

To my parents, Fareed and Samiha Issa

To My Wife Wafa

To my daughter Safa

and
To my brothers and sisters

Acknowledgments

I express my deep gratitude and appreciation to my advisors, Dr. Shen and Dr. Chew for their continuous support and guidance during the completion of this Dissertation. I would like also to thank the committee members, Dr. Wild, Dr. Overstreet, and Dr. Rao for their help and assistance, and for taking the time to review this work.

I am also grateful to the partial support received from the National Science Foundation through Grant No. MSM-8809198.

ABSTRACT

REASONING BY ANALOGY IN A MULTI-LEVEL SYSTEM ARCHITECTURE FOR THE DESIGN OF MECHANISMS

Ghassan F. Issa

Computer Science Department
Old Dominion University, 1992

Directors: Dr. Stewart Shen and Dr. Meng-Sang Chew

Since the first attempts to integrate AI technology and engineering design nearly two decades ago, few expert systems have been shown to demonstrate sufficient reasoning capabilities to solve real-world design problems. The complex nature of design, the lack of understanding of the design process, and the limitations of current expert system technology have all been shown to have adverse effects on the maturity of this research area. Therefore, our direction in this research concentrates on understanding the design process, investigating a novel area of research focusing on creative design, and incorporating the results into a system model feasible for production use. The model presented is based on the concept of reusing past experience and existing cases to solve future design problems in different application domains. The resulting system performs its task by reasoning and learning by ANALOGY while utilizing the Logical-Building Block approach to design. Our method demonstrates the use of a *case-based reasoner* in conjunction with other existing techniques, such as heuristic reasoning and first principle reasoning, to produce a system with three levels of reasoning strategies. Such a system will exhibit a learning capability by which its performance is enhanced with repeated use. A prototype has been implemented and tested for the synthesis of various mechanisms.

CHAPTER ONE

Introduction

Highlights

- The Progress in Expert System Technology
- Background in Engineering Design
- AI and the Re-use of Past Experience

Introduction

Since the development of the first so called expert system "DENDRAL" [52] in 1965, expert system technology has shown considerable success. The interaction between expert systems and other disciplines became a major issue in research and practice, and by the end of the 1970s, this interaction resulted in a wide variety of systems in many domains of expertise including chemical (DENDRAL), education (SOPHIE) [7], medical (MYCIN) [72], INTERNIST [63], speech understanding (HEARSAY) [24], and Math and Science (MACSYMA) [4]. However, it was not until the early and mid 1980's and after the success of an operational expert system called PROSPECTOR [21] that expert systems gained commercial value and were considered a major research topic eligible for funding and support. With such a success, expert systems began to operate in more challenging and widely varying domains such as business, defense, manufacturing, and engineering. A new generation of specialized software called expert system shells became available as major tools for building expert systems [54].

One of the most interesting and challenging domains of expertise that has been part of the integration efforts is the Engineering Design. This domain is clearly interesting because of its importance and effects on our every day life. At the same time, it is challenging because of the complexity and difficulty involved in understanding and automating this process.

The automation of engineering design in the light of AI technology dates to the late seventies [56,75,76], and since then has been given serious attention by researchers in both industry and academic institutions. Several universities have realized the significance of integrating AI and engineering design, and as a result have formed special inter-disciplinary groups and laboratories to pursue the design research. These institutions include Carnegie Mellon, Massachusetts, MIT, Ohio State, Rutgers, and others [65].

Despite the increased attention towards this research topic, few design-expert systems have gained popularity when compared to those found in other domains of expertise. Examples of some of the well-known systems in engineering design include: HI-RISE, a system for the design of high-rise buildings developed at Carnegie Mellon [53]; SACON, a system to aid less experienced engineers in structural analysis [5]; VEXED, a system to automate the reuse of circuit design plans, developed at Rutgers University [59]; and ARGO, a domain independent expert system developed at MCC for the automation of design based on analogical reasoning [43].

The limitation of the development of design-expert systems can be a result of two inter-related problems. First is the complex nature of design and the diversity of knowledge and data involved. Such complexity requires much of human intelligence, creativity, common sense, intuition, experience, and competence in heuristics and large computations. The second problem results from the lack of understanding of the design process and the methods and means by which designers perform their tasks.

In overcoming these difficulties, researchers agree on the necessity of first understanding the design process and then using the results as the basis for constructing design-expert systems. Indeed, this suggests that understanding the engineering design and its needs will lead the AI researchers to adopt both better and alternative methods more suitable for solving the complex design problems. A number of attempts have been cited which are focused on understanding the design theory, classifying the design into several categories, studying the behavior of the human designer (cognitive model), and demonstrating the potentials for automating the design stages [2,6,19,27,30,37,42,58,66,77]. Once the design process is reasonably understood, the next step is to incorporate AI techniques in the automation process. To do so, research in AI is being focused towards the areas of knowledge representation and reasoning strategies. The implementation of design-expert systems requires far more diverse methods of knowledge representation than that provided by the well known rule-based systems. Due to the nature of design, alternative knowledge representation paradigms may have to be utilized. This could be accomplished by combining of one or more classical paradigms such as frames, objects, or semantic nets, as suggested by Pierick [61], or it could be accomplished by developing specialized and application oriented methods suitable for the representation of highly complex design objects.

While developing a proper knowledge representation for design constitutes part of the solution to the automation of design, another and yet more challenging problem is to equip expert systems with reasoning capabilities comparable to that of a human designer. The solution to this problem relies both on understanding the design process and on understanding the behavior of the human designer as well. Recently, new research has been directed at issues involved with this problem, and includes studies in psychology, cognitive science, and AI. This research aims at developing cognitive models for design [2,30].

A major result of such studies on human reasoning, and the bulk of our work, centers around the concept of reusing past experience and existing design cases to solve new problems. The significance of this concept is that it is closely related to the nature by which a human designer tackles new problems. Most studies show that designers, when solving new problems, are reminded of similar cases that have been solved in the past. Once an existing case is found to resemble the new problem, its solution is used as an aid in constructing a new solution to the problem at hand. Once the new problem is successfully solved, its solution is stored back in memory for later use. Repeating this process over time allows humans (designers) to learn a substantial number of new design cases, and thus provides a more powerful capability to solve future problems. It is clear that such activity is not restricted to design only, but it plays a major role in the human's overall reasoning capabilities. A human's memory is constantly working, and people are always reminded of previous work with regard to understanding and solving new problems.

While the reuse of past experiences is not a new concept in disciplines such as psychology and cognitive science, it has just began to gain momentum in the AI field.

Many of the AI concepts that have been established for several years constitute the basis for developing such a model in design. *Planning* for example, is an activity which has received considerable attention from AI researchers, is considered an essential tool for engineering design. The design of an artifact could be organized in the form of a plan. This plan can be later generalized to suit the design of many other objects. AI researchers such as Fikes, Hart, and Nilsson developed a system called STRIPS in 1971, which uses planning as a problem solving method and applies it to the automatic theorem proving [28]. In 1972, Fikes and Nilsson extended their work on STRIPS by introducing plan generalization and learning, then applied the system to the execution of robot plans [29]. Currently, planning has been a major issue in research both in AI and engineering design.

A more interesting direction in AI that specifically deals with the concept of reusing past experience is *analogical reasoning*. The ability of human beings to use their past experience to guide them in solving new problems through analogical reasoning has been considered a fascinating problem in human reasoning studies and AI alike. Until recently very few people in AI have touched on this concept. This is quite obvious from the available research on the subject. Even the *AI Handbooks* have not contributed to the study of analogical reasoning except for a small paragraph in volume III [16]. Recently, a number of researchers in AI and machine learning have been working intensively on this area, with the aid of cognitive and psychological theories. The application of analogy into AI, or *computational analogy*, had been attempted in the earlier years of AI. Unfortunately, it was considered a difficult and impractical subject, and has been ignored until recently.

Studies in computational analogy (computer models of analogical reasoning) actually began as early as 1968, by Evans [25], who developed an early system called ANALOGY, which reasons by analogy to find relations among geometric shapes. Although the system incurred several limitations, it was considered a pioneering work in this research area, and gave other researchers the opportunity to further pursue such a topic. Shortly after Evans's work and in 1971, Kling developed a system called ZORBA [48], which reasons by analogy as an automatic theorem prover. This system uses the previous solutions of theorems to solve new ones. This work was considered the first of its kind to use the analogy concept in theorem proving. Following this pioneering work by Evans and Kling, it was not until the early 1980s that other researchers began to pursue this research area [11]. With the advent of research on Machine Learning, work on analogy became an important topic in AI. Important work on analogy was beginning to surface by many researchers such as the work by Carbonell in 1983 on Transformational Analogy and in 1986 on derivational analogy [10,12]. Other work on this subject has been accomplished by Burstein 1986 [9], who discussed the concept formation by incremental analogy. In 1988, Greiner [38] developed a system called NLAG which uses a set of heuristics for the selection of useful analogy based on abstraction and preference. In 1989, Hall [40] produced a comparative analysis of the computational analogical reasoning. His work includes a large survey of previous work on analogy. In 1986, Eliot [23] discussed the advantages of using analogical reasoning in the development of expert systems. Other work on analogical reasoning can be found in [15,18,71].

With the increased attention towards analogical reasoning and the reuse of past experience, a new research direction called *Case-Based Reasoning (CBR)* was born in the late eighties. Case-based reasoning is considered a general framework for the reuse of past experience that uses analogy, planning, learning, and other techniques, while at the same time focuses on memory organization and retrieval. Workshops on case-based reasoning are being sponsored by DARPA (May 1988 and 1989), in which a number of papers were presented on this topic. In 1987, Kolodner [51] presented some background in the concept of using a case-based reasoner to extend the capabilities of problem solving. She uses a system called JULIA which operates in the meal planning domain as an example to illustrate the case-based inference. Hammond in 1988 [41], presented a framework for a case-based planner in which he suggests six basic processes required in case-based planning. Ashley and Rissland, 1988 [3], demonstrated some techniques employed in a system called HYPO for representing and applying knowledge in the form of real and hypothetical cases to assist attorneys in solving new legal cases. Navinchandra, 1988 [60], presented a system called CYCLOPS, a design problem solver based on the concept of case-based reasoning. This system works in the domain of landscape design and uses previous cases to solve new design problems.

Although the reuse of past cases through analogical reasoning constitutes the major part of our model, two other problem solving techniques are used in the system. A *Heuristic Reasoner* (HR) is located at the bottom of the CBR, and consists of design and evaluation rules that are based on designers' experience and heuristics in solving several design applications. Following the HR, and at a lower level, is a *First Principle Reasoner* containing fundamental and detailed knowledge of engineering design. The knowledge base of this reasoner consists of information regarding a number of design components which constitute the basic building blocks of many mechanical designs.

First Principle Reasoning has been used recently in only a few AI systems. Very little work have been cited in this area, and most systems using this reasoning have been directed toward diagnosis rather than design. We feel that first principle reasoning could be as useful in design as it is in diagnosis. There has been some work done on this area in MIT [39] focused towards digital design. In 1987, Reiter [64] did some work on the theory of diagnosis from first principles. Fink and Lusth in 1987 [31], presented a system called IDM, a diagnosis system with two types of knowledge based both on experience and diagnosis functions. Chandrasekaran and Mittal talk about the difference between deep and compiled knowledge for problem solving. They present a system called MDX for medical diagnosis based on the deep knowledge structures [13]. The IEEE Expert Journal (May 1991) has featured a number of short articles pertaining to the Compilation issue of First Principle Knowledge.

Adding a heuristic reasoner and a first principle reasoner as an aid to the analogical process in the CBR, would result in a much more powerful system overall. These two reasoners play a major role in the overall problem solving process and promise to overcome many of the difficulties associated with the analogical reasoning observed in most of the cited models. Using case-based reasoning or analogical reasoning does not mean that a system will always be successful in solving all kinds of different problems. The simplest case to illustrate this problem is when the system operates for the first time with no prior knowledge existing in its CBR. Another example would be in the case where the CBR cannot retrieve any analogous problem to that in hand. Furthermore, there is always a possibility of retrieving a certain case that can only aid in constructing a partial solution to the current problem. Finally, there could be a need to access these reasoners to verify the solution of the CBR. For such reasons, adding the HR and FPR would distinguish our systems from others using the concept of reusing past experience through analogical reasoning. It would also result in a system capable of solving a wider variety of design problems with minimum failure.

1.1 Background in Engineering Design

This section presents a brief introduction to some of the concepts and terminology regarding the engineering design process.

The definition of Design: Design is a goal-oriented decision making activity which results in the construction of an artifact that satisfies a set of constraints and requirements. Given a set of specifications and requirements, a designer's goal is to produce a formal description of a device which satisfies those requirements and which is feasible for implementation.

Classes of design: The definition of design above, provides a general description of design. However, "design" is a general term and can vary in complexity from one situation to an other. It could be simply a routine process which needs little if any creativity with a well defined set of requirements, or merely a look up from a data base that finds designs to match these requirements. It could also be a very complicated process which needs a great deal of creativity and experience. Design requirements could change with time, and designers are expected to handle such changes. Experienced designers are capable of producing acceptable designs and inventionS without a complete set of requirements or specifications, and even without a pre-defined goal. Brown and Chandrasekaran, 1985 [6], have categorized design into three classes according to the complexity and creativity involved (see section 2.1.2).

Design is, therefore, a diverse problem solving activity involving different levels of complexity and creativity. This suggests the necessity of understanding the level of complexity of a design problem and the chances of its success, before any actual attempts to automate the design process using expert systems are made.

Design Stages: The design of an artifact is an evolutionary process in which several iterations and refinements may be necessary in order to produce the final acceptable design. This process consists of several stages, each of which contributes to the final product. In general, and as suggested by Fenves in 1978 [27], the design process consists of three stages:

1. *Preliminary design,* also called *conceptual design* or *conceptual synthesis* has been defined by Ulrich and Seering [79] as the transformation of functional or behavioral requirements to structural description or physical embodiment. At this stage, the resulting description of an artifact is usually a rough design with many approximations, but maintains the essential features of the intended design. The result is a set of high-level design alternatives, containing little detail, representing actual designs. These alternatives need to be evaluated and analyzed in the next stage.

2. *Analysis* and *Evaluation* stage. The analysis stage starts with a given description of the state of a design object and arrives at a description of the performance of that object. The goal of the analysis stage is, therefore, to select one or more alternatives constituting feasible or optimal designs. This is usually accomplished by applying a set of evaluation rules on each design alternative. The rules usually work by eliminating those alternatives which violate any of the functional or structural requirements.

3. *Detailed Design.* This is the final stage in the design process in which the preliminary designs undergo further refinements. Detailed information is added to the designs and final component selection is done. More detailed analysis is then performed on the resulting design which will be ready for implementation.

The work presented in this dissertation, however, deals only with automation of the first two stages of design. Preliminary designs would be created and then evaluated based mostly on qualitative reasoning. Qualitative reasoning has little regard for using numbers and lengthy calculations. It is based mainly on decision making activities and empirical knowledge.

1.2 AI and the Re-use of Past Experience

This subsection presents an introduction to a number of AI concepts and techniques that relates to our research and to the system model. It serves as an introductory section to some of the terminology used through out this dissertation.

Planning

Planning is a form of problem solving in which a course of action is decided upon before taking any action. Planning is an important activity performed by people in most everyday activities. Planning could be used to find optimal solutions for complicated problems with one or more interacting goals. While there are several approaches to planning in the AI field, two approaches have been widely used. First, *nonhierarchical planning*, in which a sequence of problem solving actions are recorded in order to solve given goals. However, nonhierarchical planning has several limitations. The plan does not distinguish between important actions and ones that are less important. The plan would also be filled with details that are not needed until later stages of the problem solving. This means that details are executed at the same time major decisions are taken, and if for any reason backtracking is needed, the system would have wasted a great deal of its time in dealing with such details. A well-known system that uses this form of planning is STRIPS.

To overcome the limitations of nonhierarchical planning, *hierarchical planning* is used in which a plan consists of different levels of details or abstractions. The top levels of a plan usually contain very general knowledge and major decisions. While lower levels contain detailed knowledge. This means that the details of a plan will be executed later in the problem solving, and the system would not have to deal with it until the more important decisions have been considered. A system that uses this form of planning is ABSTRIPS.

Machine Learning

Machine learning can be defined very generally as any process by which a system improves its performance. This could be done by acquiring and applying new methods and knowledge, or by improving existing methods and knowledge. While our system makes a use of one form of learning, a comprehensive study of machine learning is not within the scope of this dissertation. See [11,16,18] for more details. Nevertheless, we can define learning in accordance to our work as a process by which a system can acquire new knowledge through the course of its problem solving, by storing the new knowledge, and by the ability to use such knowledge to solve future problems. Chapter 4 of this dissertation relates reasoning by analogy to other Machine Learning Strategies.

Analogical Reasoning

"*Analogy pervades all our thinking, our every day speech and our trivial conclusions as well as artistic ways of expression and the highest scientific achievements.*" --Polya 1957 [62].

Reasoning by analogy provides an essential factor of human problem solving and learning. Through analogical reasoning, people are able to relate the contents of their memory to new situations that arise in the course of every day activities. Thus analogy plays a major role in the ability of humans to use their prior experience to solve new problems which bear some resemblance to that experience. Analogy is not just simply a pattern matching mechanism, it involves comparisons between concepts, causal knowledge, and deep understanding of the compared objects. Additionally, it includes the means by which knowledge about an existing episode can be transferred or adapted to the new situation in order for a solution to be derived. As for AI problem solving and learning, analogy promises to overcome the horrendous search complexity for finding solutions to novel problems and for inducing generalized knowledge from experience.

Research in analogy, however, remains in its infancy stage, and surveying the literature about this topic indicates little agreement on an exact definition of analogy. We present a definition of analogy based on some of the definitions found on this topic. Based on definitions by Carbonell [12] and Hall [40], we can define *analogy* or *analogical reasoning* as a mapping between information of a source domain (base problem) and a target domain. The mapping transfers knowledge from a previously solved episode (given the terms source or base) to a new problem (target) that shares significant aspects with a similar past experience. The result of this process is a solution to the new problem.

Case-Based Reasoning

The concept of case-based reasoning is similar to that of analogical reasoning. The idea is to use the solutions of previous cases in solving new related ones. Case-based reasoning however, places more emphasis on memory organization, indexing, and retrieving. The general concept is to treat a knowledge base as a long-term memory capable of storing and retrieving episodes very efficiently. A very well-known model for such organization has been introduced by Schank in 1982 [68], called the MOPs theory, which was later implemented by Kolodner [49-51]. According to this theory, understanding of a new case involves finding the best knowledge in memory that can be used to make predictions from it. Finding this knowledge is equivalent to integrating the new case with what is already in the memory. As reasoning is being carried on, memory is constantly probed and updated, and as a result, the case becomes more integrated and more knowledgeable for making a prediction.

Reasoning From First Principle

Reasoning from first principle involves using fundamental or deep knowledge about the artifacts being modeled. Fundamental knowledge is usually based on well established properties of objects, and on principles that have been proven to be correct. Such knowledge may involve a description of a certain object and why or how such an object operates the way it does, or it could involve basic theories and laws such as those in physics, chemistry, biology, or other domains. This type of knowledge is usually acquired by going to schools and reading from books. People use this knowledge and build on it from experience to form what is known as heuristic knowledge. When people are faced with difficult problems for which they have no experience in solving, they usually go back to the fundamental knowledge that they have acquired earlier for assistance. At the same time, experts are expected to rely on fundamental knowledge to present well formed explanations for their results.

Rule-based systems

Rule-based systems are some of the most commonly used and most practical form of expert systems that exist today. This is due to the maturity level that such systems have reached, and to the wide-spread availability of development tools. A typical rule-based system consists of a knowledge base containing mostly heuristic knowledge stored in the form of IF-THEN rules, an inference engine that controls the execution of the rules, and a certainty system to handle uncertain knowledge and uncertain results. Rule-based systems have proven to be very efficient in handling problems that are diagnostic in nature, but have failed to do the same in other forms of problem solving such as design. Many of the limitations of rule-based systems and other related systems that are based on similar principles have been criticized heavily by both researchers and practitioners. Such criticisms have resulted in many people referring to them as knowledge-base systems instead of expert systems.

CHAPTER TWO

Literature Survey

Highlights

- Understanding Engineering Design
- Related Research in AI
- Application of AI To Engineering Design
- Summary of Research

Literature Survey

Chapter Outline

This chapter presents a survey of related work pertaining to research area. The chapter begins by discussing some of the major work in the area of understanding the design process, followed by a discussion of the AI methods suitable for the automation of design based on the concept of re-using past experience. Finally, a number of systems are presented to serve as a comparison for our work.

Each review is followed by some comments and criticism which point out the advantages and disadvantages of the work presented. The chapter is concluded with an overall summary to the previous work and how that work compares to our research efforts.

2.1 Understanding Engineering Design

2.1.1 Fenves 1978 [27]

Fenves presents some early work based on the integration of AI in the preliminary design of structural components. His work aims on showing the potentials and needs of using AI technology in the automation of design, based on a formal method for describing design specifications.

Fenves begins first by describing the nature of design specifications and the formal representations and methodologies developed so far, he then concentrates on the potential of AI applications in the development and use of design specifications.

Fenves treats structural engineering design as a process consisting of three phases: a) preliminary design, in which a structural configuration is selected and preliminary component parameters are chosen; b) analysis, in which the response of the structure is evaluated; and c) detailed design, in which components and connections are selected so as to satisfy the design objectives.

One of the key issues discussed in this work is that design specifications are the primary communication tools and control mechanisms for the design and construction of structures. Nevertheless, at present, there are no organized formal methods for generating and reviewing proposed new specifications or changes in existing ones. Fenves therefore, attempts to present a formal method for representing and developing design specifications while borrowing some techniques from AI. A specification set can be viewed as a collection of *provisions*, or normative statements. A provision is a requirement statement that must be satisfied by the design object and could include a subject, predicates, operators, and so on. Provisions are then represented by decision tables to overcome the complexities of provisions conditions and properties. Hierarchical interaction between data items generated by provisions could also be represented using information networks. Finally, descriptor trees are used to organize, outline, and index provisions according to their scope and performance sought.

Such representation methods could be applicable to the following processes in the specification development:

- Formulation, the development of the information content of the specifications.
- Expression, the exposition of the information content in both conventional textual form and forms adaptable for computer processing.
- Use, the interpretation and application of the specifications to the evaluation of design.

Following the discussion on the nature of design specifications and on the methods to formalize them, Fenves utilizes an AI framework that is applicable for structural engineering in the form of a search in a problem space. The problem space consists of: knowledge elements constituting the initial state, the goal state and the intermediate states; operators for changing the knowledge states; the rules of operations; and some heuristics for the selections of states and operators. The search is then depicted as the process of selecting current states, selecting operators, applying operators, checking for the goal state.

The introduction of AI as an aid to specification formalism is viewed by Fenves from three different levels. At the *first level*, it is postulated that many AI techniques for recognizing, sorting, weighing, and ordering problems modeled by graphs and trees could be applied directly to improve ordering of the structure of specification provisions. AI techniques for instance could be used to transform a decision table into a decision tree which later could be expressed as one or more sentences containing conditional clauses. In other words, AI techniques such as traversals and searching algorithms are used to transform decision tables into linear textual format easily understood by the designers. At the *second level*, Fenves indicates how methodologies and concepts from AI and information processing psychology are applied to the understanding of engineers processing needs and limitations to the organization of specifications to correspond more closely to the problem solving process. Finally, at the *third level*, He proposed merging recent developments in the engineering fields with AI concepts to convert the current ill structured problems into well structured ones, by associating the normative information of the specifications with the designers problem space, thus providing explicit means for discrimination, evaluation, and selection among provisions.

In summary, Fenves work has been focused towards using current AI techniques to overcome many of the problems and limitations associated with understanding and writing specifications. The goal is to develop a well structured formal method for the representation of specifications. As far as reasoning and problem solving is concerned, Fenves only touched briefly on this subject by depicting the problem solving as a search space with a set of operators. This model is thus similar to the classical Means End Analysis which has been seen in many early AI systems such STRIPS and GPS.

2.1.2 D. C. Brown and B. Chandrasekaran 1985 [6]

The work presented by Brown and Chandrasekaran has two major objectives. First, they attempt to present a better understanding to the design process by organizing it into classes. Second, they present a different approach for automation of the design process. Their approach is based on design refinement to overcome the limitations imposed by the commonly used rule-based systems. Brown and Chandrasekaran categorize general design into three classes as follows:

Class 1 Design: This class of design is rarely seen in every day design activity. It is an extremely innovative behavior which usually leads to the invention and creation of completely new products. The knowledge sources and the problem-solving strategies are not known in advance.

Class 2 Design: This class may arise during routine design when a change in requirements is introduced which causes the designer to use new components and techniques. This design class involves substantial innovation and includes some planning.

For this class the knowledge sources are known in advance, but the problem solving strategies are not.

Class 3 Design: This is a most common form of design. It involves the selection from among a well understood design alternatives. The design goals and requirements are fully specified, subcomponents and functions are already known, and knowledge sources are previously identified. As simple as this class may seem, it is still considered overall complex. This is due to the large number of possible combinations of initial requirements, the numerous components and subcomponents that have to satisfy the requirements, their immediate consequences, and the consequences of other design decisions.

Brown and Chandrasekaran conclude that class 1 and class 2 design are outside the reach of the current AI technology. Instead, they focus on the automation of class 3 design and they present an approach for a design problem solver. Their approach is a top-down problem solver consisting of a hierarchy of design *specialists*. Each specialist performs the same problem solving technique, but contain different type of knowledge. Specialists at higher levels contain general information, while those at lower levels contain detailed knowledge regarding a design component. Each specialist attempts to solve parts of the design by selecting and invoking one of the available plans in its knowledge base. A plan could be a sequence of calls to other specialists or other tasks.

In summary, the key contribution of this work is the classification of design into three classes. Breaking design in into classes gives a better understanding to the complexity associated with design, and provides a good feasibility measure for the implementation of a given design problem.

The hierarchical structure of the problem solver and the use of specialists, coincides with research area on *planning and problem solving*. Recently, there have been many systems developed that are based on planning and on top-down refinement.

2.1.3 Beth Adelson 1989 [2]

Adelson focuses her study towards understanding how designers perform their design tasks. She tries to find answers to this problem by studying the cognitive model underlying design. Cognitive models describe the complex behaviors required to solve engineering problems in terms of the underlying functionality of a design problem. Adelson research is thus focused on producing a cognitive model of the design process. She illustrates her approach by presenting a model based on software design.

Adelson discusses the concept of using protocol analysis for understanding the cognitive behavior of design. Such research begins with choosing a cognitive skill to be modeled, constructing a task that will allow the skill to be observed, and then selecting a method for recording the performance of the task. Once a task is selected, protocols can be collected of subjects performing the task. Adelson presents several advantages for protocol studies. First, it allows researchers to study complex, interacting behaviors necessary for the implementation of real-world problem solving. Second, protocol analysis has the ability to yield data bearing on the questions generated by the cognitive models. Additionally, protocol studies allow researchers to observe designers behavior in a natural setting, protecting the results against distortion by the experimental environments.

To illustrate the approach of the cognitive theory development, Adelson presents an example on learning to program by analogy. Adelson claims that programming problems have features common to that of engineering problems. She presents a general theoretical framework for learning by analogy based on work done by a number of researchers such those presented in our survey. In general, the framework consists of the following steps: retrieval, mapping, evaluation, debugging, and generalization. Adelson concern however is to further specify this general framework in relation to mapping, evaluation, and debugging, to produce what she claims: purpose-constrained mapping, active evaluation, and active debugging (see Adelson 1989 for more detail).

As a task for here protocol study, Adelson constructed a tutoring model which teaches students who acquire some basic knowledge in Pascal how to use *stacks*. The intention of this model is to build on students existing knowledge of Pascal through the use of analogy. The system uses analogy between a stack and a cafeteria rack that is used for holding plates. Using an analogy between such objects, the system goal is to constraint the mapping between such objects through emphasizing on the purpose rather than surface features. The system also makes sure that internal features or mechanisms in the cafeteria rack such as the spring that supports the plates are irrelevant in such analogy.

In summary, Adelson attempts to use protocol analysis to develop a cognitive model in design. While protocol studies have several advantages they have some disadvantages which are discussed by Finger and Dixon in next section of this dissertation. Additionally, Adelson chooses a software model as a task in her protocol study. Since her research topic is to uncover how designers design, an engineering design task would have been more informative. Finally, the model did not address the retrieval issue in the analogical process.

2.1.4 Finger and Dixon 1989 [30]

Finger and Dixon have presented a number of contributions towards understanding and automating the design process. In their latest work discussed in this section, they attempt to shed some light on understanding the design theory and on how designers design. Their work can be divided into two categories: one that gathers data on how designers design and the other that builds models for the cognitive process. They have defined the cognitive model as a process that describes, simulates, or emulates the mental processes used by designers while creating designs.

Finger and Dixon study the engineering design process in terms of three models. A *descriptive model*, intended for describing and understanding the design process as performed by human designers. Secondly, a *prescriptive* model which addresses issues such as how the design process ought to be performed and how to make it more formal, and what attributes a design artifact ought to have. Finally, they present a *computer-based model* which expresses methods by which a computer may accomplish different tasks.

Related to the descriptive model, Finger and Dixon discuss a method for observing human behavior while doing tasks called *Protocol Analysis*. In such method, a designer is asked to be verbal while accomplishing a given design process. The intention is to observe and record all the steps and actions performed by the designer. However, Finger and Dixon criticize this method for a number of reasons. One criticism is that it is rather difficult to record parts of the design process which are inherently nonverbal such as geometric reasoning. Moreover, the requirement to verbalize may interfere with the design process itself. Finally, protocol studies must address the problem that even though subjects may not have any reasons to withhold information, they may do so unconsciously.

Another model related to the descriptive model discussed by Finger and Dixon in this work is the issue of building cognitive models for design. This farely new research topic is aimed on building computer-based models that describe, simulate, or emulate the skills that human use as they solve problems. A cognitive model describing the process that constitute a skill could be specified as a set of mechanisms with defined functionality. Each mechanism is described as a process which transforms classes of input into classes of output. The interaction between such mechanisms must be defined in the cognition model.

Next, Finger and Dixon discuss the prescriptive model of design. The intention of such study is to determine how the design process ought to proceed and what attributes a design artifact ought to have. Based on literature survey, Finger and Dixon present two prescriptives for the design process. First, a canonical design process which classifies design according to the creativity and efforts involved, and which presents design as an interactive progression process consisting of several stages. This model also suggests that design is strongly influenced by the properties of the designers themselves. In practice however, designers do not usually follow this model because of lack of systematicity on the part of designers.

Finger and Dixon touch on yet an other prescriptive model called *Morphological Analysis*, which deals with dividing large design problems into several subproblems. Such model however is mostly suitable for configuration problems.

As far as prescriptive models of the design artifact, two systems are discussed by Finger and Dixon. The first system is an Axiomatic Design System based on two axioms 1) Maintaining independence of functional requirements, and 2) Minimizing the transformation necessary to meet the functional requirements. The second system is called Quality Loss System, stating that a good design is one that minimize the quality loss over the life of the design, where quality loss is defined to the deviation from desired performance.

Finally, Finger and Dixon discuss computer-based models of the design process. In this model they make a distinction between computer process that design and those that analyze or evaluate. They also distinguish between three types of design problems. First, parametric design which is the process of assigning values to attributes which are called the parametric design variables. Second, configuration design, or structural design, where a physical concept is transformed into a configuration with a defined set of attributes. Third, a conceptual design, where functional requirements are transformed into physical embodiment or configuration.

In summary, the work presented by Finger and Dixon represents the future directions of the engineering design research. They have presented a well structured study of design which began with a descriptive model, a prescriptive model, and ended with a computer model.

Their work included a quite comprehensive survey of research that has been carried on in many other countries beside the United States. From that survey, they conclude that most work done in the U.S has been only focused on the descriptive and computer models of design. Accordingly, they suggest the need to catch up with many European countries that have already began an intensive work on the prescriptive models of design.

2.2 Related Research in AI

2.2.1 Kling 1971 [48]

Kling presented one of the pioneering work using analogical reasoning in the automation of theorem proving. He describes a system called ZORBA, an automatic theorem prover based on analogy. The system works by finding a proof to new theorems (target) by using the solution of an analogous one supplied manually by the user. The goal of the system is to constraint the starting clauses used in the proof of the new theorem based on the clauses used in the proof of a previously solved theorem. Such constraints will make problem solving converges faster to a solution.

The input to ZORBA consists of a target theorem to be solved T', and an analogous base theorem T with its proof P. ZORBA performs an *Initial-Map* on the two theorems based on syntactic similarities of the predicates in theorem statments. Before these predicates can be mapped, however, they have to belong to the same semantic category. Such mapping is done over only the predicates and variables of each theorem. The system then extends (EXTENDER) the mapping so that the clauses of the source proof are also mapped to the target clauses. Performing an initial-map first on the predicates and variables, then extending it to cover the clauses can reduce the search space substantially.

The problem with the above model however is that relying on syntactic similarity between predicates is rather limited. There are many situations in theorem proving where two syntactically different predicates could have the same meaning. Additionally, an assumption has been made in Zorba that syntactically similar theorems will have similar proofs, which is not always true. Finally, Zorba has no mechanism for determining and retrieving analogous source theorem, instead the theorem and its proof are supplied by the user.

2.2.2 Carbonell 1983, 1986, 1988 [10-12]

Carbonell criticizes the traditional problem solving approaches such as that of Fikes and Nilson (STRIPS) [28], as they solve problems over and over again without the benefit of using prior experience to aid in solving new problems.

Carbonell mainly discusses the classical *Means Ends Analysis* method in which an initial state is transformed to a goal state by applying a set of transformational operators. His idea is to extend this method to allow the use of previous experience. To do so, Carbonell makes a use of a *Reminding* function in which a new situation is reminded with a previously solved one. This is accomplished by computing a difference function (similarity metric) based on finding differences in:

1. The initial state of new problem (target) and initial state of previously solved one (source).
2. The final states of both the target and source problems.
3. The path constraints under which the target problem must be solved and path constraints present when the source problem was solved.
4. The proportion of operator preconditions of the source operator sequence satisfied in the target problem.

Once a similar experience is established, the next step is to transform the old solution using transformational operators called T-operators to a form suitable to the new problem. The transformational process is then viewed as a problem solving process, searching in the space of possible solutions of the new problem. The initial state is considered the source solution, while the goal state is the target solution. The duty of the T-operators is to reduce the difference between these two states and thus transforming one solution to another.

This approach however has several limitations. It does not address the issue of finding an analogous solved problem quite well. Heuristics were introduced to determine similarity among problems based on similar initial state and goal state. This is actually not a strong criteria since it is only based on surface features of problems. Hence, two problems may have similar surface features and have completely different solutions.

The model also did not address the issue of providing a mechanism for detecting the failure of an analogy process so that it could be solved by other means. Finally, there was no constraints applied on the T-operators. With no restrictions, the T-operators could result in a meaningless transformation, where one solution could be transformed to any other through a repeated application of the T-operators.

To overcome such shortcomings of the transformational analogy model, Carbonell presented an other model called *Derivational Analogy* [12]. The central idea in this new dissertation is to rely on more information regarding subject problems other than surface features. To do this the derivational analogy approach stores all the steps (plan) used in solving previous problems. This means that two problems are considered similar if much of the underlying plan for solving one problem can be used to solve the other problem.

Although this new model is more powerful than the previous one in terms of the type of knowledge used in the analogy process, it has some limitations. One major problem is that how a system using this model knows when and what derivations of previous solutions to store in memory. The system in such case has to anticipate how the solution of the current problem will be used to solve future problems; an apparently difficult task.

Another problem is that in order to know which part of an existing plan match those of a new problem, the system must at least attempt to solve the initial parts of the new problem, and then allow the analogy process to take over based on similarity with an existing problem solution. This is indeed not an easy task since the system must know when to stop and when to permit the analogy process take over.

2.2.3 Kedar-Cabelli [46,47]

Kedar-Cabelli's work is concerned with finding an approach to constraint the possible analogical mappings between a source and a target problem. Such work is intended to solve a major problem in many analogy situations in which the space of legal analogous is usually very large while only a small set of this space is actually considered useful analogous.

Kedar-Cabelli's approach is based on using the concept of *purpose* to guide the analogical mapping and to constraint the number of allowable analogous. In his model two objects are considered analogous if a causal network of relations can be mapped from one object to the other while demonstrating how both can be used for the same purpose. For example, a "magic marker" is considered analogous to a pen if it can be shown that the former could be used for writing. In other words the magic marker can have the same purpose as the pen.

Kedar-Cabelli used his approach in "concept-learning by analogy" in which, given an unfamiliar concept, one can learn about it by mapping over many aspects of a familiar situation. This is like learning the concept of the atom from the concept of the solar system.

As part of the work proposed, a unifying frame work is presented and consists of the following steps:

Given:

- goal concept,
- purpose of the goal concept,
- domain theory in the form of IF-THEN rules,
- a new target example.

Find:

- a familiar base example,
- an explanation of how the base concept is a member of the goal concept,
- an explanation of the target example is a member of the goal concept derived from the explanation of the base example.

Process:

1. Retrieve a known base example of the goal concept.
2. Explain how this example satisfies the purpose of the concept using the domain theory.
3. Map the explanation derived from the base concept to the target concept.
4. Justify that the explanatory inference of the target are satisfied.
5. Learn or refine the goal concept.

In summary, purpose-directed analogy introduces constraints that limit the analogical inference by selecting only those useful analogies to solve the problem. Nevertheless, this approach has some limitations. The model does not treat recognition and evaluation of the source problem. Instead such information is provided for the system. Additionally, the knowledge used in the domain theory could be sufficient to derive the same results of the analogical inference. This implies that from the theoretical point of view, the learning process is not valid since the results of the analogical inference could be derived from the domain theory. Finally, the model was not clear on how the learned concepts were stored or on how such concepts are to be used in the future.

2.2.4 Gentner 1985, 1988 [35,36]

Gentner addresses the issue of learning by analogy based on a theoretical framework called *structural-mapping*. The central intuition is that analogy is a mapping of knowledge from one domain (the base) into another (target) conveying that a system of relations which holds among the base objects also holds among the target objects. Central to this intuition is the principle of systematicity. This principle shows preference for mapping systems of predicates with causal relations rather than mapping isolated predicates. In other words, a model that uses such model does not consider surface features of objects as necessary for analogy. It also means that two objects need not resemble one another to be considered analogous. Instead, emphasis are on causal knowledge connected as networks.

Given a target and a base problems, *accessibility* is the process of matching knowledge in the base and target (reminding). *Mapping* on the other hand, is the process that occurs after a situation has been accessed and involves matching predicates of the base with corresponding ones of the target based on some mapping rules which has restrictions to insure systematicity and consistency. Gentner also introduces the concept of *soundness*, which says that an analogy is sound when there exists a highly systematic relational structure that can be mapped into a target domain.

In summary, the systematicity principle is a good method for constraining the type of knowledge to be mapped from a base domain to a target domain. Only connected knowledge which forms causal networks are considered for mapping. Other surface features which are unimportant are considered irrelevant in the mapping process. Nevertheless, this model has some limitations. First, the base problem is assumed given and the appropriate aspects of the base and target are put manually in correspondence. Second, the systematicity principle is neither necessary nor sufficient for constraining all types of mapping. There are situations when unwanted attributes can be prevented from being mapped by simply checking that they are inconsistent with information of the target. Conversely, the systematicity principle alone is not sufficient as a constraint on analogical mapping. There could some important causal networks in the base which are irrelevant to the target problem. Thus using the systematicity principle, a system cannot avoid such causal network. This suggests the need for a mechanism which can determine which causal network should be mapped or not.

2.2.5 Eliot 1986 [23]

The objectives of Eliot work is to incorporate analogical problem-solving into expert systems. In his work, Eliot presents some definitions associated with analogy and analogical thinking. He then, discusses the potential of automating and using analogy in expert systems.

Eliot points out three types of difficulties associated with the use of analogy. First, there are little agreement on the exact definition of analogy as defined by researchers in many different fields, and there little distinction between analogy and other types of problem-solving that use prior experience. Second, the study of analogy is still at its infancy, and even though it plays major roles in learning by examples and learning by being told, such studies do not specifically account for the analogy process. Third, due to the complexity of analogy and analogical thinking, researchers have concentrated on subsets of analogy and have provided results that are disjoint from other studies.

Eliot's definition of Analogy. Eliot defines the term analogy as representing both an artifact and a process. He refers to "analogy" as an artifact or a static product arising out of the analogy process. He also refers to "analogical thinking" as the process which produces analogy. Although Eliot does not maintain any further distinction between analogy and analogical thinking in his work, he claims that both consisting of three features: 1) representations for attribute knowledge, 2) representation for the relationship knowledge, and 3) operations working upon attributes and relationships to produce analogy.

As part of Eliot's definition there must be a knowledge base available which contains prior experience. The knowledge base must consists of problems and their corresponding solutions. Thus the requirement for analogy deals with problem/solution pairs, and a matching between these pairs. These aspects distinguish analogy from other types of problem-solving methods.

Eliot's suggestions in the usage of analogy with expert systems: Eliot recognizes three different issues in expert systems that suggests the use of an analogical problem solver (APS):

1. Excessive effort required to build expert systems.
2. Difficulties in articulating knowledge used by experts and encoding a knowledge base.
3. Proper system interaction with the end user.

The first of these issues attempts to reduce some of the efforts encountered while building new expert systems. Eliot's suggestion on this matter is to use prior experience used in selecting the proper tools for building previous expert systems as an aid to help in the selection of proper tools for the new expert system. He suggests the development of automated packages utilizing APS to resolve such selection. The second issue addresses the utilization of an APS in the knowledge acquisition stage of building expert systems. The APS component can be used to derive rules automatically and with the help of the knowledge engineer through determining relation ships among different cases. Finally, the issue of end user interface is addressed. The goal of this issue is to equip systems with the ability to tailor the explanations specifically for each user. It also suggests using analogy in teaching the system users new concepts. As an example, the systems knows that a certain user is an experienced Pascal programmer who is trying to learn the C language. Therefore, the system uses analogy between pascal and C in the teaching process.

Although the work presented by Eliot is not directly related to our topic in reasoning, it provides some important thoughts to be considered for enhancing the development of expert systems. These thoughts however could be limited in the difficulty associated in implementing them. For instance, using an APS for building expert systems will require storing the steps and decisions of knowledge engineers used in the selection of there tools every time a tool is selected or rejected. This could be a difficult and an imprecise task. A tool selection package could be equally efficient and implemented simply using a rule-based system.

2.2.6 Greiner 1988 [38]

Greiner's work on analogy is based on defining *analogical inference* as the process which proposes new *conjectures* about a target analogue based on facts about a source analogue. His analogical system makes a use of analogical hints such as " A is like B ", from which only those *useful* conjectures are proposed. The intuition is that useful analogy provides the information needed to solve a problem, and no more. Such intuition leads to two sets of heuristics: one set is based on abstraction while the other is based on a preference for the most general set of new conjectures. Greiner developed a system called NLAG which uses such ideas and demonstrated its work by presenting examples to solve problems in the hydraulics domain by using hints and knowledge from the electric circuit domain.

Notations of Useful analogy

Let " $\models$" stands for useful analogical inference operator.

It has three inputs:

- a theory *Th* containing facts, rules, constraints, etc. (general knowledge),
- an analogical hint $A \sim B$ (A is like B),
- and a target problem to be solved, *PT*.

It has one output:

- a new proposition $\Phi(A)$, where Φ is an analogy formula.

Using the above notations, the analogical inference becomes:

Th, $A \sim B \models_{PT} \Phi(A)$

which can be translated into English as follows: Given a theory *Th* and a hint $A \sim B$ we can use analogical inference to solve a target problem *PT* resulting in new knowledge $\Phi(A)$.

Learning: Let σ be a new problem to be solved. A deductive system can solve σ only if it was in the closure of the theory *Th*; $Th \models \sigma$. Learning is one step to expand the deductive clouser of *Th* by adding a proposition *p* to the theory *Th* to form $Th' \leftarrow Th \cup \{p\}$. To ensure that our system has learned something, *Th'* closure must be larger than *Th*. This means that *p* did not already exist in *Th* and is not deducible from *Th*, ie. $Th \not\models p$. To keep the resulting *Th'* consistent we assume that *p* is not known to be false.

Learning by analogy: Greiner now introduces the first to inputs described in the notations above. In addition to the theory *Th* he uses the analogical hint $A \sim B$. A learning step is learning by analogy if a new sentence *p* is about the target analogue. Using the same notation above, *p* stands for $\Phi(A)$, where A is knowledge about the target.

Useful analogy: Greiner goal is more specific than the general analogy above. He is looking only for analogies which helps to solve the target problem *PT*. He now introduces the third input (*PT*) to the analogical formula. Now $\models_{PT}$ returns only those Φ(A) conjectures which are useful to solve the problem *PT*. Formally speaking *Th* U {Φ(A)} $\models$PT.

The problem with above model so far is that given a hint does not provide which properties nor how they should be mapped from a base problem to a target problem. This may lead to a large number of analogies, few of which are useful. The goal is therefore to find the useful analogies efficiently and to find ones which are most likely to be correct. To help with this matter Greiner introduces two s(*' ^^^S*^Turistics to find the best analogies from the search space of analogies. The first set is called *abstraction based heuristics*, while the second is called *minimal constraint heuristics*. The basic principle behind the abstraction based heuristic is that the analogy process must only consider knowledge which is represented as chunks of information with causal connection among them. Such chunks of knowledge or as called abstraction, when generalized would constitute a connected sequence of steps that can solve problems either partially or completely. With such principle, unconnected knowledge would be considered irrelevant for the analogy process. This principle is actually what Greiner called H_{ABST} heuristic. While this heuristic restricts the set of allowed analogies, the remaining space remains far from trivial. For this, Greiner introduces two other heuristics. The first is called H_{JK} or the Justification Kernel for the purpose of selecting the relevant abstraction at the proper order. The second, is called H_{CD} or Common Domain, which requires that all the constants or values used to instantiate an abstraction to be from the same domain. This rule therefore preserves the integrity of the target conjunctures from being mixed up with values coming from the base problem (which could be from a completely different domain.)

The next set of heuristics introduced are those called "Minimal Constraint"- based heuristics. The first of these heuristics is called H_{FC} which stands for fewest conjectures. This heuristic is used to select an abstraction that would result in a less number of conjectures added to the initial theory. A second heuristic is the H_{MGA} which prefers the selection of more general abstractions over specific ones. Finally, the third minimal constraint heuristic is the H_{HT} standing for findable terms. This heuristic insists that all the constants used to instantiate an abstraction to have some support from the initial theory. This is to avoid using meaningless terms for instantiation.

In summary, Greiner's work has been aimed on establishing a general theoretical frame work for analogical reasoning. His work is based on supplying the analogy process with a hint to aid in the problem solving. Using such hint, a system must only consider useful analogies which can contribute to the solution of the target problem. This was accomplished by considering abstractions or chunks of connected knowledge, while ignoring irrelevant pieces of knowledge. Through the use of heuristics the target instantiation was made consistent with knowledge in the initial theory. The learning process was accomplished by adding new conjectures to the initial theory. The constraint on these conjectures are to be non deducible form the initial theory before adding them.

In general, while Greiner's formalization of analogical reasoning has represented the analogy process quite fully, it is still considered a simplification of the theory of analogical reasoning. While the model has described methods for better selection of abstractions, it did not account for how potentially relevant abstractions are found to begin with. Additionally, it had only used domain-independent constraints such as those in the heuristics to limit the analogical inference. Domain-dependent knowledge such as the context in which the analogy is performed could also be used as constraints.

2.2.7 Kolodner 1987 [49-51]

Kolodner investigates the analogical problem solving, in particular issues dealing with memory organization and retrieving. She argues about the advantages of incorporating *Case-Based Reasoning* in the problem solving task. As a joint research, Kolodner and a number of other researchers such as Schank, Simpson, and Sycara-Cyranski, have been studying the concept reasoning from memory in relation to that of human beings. Some of the early work on this subject was the development of a theory to deal with the operation and organization of long term memory by Schank in 1978-81. The theory was called MOPs which was extended and used by Kolodner to produce what is called E-MOPs (Episodical Memory Organization Packets). This theory was applied in an early program called CYRUS (computerized Yale Retrieval and Updating System). CYRUS was only a retrieval and updating system which stores episodes about events of former secretaries of state, and later retrieve these episodes in the form of question answering. The concepts behind the long term memory of CYRIUS was extended in a case-based system that uses analogical reasoning called MEDIATOR.

This system suggests resolutions to physical, economical, or political disputes by using analogies to past cases. A particular dispute reminds the system with an analogous dispute, its resolution, along with failed mediations. Such knowledge is then used to find a solution to the current dispute.

Cases in MEDIATOR are classified according to general categories, in order to reduce the efforts needed in the reminding process. For example, given a situation where two sisters are fighting over who's right to occupy a certain room in the house, and given the situation of the dispute between Israel and Egypt over Sinai, the system would classify both cases under one general category; disputes over physical objects. Such cases are then considered analogous, and the resolution or failed mediations of one case could be used to solve the other.

Knowledge in MEDIATOR is stored in a long term memory consisting of generalized episodes. An episode is represented by a frame structure consisting of slots for narrative features, a set of indices to more specific episodes, and a pointer to a specific precedent case. The memory is organized as a hierarchy based on the classifications of these episodes. The system uses cues regarding the target case, and traverse the memory beginning with general classifications while following pointers leading to more specific episodes. Episodes that matches the target case are ranked according to the degree of similarity. The top ranked case is then retrieved and can be used to suggest dispute resolution plans, classify the target failure, or suggest a recovery.

Mapping between a source case and a target is accomplished over the frame slots. The system then would evaluate the mapped knowledge first, by direct check over the slots instantiation of the target frame to search for inconsistency, and second by consulting the user about the consequences of the mapped knowledge.

Recent work by Kolodner includes the implementation of yet an other case-based system called JULIA. The system is also based on the E-MOPs theory, and makes a use of both general and specific knowledge. JULIA is a meal planning system which suggests food menus for different costumer requirements based on previous cases.

The case-based reasoner in JULIA runs in conjunction with a constraint satisfier, problem reduction problem solver, constraint propagator. The problem reduction problem solver breaks a new case into a plan with several subgoals to be achieved. These are subgoals that belong to the organization of a certain meal. The case-based reasoner then attempts to satisfy these goals by using solutions to a previously retrieved case from memory. All goals and subgoals are posted on a global blackboard by a goal schedular so that all process in the system can have access to.

In summary, the research issues presented by Kolodner and her colleagues offer a number of contributions to the study of case-based reasoning. Most of such research has been directed towards developing models for a long term memory. These issues include methods for memory organization and retrieval. One key contribution of their research has been in presenting a retrieval model of analogous cases based on shared abstractions. In such a model, any new or existing case must be classified and fitted under a certain predefined general category. Such classification however, imposes great limitations on the system. It means that the number of classes that the system would recognize are only those with general episodes available. Addition, there could be some cases that can or must be classified under more than one general category. The model would be much more efficient if the generalization and classification processes are computed dynamically following the problem solving process.

2.2.8 Chandrasekaran and Mittal 1983 [16]

Chandrasekaran and Mittal present a model for a diagnostic system, based on using deep knowledge. Their argument is that most current expert systems are basically based on surface knowledge (experiential or heuristic), which are not powerful enough to solve more difficult problems. They attempt to show that using fundamental knowledge in diagnostic expert systems will enhance the problem solving capabilities and will allow for solving much more difficult problems. They illustrate their model with an example of a medical diagnosis system called MDX. The intuition in this work is that, deep knowledge allows for solving difficult problems while at the same time would provide better explanations to results generated from using experiential knowledge. They claim that currently there are no formal methods for representation and manipulating deep knowledge.

While Chandrasekaran and Mittal do not attempt to present a formal model for representing deep model, they present a model for the diagnostic process, and they focus on a method that makes the use of deep knowledge in expert systems more reasonable by compiling such knowledge into a well structured one.

The diagnostic process is viewed as a hierarchical model consisting of concepts. As with any hierarchical model the top levels consist of general concepts while the lower levels consists of details. With such a model, the diagnostic process becomes a top-down refinement process. When a concept is established to be true, it is refined by expanding its successors nodes. Diagnostic rules are included in each concept to determine whether it is true or not. Each concept in the hierarchy is called a *diagnostic specialist.*

The hierarchical structure of the diagnostic process, which is basically composed of different specialists, is a result of compiling general pieces of knowledge compromised of deep understanding of concepts and objects related to a specific disease. Compiling such general knowledge into a structure consisting of rules and procedures, provides a direct method for accessing and limiting the information needed for a specific problem. In other words, only those pieces of deep knowledge that are needed to solve a given problem are compiled into the structure. General deep knowledge is referred to as U, while the subset of U that is compiled, is referred to as D structure.

With such model it is expected that the system would provide better explanation based on fundamental knowledge, just as human experts do. The system can answer questions based on the D structure, by tracing back the rules executed in the solution path of the hierarchy. However, there are situations where the compiled knowledge in the D-structure is not sufficient to answer more difficult questions which depend on knowledge left out in U during the compilation process. To solve such problem without going back to the ill structured knowledge in U, the system can insert explanation in within each specialist at the time of compilation and based on knowledge in U.

In summary, the work presented attempts to explore the concept of using deep knowledge in expert systems. The work was basically focused on the advantages of deep knowledge in terms of explanation rather than problem solving. Additionally, the work was directed more towards understanding the diagnostic process rather than developing formal methods for representation, or explaining how the compilation process takes place. The major contribution of this work was the idea of compiling deep knowledge to form a subset of knowledge that lies between the ill-structured deep knowledge and a less rigid surface knowledge.

2.3 Application of AI To Engineering Design

2.3.1 Huhns and Acosta [1,43]

Huhns and Acosta, two researchers at the Microelectronics and Computer Technology Corporation, have developed a distinguishable system called **Argo**. They claim that Argo is a tool for building knowledge-based systems that improve with use. Generally speaking, Argo is a tool that incorporates analogical reasoning to solve new design problems. It performs its problem-solving activities in the form of design plans. These plans are stored in memory in different levels of abstraction. The key point about argo is that inexact analogies become exact at a higher level of abstraction.

Argo's analogical reasoning solves new problems by using plans of previous solutions. A plan in Argo is a directed acyclic graph having nodes corresponding to design rules and edges corresponding to dependencies between the rules. Knowledge in a plan is stored in a hierarchical form with most general knowledge being at the top of the hierarchy. The learning process in Argo computes different levels of abstractions for solved plans. The levels of abstractions are performed automatically by the system simply by deleting leaf nodes from a plan. The more leaf nodes are deleted the more abstract a plan will be. Therefore, given two plans that are considered different at a certain level of abstraction may become equal at a higher level of abstraction.

The problem-solving strategy in Argo consists of two cycles. The first cycle is the design cycle based on refinement procedures on the behavioral specifications of design artifacts. All the steps of the design solving process are stored in the form of hierarchical plans. The design process is usually accomplished manually by a designer whose goal is to train the system on how to solve such problems. Once a plan resembling a full or partial solution is constructed, the second system cycle takes effect. The second cycle thus marks the beginning of the learning phase in Argo. It is accomplished by generating and storing abstract plans of different levels for later use. This process is then, followed by computing *macrorules* for each edge in each plan with a proper form of explanation. In other words, the system does not explicitly learn the design plan or abstract plans. Instead, it compiles rule instances of each plan into a set of macrorules embodying the plan's relevant preconditions and postconditions. The macrorules are then inserted in a partial order, depending on their level of abstraction. The macrorules computation is considered the generalization method in Argo.

In order for Argo to reuse its knowledge to solve new analogous design problems, it will search the space of macrorules and attempts to find the most applicable but least abstract form of macrorules. This means that the system starts its search first for more specific macrorules, and then for more general ones, and so on, until a match is found.

In summary, the operation of Argo begins first by getting trained by a designer with some design examples. The system construct plans with different levels of abstraction to embody the problem solving strategy used in the training examples. Macrorules are then computed as a form of generalization, and are stored for later use. Through the use of macrorules, the system is able to solve new design problems based on explanations of previously solved problems.

The system however has some limitations. The first of such limitations is in the abstraction method used. Argo accomplished such task simply by deleting leaf nodes from the plan. Such deletion could result in eliminating important information or details that are necessary to solve certain problems. A second limitation could be a result of the training process. The system must be trained with sufficient number of examples so that it can solve analogous problems. Such training could be a difficult task if to cover a large spectrum of different design situations. A third limitation is that the system cannot learn anything beyond its deductive closure of its rule base, because the plans are built from an application's domain rules. Finally, the use of macrorules has a serious overhead, especially when the problem to be solve is very simple and can be solved by applying a simple sequence of heuristic rules. Determining the proper set of macrorules brings about a search problem which could compromise the system's performance especially when Argo has to search each level of abstraction for each plan.

2.3.2 Navinchandra 1988 (CYCLOPS) [60]

Navinchandra have developed a system called CYCLOPS that operates in the domain of Landscape Planning and design. This system is based on the concept of reusing past design cases to solve new ones. The problem solving process in CYCLOPS is composed of the following steps: finding the problem, setting up corresponding goals and sub-goals, finding appropriate cases and applying relevant sub-parts of the case to achieve the goals and sub-goals. The solution generated from these steps could be drawn from several cases, and from a variety of sources: from the same domain, from a similar domain, and from a completely different domain.

CYCLOPS uses a problem-solving technique called *demand posting*. This technique solves design problems by drawing analogies between different problems based on deeper understanding rather than surface features of the target and the base cases. The main idea behind this technique is that, the CBR in a design system can be controlled by asking relevant questions and modifying them appropriately. Asking questions serve as cues into memory to help retrieve appropriate precedents from which analogies are drawn. CYCLOPS performs this kind of reasoning by redefining the problem questions given to it. For example, given a problem X the program first ask: "Is there a way to eliminate X?" , "Is there a way to eliminate the causes of X?", "Is there a way to eliminate the effects of X?", and so on recursively until a solution for X is found. This method is called *dependency tracking*. CYCLOPS combines between dependency tracking and subgoal matching into one algorithm. Dependency tracking helps in identifying relevant cases, while subgoal matching extracts relevant knowledge from the cases.

The knowledge base in CYCLOPS consists of a collection of different precedents. Each precedent in turns consists of an initial situation describing the problem and includes conditions, effects, and explanation, and a solution part which includes an action, effects, and explanation. Once a precedent is matched through asking and answering questions through dependency tracking, it becomes a duty of the subgoaling mechanism to try and satisfy each of the subgoals in the precedent which in turn will satisfy the subgoals of the target problem. Instantiation of knowledge in the precedent with that of the target is accomplished by the system through tracing an object hierarchy. Such hierarchy represents a universe of related objects. Objects at the same level that are close to one other (a fixed value specified by the system designers) can be matched together.

The learning process in CYCLOPS begins after the demand posting completes the solution of a new design problem, in which the new solution will be stored in the knowledge base for later use in solving other problems.

In summary, the demand posting technique appears to be a reasonable form of problem solving and reasoning. The concept of asking questions on how to eliminate problems in design, and accordingly retrieve previous cases that could answer the questions, seems to be similar to that used in human reasoning. However, the design process is clearly more involved with just simply fixing problems. The work presented here did not address other issues in design, or at least did not mention any other form of questions beside elimination ones. An additional problem with demand posting that was present in CYCLOPS is that, if there exist more than one independent cause to a problem, the system would only recognize the existence of the first one only.

One other limitation of CYCLOPS is the enforcement of the closed world assumption on the knowledge base. This assumption is due to the fact that CYCLOPS does not perform any evaluation or verification on the result of the analogical process. This is a serious limitation such as that in ARGO above, hence the advantages of analogy and learning is to establish knowledge which is not deducible from the knowledge base in the first place.

Finally, CYCLOPS does not address the issue of failing to find an analogous case in the knowledge base. This is a serious problem that have to be accounted for in systems that are based on using previous cases to solve new problems.

2.4 Summary of Past Research

We conclude this chapter with a number of points which will serve as objectives to satisfy our goal in the development of a general frame-work for design.

- The literature shows that there is a continuous interest in the integration of AI and engineering design.
- The inadequacy of rule-based systems alone as a tool for the automation of highly complex domains such as engineering design.
- Understanding engineering design and the cognitive behavior of designers are essential for the development of expert systems that can demonstrate intelligence and reasoning capabilities.
- There is indeed a lack of application-oriented systems that employ sophisticated reasoning techniques such the use of analogy or first principle reasoning. Most existing systems are based on production rules.
- The literature shows that there has been many attempts to understand analogy, however much of such research did not present complete models of analogical reasoning.
- There is a great need for applying analogy to systems that use real-world design examples. Trivial examples ignores a major problem encountered in real-world situations which has to do with the representation of Background Knowledge.
- Much of the research on the application of analogy did not treat the issue of unsuccessful analogy in their systems. The assumption has been as if a new solution will always be generated based on the analogy process.

CHAPTER THREE

Application of Knowledge-Based Systems to Kinematic Structural Synthesis of Mechanisms

Highlights

- Kinematic Structural Synthesis
- Graph Representation
- Applications

Application of Knowledge-Based Systems to Kinematic Structural Synthesis of Mechanisms

Chapter Outline

This chapter presents a highlight of our earlier attempts to integrate Artificial Intelligence techniques with the Mechanical Engineering domain. The chapter begins with some background in the mechanisms field using Kinematic Structures and then presents a knowledge-based system which utilizes the kinematic structures approach for the synthesis of various types of mechanisms. The work presented focuses on integrating a rule-based system with procedural languages. The chapter shows the advantages of this approach and points out the limitations of such a system.

The system presented in this chapter utilizes a procedure for synthesizing design alternatives based on the principle of separation of structure from function. This principle in turn resolves the problem of knowledge representation of design alternatives through the use of graph structures which are then evaluated using a set of heuristic rules. This expert system has been implemented and tested for the conceptual synthesis of *variable-stroke engines* and *robot-hands* kinematic structures.

3.1 Knowledge Representation using Graphs

The purpose of using graphs is to capture the essential topological characteristics of a mechanism and representing such a mechanism with simple vertices and edges. As an example, Figure 1 shows a schematic diagram of a slider-crank mechanism and its associated graph of the kinematic structure.

In this case, the graph has *vertices* (corners) that correspond to *links* of the mechanism, and *edges* (straight lines) that correspond to the *joints*. The edge-connection of vertices within a graph therefore corresponds to the joint connection of links within a mechanism. These edges are labeled according to the type of joint that connects the corresponding links together. Note the correspondence between the labeling of the mechanism with that of the graph. In this way the essential knowledge of the structure of the mechanism has been represented by a graph.

Links are classified according to their number of connections. A binary link for instance, can only be connected to two other links, while a ternary link can be connected to three other links, and so on. Joints on the other hand are classified according to the types of permissible movements they possess. The number of permissible movements by a joint$_i$ is called the degree of freedom (f_i) of the joint. A sliding pair (P) and a turning pair (R) each has a single degree of freedom, while a gear pair (G) has two degrees of freedom.

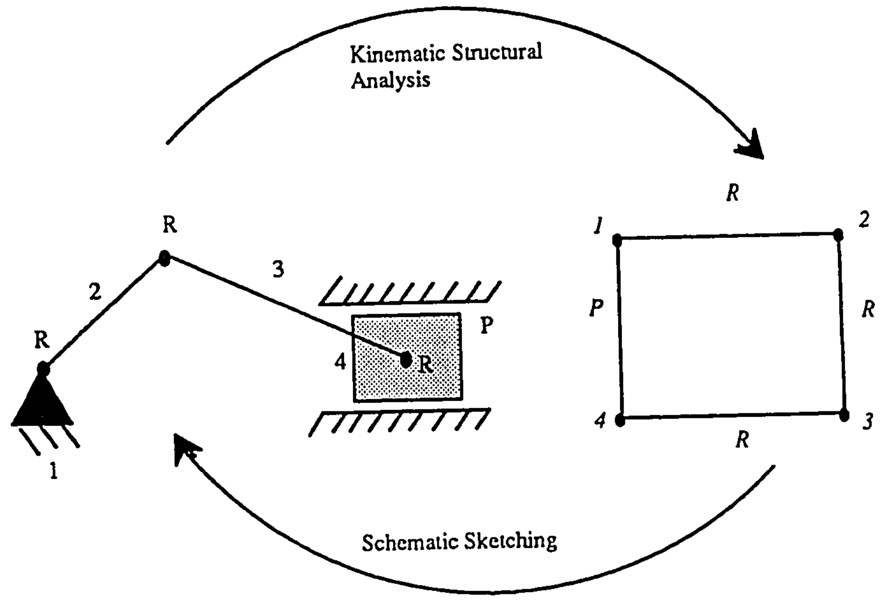


Figure 1. Slider-Crank mechanism and graph of kinematic structure

An important property of a mechanism is its overall degree of freedom (F). This property imposes a constraint on the type of the mechanism structure as well as the number of movements permitted by the mechanism. The degree of freedom can be calculated using the following equation:

$$F=\lambda(l-j-1)+\sum_{1}^{i}f_i \qquad (1)$$

where:

F: degree of freedom of mechanism.

l: number of links in mechanism.

j: number of joints in mechanism.

f_i: degree of freedom permitted by the i_{th} joint.

λ: mobility factor. Planar = 3 and spatial = 6.

This equation generally known as Grubler's equation does not apply when the mechanism has special dimensions. However, in this investigation, with the emphasis is placed at the conceptual design stage, mechanisms that possess such special dimensions will not be considered.

As an example, Figure 1 represents a planar mechanism (λ = 3) with four links (l = 4), four joints (j = 4), and each joint has one degree of freedom. Applying this information into Eq.(1) results in:

F = 3 (4 - 4 -1) + 4 = 1.

Graph representation provides a consistent method for synthesizing kinematic structures of various types of mechanisms. This makes it computationaly possible in terms of a computer program. Hence, for such a program to be feasible it must follow a set of constraints to limit the number of possible graph chains (templates) which are described in greater detail in Section 3.2. The graph generation process could be constrained by the designer's specifications which directly imposes constraints on the complexity, efficiency, functionality, and the cost of the mechanism. Typical constraints include the number of links, joints, degree of freedom, number of independent loops, and the type of joints used. For example, if the specifications require a planar (λ =3) mechanism with a degree of freedom (F) equal to one, and the types of joints used are limited to those that have only a single degree of freedom (f_i=1), then the relation between the number of links and joints can be determined using Eq.(1) as follows:

$$F=\lambda(l-j-1)+\sum_{1}^{i} f_i$$

$$1 = 3\,(l - j - 1) + j$$

so that:

$$j=\frac{(3l-4)}{2} \qquad (2)$$

It is also possible to determine the relation between the number of independent loops of a graph and the number of links and joints using the following formula:

$$L_{ind} = j - l + 1 \qquad (3)$$

Thus if the number of links and joints are known, then it is possible to calculate the number of independent loops that a graph chain would contain.

3.2 MOTIVATION

Kinematic structural synthesis is the process of designing mechanisms based on the structure (or topology) of the mechanism. This was the result of the technique of separation of structure from function proposed by Freudenstein and Maki [32,33] in the conceptual phase of mechanisms synthesis. At this point, kinematic structural synthesis has shown considerable success, yet the application of this approach has not gained wide acceptance as evidenced by the relatively small handful of applications [8,20,34,83] since the approach was first suggested. The reason may lie in the unfamiliarity of the approach as well as the deficiencies presently encountered in the application of this approach. If eliminated, the ease of application of this method would measurably improve and widen its usage. In the following section, some of these deficiencies will be discussed.

3.2.1 Deficiencies in Current Approach to Kinematic Structural Synthesis

Implicit in kinematic structural synthesis is the specification of evaluation criteria to restrict the number of candidates to a set manageable within the constraints of the available resources. These criteria are based on functional as well as structural considerations. Under current procedures, one would impose a set of criteria which may end up to be excessively strict or overly lax. The result may be a drastic sweep that wipes from further consideration potentially excellent candidates, or conversely, a retention of too many structural candidates with insufficient resources and means to crystallize to a handful of excellent ones. A cursory study of the investigation reported in [34] may convey a false sense that this problem is non-existent. It was through a careful process of sorting that the neat and tidy result presented, appeared to be so.

The second deficiency lies in the need for schematic sketching based on the kinematic structures enumerated with the aid of graph theory. While it is rather straight-forward to sketch the graph when given the schematic of a mechanism, the reverse procedure of sketching the mechanism from a given graph (schematic sketching) as illustrated in Figure 1, is not. Although it is not a difficult procedure, there is quite a bit of skill, creativity and experience needed in sketching the schematic of the mechanisms to obtain one that looks "reasonable". By "reasonable", it means proper proportioning of various parts of the mechanism as well as their location. This is where the skill, creativity and experience is called for. In other words, what is required is the capability of an expert who is able to *conceptually* perform some kind of dimensional optimization so that an evaluation of the functionality of the mechanism can be deduced just by looking at the well-proportioned schematic (which indeed may or may not be possible).

The third deficiency lies in applying evaluation criteria for elimination of non-useful kinematic structures from the large set enumerated. Designers may be prone to be biased when it comes to applying evaluation criteria (obtained from experts) to evaluate candidate schematics. This is understandable considering the tendency to evaluate candidates differently depending on the designer's psychological condition at the moment in question. It is also possible that there is a psychological need to see some of the candidate structures that have received much time and effort during schematic sketching, not to be eliminated.

Finally, current specifications of evaluation criteria have been based on sufficiency conditions. This means that there are criteria that are necessary, but in themselves, are insufficient for elimination of the structural candidates from the set. Under such conditions, it would be best to evaluate all candidates based on some numerical scoring system. Different points may be allocated to the various criteria and each kinematic structure is then evaluated based on the entire set of criteria, so that an aggregate of the points of each kinematic structure would give the designer a measure of the desirability of that kinematic structure. Also, based on the aggregate points, a ranking of the kinematic structures can then be performed.

The following section is a discussion on how knowledge-based systems may be applied to eliminate some of these deficiencies.

3.2.2 Advantages of Incorporating Knowledge-Based Systems

The incorporation of knowledge-based systems can reduce many of the deficiencies discussed in the previous section, thereby increasing the usage of the technique of kinematic structural synthesis in the design of mechanisms. Knowledge-based systems are inherently unbiased. Once the rules for evaluation are set, they will be applied consistently to all the structural candidates. Though not impossible, it may be difficult for a human designer to maintain a totally unbiased view during the long tedious process of evaluation of the large numbers of kinematic structural candidates.

Secondly, in the list of evaluation criteria, there may be rules that are insufficient in themselves to warrant structural candidates that fire such rules, immediate elimination from further consideration. Knowledge-based systems permit such candidates to be allocated points that will permit subsequent ranking of the entire pool of candidates. Such rankings permit some possibly excellent candidates (from a systems viewpoint, at a later stage of design) to remain in the pool and not be prematurely eliminated.

Thirdly, knowledge-based systems would also free the designer from the burdens of schematic sketching that involves skill and experience in proportioning mechanisms so that they look "reasonable". This dimensional synthesis process indeed should be delegated to a later stage of design. If this were not done, the schematic sketching process will leave much room for error in judgement and therefore result in the retention of poor candidates for future consideration, or worse - the inappropriate elimination of excellent candidates from further consideration. Instead of schematic sketching, the knowledge-based system will utilize purely kinematic structural rules as well as functional considerations that can be resolved into structural considerations. Most can be, as pointed out in [34]. Those that cannot be, (such as the relationship between stroke-variation and compression ratio in a variable-stroke engine) are inherently dimensional considerations that should therefore be left to a later stage of design.

Finally, knowledge-based systems permit the designer to experiment with rules in cases where access to expert knowledge is not readily obtained, or when the set of rules is insufficient to reduce the pool of candidates to a manageable number. This will not only free the designer from the fear of premature elimination of promising candidates, it would permit a learning opportunity at a much reduced level of workload.

3.2.3 Problems in the Implementation of Knowledge-based Systems for Kinematic Structural Synthesis

One of the direct difficulties with implementing knowledge-based systems to the evaluation of kinematic structure synthesis may lie in translating commonly understood terms such as a *crank* or a *connecting rod*, or a *piston* in a way identifiable by the knowledge-based system. From a schematic diagram (such as Figure 2 for a variable-stroke engine), a human may easily identify such elements, but for a knowledge-based system to be applicable, it is important that a scheme be devised such that these elements can be deduced directly from the kinematic structure.

In this chapter, the following scheme has been used. Any given labeled kinematic structure can be described by an NxN symmetric matrix with N being the number of links. For efficient storage, only the upper triangular part of the matrix need to be stored. As an example, consider a six-link variable-stroke engine mechanism [34] shown in Figure 2. The matrix representation of the kinematic structure (Figure 3) could be obtained directly from an automated graph generation stage. To identify the various kinematic elements directly from this matrix (adjacency matrix), the following procedure may be used. Since the sliding pair (P) has been constrained to be next to the fixed link (link #1), by searching for the column in the adjacency matrix where a sliding pair occurs, one can find which link is the piston. In this case, it would be link #6 since the sliding pair (P) occurs in column 6. Furthermore, one can deduce from the matrix that link #2 is the crank (a binary link). The approach is to go across row #1 until a revolute pair (R) is encountered. Then go downwards (along the column) and count the number of revolute pairs (R) in that column. If there are three or more, return to row #1 and continue along this first row until another R pair is encountered. If there are only two R's in that column, then that column number corresponds to the link number that is the crank. In this example column #2 has two R's with its top most element (in row #1) having an R. Therefore, link #2 is the crank. Link #4 is not because column #4 of the matrix has 3 R's.

By using the adjacency matrix, it is now possible to identify the various parts of the mechanism without the need for schematic sketching. More importantly, it is a way of communicating these important technical terms for use in the knowledge base.

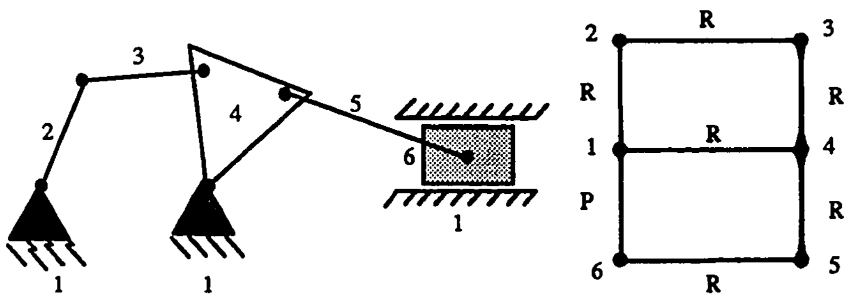


Figure 2. Design Representation of a six–link Variable–Stroke Engine

	1	2	3	4	5	6
1		R		R		P
2	R		R			
3		R		R		
4	R		R		R	
5				R		R
6	P				R	

Figure 3. Matrix representation of the mechanism in Figure 2.

Without it, the evaluation rules within the knowledge base would not be able to act on the knowledge of the kinematic structure that is implicitly contained within the adjacency matrix.

While the above example relates to that of an engine, this procedure of component identification using the adjacency matrix is directly applicable to other mechanical systems as well; similar rules can be generated to identify the various components of the system.

3.3 SYSTEM ARCHITECTURE

The objective of this system is to automate the conceptual stage in the design of mechanisms. This can be approached as consisting of two separate parts each belonging to the conceptual stage. A graph enumeration procedure which generates numerous graphs of kinematic chains is then followed by a rule-based evaluation procedure which applies a set of heuristic rules on the graphs for the evaluation of kinematic structural candidates.

The basic system architecture is shown in Figure 4 and consists of the following modules:

1. **User Interface Module:** An interfacing module which permits the user to interact both with the generation and evaluation modules, at the appropriate stage of the conceptual design process.

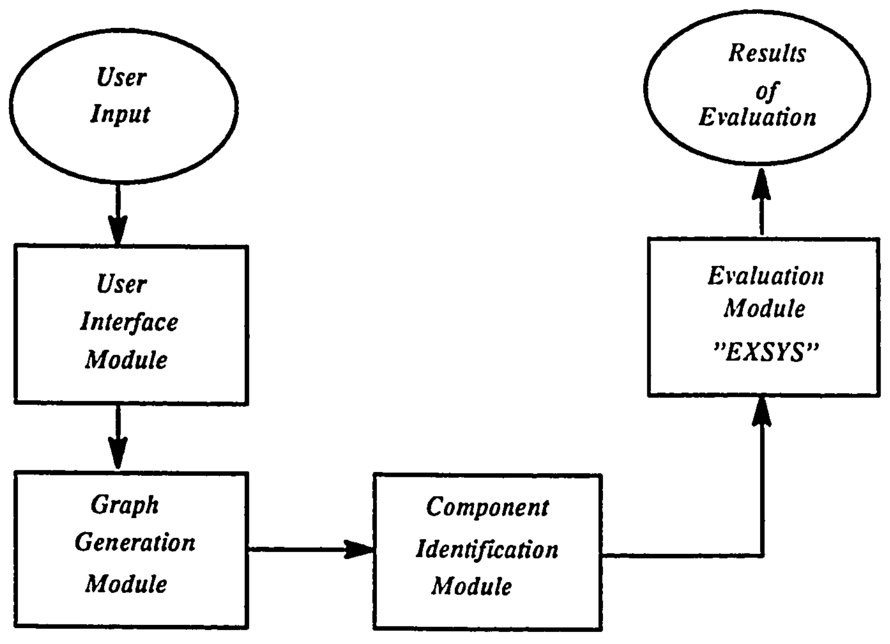


Figure 4. System Architecture

2. **Graph Generation Module:** A procedure that operates on a set of specifications, constraints, and user input to generate numerous kinematic structural candidates represented by graphs.

3. **Component Identification Module:** Through the application of certain identification rules on the graph representations, this module is able to identify the basic components that make up the mechanism.

4. **The Evaluation Module:** The evaluation module is a rule-based system consisting of heuristic evaluation rules. It takes the output of the Component Identification Module and then applies to it a set of heuristic rules to evaluate the design candidates.

3.3.1 User Interface Module

Through the user interface, a designer can select the desired application to work on. The application could be the automation of a robot-hand design, a variable-stroke engine, or any other application supported by the system. Once the application type is selected, the user is permitted to access the graph generation module. The user then inputs the desired design specifications such as the number of links and the number of joints. According to the user's input, the system offers a set of menus containing the available templates that satisfy the initial user requirements to select from. An atlas of such templates (kinematic chains) may be found in [55]. Once a template is selected, the user has a choice of entering the labels on the graphs manually or allowing the system to do so. The user will then also have the choice of browsing through the kinematic chains and selecting candidates for later evaluation.

The second part of the user interface is between the user and the evaluation module. The system presents the user with a complete analysis and evaluation results (see section 3.4) for each design candidate. The user is then permitted to investigate and request information regarding the results. In this way the solution can be traced by viewing the rules which contributed to the final decision and thereby provides that opportunity to ascertain the reasons behind the acceptance or rejection of a particular candidate. The system also readily permits the change of some of the heuristic rules and specifications to re-evaluate the kinematic structural candidates thereby permitting a comparison of the results before and after the information change. As part of this user interface, the system will redisplay the graph representation before displaying the final results. This is a useful feature especially when there is a large number of graphs to be evaluated.

3.3.2 Graph Generation Module

This module is implemented in the Pascal language with graphics capabilities. Its purpose is to create the kinematic structural candidates using graph representation. This module begins the graph generation process based on specifications such as those shown in Table 1. These specifications would permit the module to generate graphs for the

Variable-Stroke Engine Specifications	Robot-Hand Specifications
1. The search is limited to only plane mechanisms with $\lambda = 3$.	1. The search is limited to only plane mechanisms with $\lambda = 3$.
2. Joints are limited to to those with surface contact - turning pair (R) and sliding pair (P).	2. Joints are limited to Turning pair (R), sliding pair (P), and Gear pair (G).
3. Total degree of freedom (F) = 1.	3. Total degree of freedom (F) = 2.
4. The search is limited to mechanisms with only six-links and seven-joints, and eight-links and ten joints.	4. The search is limited to mechanisms with only five-links and six-joints, and six-links and seven-joints.
5. Six-link mechanisms must have exactly two independent loops, while eight-link ones must have exactly three independent loops.	5. Both five and six-link mechanisms must have exactly two independent loops

Table 1. *A list of some selected design specifications*

different permissible combinations of links and joints. The results from this module are some graph templates (unlabeled graphs) which are presented to the user for labeling. Table 1 is a sample of some of the kinematic structural specifications for both robot-hands and variable-stroke engines shown as an illustration.

By using the above specifications, the system presents the designer with the resulting graph templates. Figure 5 shows some of these templates. By labeling the arcs of a graph template in all possible ways with the allowable joint types, a chain of graphs (graphs with the same topology but with different labels) is produced. For example, for a graph with six joints, it can be labeled with two P's, one G, and three R's to give

6!/(2! 1! 3!) or 60 graphs. This labeling could be done either manually by the user or automatically by the system. One thing that the system has to be aware of in the generation of such graphs is that some of them are isomorphic. Isomorphic graphs have similar characteristic function value and therefore represent redundant mechanisms. Redundant graphs have to be detected and deleted before passing them to the evaluation stage. By calculating the characteristic function for the adjacency matrix of each graph, the isomorphic graphs can be identified by comparing their characteristic function values. For example, if the difference in the characteristic function values of two graphs is smaller than 1×10^{-9} the graphs are considered isomorphic.

The characteristic function for an NxN matrix is calculated using the method proposed in [78].

- Use the formula $CE = \det(\lambda I - A)$ for the characteristic equation where: det is the determinant, I is the identity matrix, A is the adjacency matrix representing a graph.
- Assign λ, R's, P's, and G's different random real numbers, and calculate the value for CE.

The resulting labeled graphs are actual representations of the structures of design alternatives. Each graph contains sufficient information for initial analysis and then evaluation. Initial analysis is performed on an adjacency matrix corresponding to each graph. Such analysis determines the basic components and some important features of a given design and which are passed to the evaluation module.

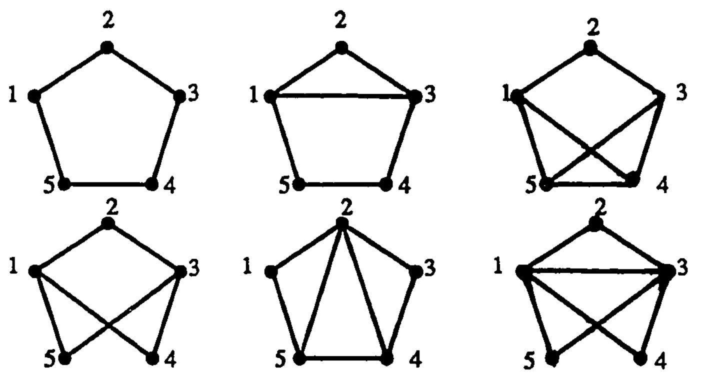


Figure 5a. Five–link templates for Robot–hand design

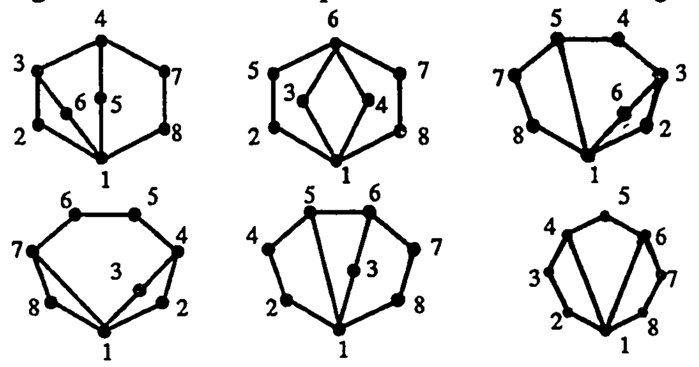


Figure 5b. Eight–link templates for engine design

Figure 5. Some of the possible graph templates

3.3.3 Component Identification Module

The internal representation of the graph structures is represented in the Pascal language as a record consisting of a number of fields, some of which are:

```
graph = record
num_joints:   Integer;            (* no. of joints *)
num_links:    Integer;            (* no. of links *)
labels:       l_array;            (* graph labels *)
dof:          real;               (* degree of freedom *)
char_equation: real;              (* characteristic eq. *)
num_loops:    Integer;            (* no. of Fund. loops *)
f_loops:      loops;              (* list of Fund. loops *)
adjacency_mat: mat;               (* NxN matrix  *)
end-of-record;
```

The adjacency_mat field contains the adjacency matrix of a graph structure such as that shown in Figure 3. It is an N x N matrix representing the topology of the graphs with N being the number of links or graph vertices. The entries in the matrix correspond to the type of joints being used. The adjacency matrix plays a major role in the operation of the system, both in component identification and in evaluation of the graph candidates. Through the application of some rules over the matrix, the system is capable of identifying consistently the type and connectivity of each component of the kinematic structure which could later be used to construct a schematic diagram of the mechanism.

The identification procedure scans each entry of each row of the adjacency matrix while applying the identification rules. As an illustration, consider the variable-stroke engine example shown in Figure 2. Some of the identification rules that could apply are:

1. Consider Row 1 (Link 1) to be the fixed link (GROUND).
2. The number of entries in a row denotes the type of link. Two entries represent a binary link, three entries represent a ternary link, and so on.
3. A row with two entries with an "R" in column 1, represents a CRANK.
4. A row with two entries with a "P" in column 1, represents a PISTON.
5. A row with two entries with an "R" in the column number corresponding to that for the PISTON, is a CONNECTING ROD.
6. A row with three entries with an "R" in column 1 and an "R" in the column number corresponding to that for a CONNECTING ROD, is a ROCKER.
7. Other rows that do not follow Rules 1, 3 through 6 represent CONNECTING LINKS.

Now applying the above rules to the matrix of Figure 3:

- Link 1 (represented by Row 1) is a ternary GROUND...(Rule 1&2).
- Link 2 is a binary link representing a CRANK...(Rules 2&3).
- Link 3 is a binary CONNECTING LINK, connecting Link 2 and Link 4...(Rules 2&7).
- Link 4 is a ternary link representing a ROCKER...(Rules 2&6).
- Link 5 is a binary link representing a CONNECTING ROD...(Rules 2&5).
- Link 6 is a binary link representing a PISTON...(Rules 2&4).

A comparison between the above results and the schematic diagram of Figure 2 shows the correspondence between the information extracted from the adjacency matrix and the components that make up the mechanism. In this way the system is able to operate on the adjacency matrix without the need for schematic drawing. All the needed information regarding a design candidate is extracted in this manner and the system is ready to perform its next stage of the preliminary design process.

3.3.4 The Evaluation Module

The purpose of the evaluation stage is to apply a set of heuristic rules to determine the feasibility of the design alternatives. The majority of these rules are based on the designer's experience in a certain design application and may differ from one designer to another. The evaluation rules are implemented using EXSYS [26] - a rule-based knowledge base system. They work by determining the acceptability of any given kinematic structural candidate with an associated confidence factor.

Knowledge in EXSYS consists of IF-THEN-ELSE rules stored in the knowledge base. A rule in EXSYS is made up of a list of IF conditions containing English sentences or algebraic expressions, and a list of THEN and ELSE consequences which may contain more statements, probability of a particular choice, or maybe a request to run an external program to perform tasks such as calculations, reading or updating data files, and displaying graphics.

At present, the system presented contains 85 rules. It consists of a set of fundamental rules applicable to most mechanisms, a set of rules applicable to the evaluation of kinematic structures of variable-stroke engines, and another set of rules applicable to the evaluation of robot-hand kinematic structures. These rules operate on the knowledge represented by the graphs from the Graph Generation Module, as well as the knowledge from the Component Identification Module.

Results from the evaluation module will include either an acceptance (or rejection) of a given design configuration with a given certainty factor for the decision. The certainty factor reflects the importance of those rules which contributed to the final decision. The system then presents a summary list of the analysis for that configuration.

A list of some evaluation rules for the variable-stroke engine and robot-hand designs is, presented in Table 2.

To implement such rules in EXSYS, they are transformed to an IF-THEN-ELSE format. For example rule number 7 for the variable-stroke engine could be stated as follows:

RULE # 7

IF: [Link_i = "CRANK"]

and [Link_i is not binary]

THEN: REJECT mechanism with Probability = 9/10

"This configuration could lead to complicated

cranks and high piston side thrusts"

The rule above has two conditions that have to be met to perform the THEN part. If a mechanism contains a crank that is not binary, then it is rejected with a probability equal to 9/10. The probability shows the strength of the rule condition, thus a 9/10 probability shows that the crank being binary is a necessary condition that has to be met.

Variable-Stroke Engine Rules	Robot-Hand Rules
1. Consider only plane mechanisms with degree-of-freedom equal 1	1. Consider only plane mechanisms with degree-of-freedom equal 2
2. Consider only mechanisms with six links and seven joints, and mechanisms with eight links and ten joints	2. Consider only mechanisms with five links and six joints, and mechanisms with six links and seven joints
3. All mechanisms must not contain more than two sliding pairs (P)	3. Six-link mech. must contain only one gear-pair (G), while five-link ones must contain two gear-pairs
4. The ground link of eight-link mechanisms must be ternary	4. Only one sliding pair is permitted to provide translation manipulation.
5. Any link connected to ground by a turning pair is considered a CRANK	5. Any two links connected by a joint labeled with a G are considered gear-pair
6. Any link connected to ground by a sliding pair is considered a Piston	6. Any link connected to one of the gear pairs, exists in the same loop of the gear pair, and not connected directly to ground is considered a gear-arm
7. The crank must be binary. This is to avoid complicated cranks and high piston side thrust	7. The number of vertices exceeds the number of edges representing turning pairs and sliding pairs by one.
8. The piston must be binary. This is to avoid excessive side-thrust and high piston friction	8. The number of turning-pair and sliding-pair edges exceeds the number of geared edges by the degree of freedom of the mechanism.
9. The crank cannot be directly connected to a sliding pair. This is to avoid high piston thrust and difficulties in varying the engine stroke.	9. The number of Fundamental circuits (loops) equals the number of geared edges with each circuit containing not more than one G
10. Multiple cranks results in multiple drive loops. This situation is not permitted, since one of the loops must be the control loop	10. There can be no Fundamental circuit formed exclusively by turning-pair edges. Otherwise, either the circuit is locked or the rotatability of the cranks are limited.

Table 2. *A list of some evaluation rules*

3.4. Design Examples

This section presents two examples which illustrate the system operation during the automation process. The first example is in the automation of the preliminary design of a variable-stroke engine [34], while the second deals with the automation of a robot-hand design [17]. These examples include the basic concepts described previously in this chapter.

3.4.1 Example on the Automation of Variable-Stroke Engine Design

This example shows the various steps that take place in the system during the design of a variable-stroke engine. The designer begins by selecting the type of mechanism to be designed which is a variable-stroke engine in this case. The number of links, joints as well as other required parameters are then specified. Based on this input a menu such as that shown in Figure 5, is displayed from which a desired graph template is to be selected. The designer, then begins labeling the graph template with the allowable labels, or permits the system to generate other alternatives automatically. The results of this process is shown in Figure 6, and it represents one of the design alternatives of a variable-stroke engine.

Once the user completes these steps, the system begins its component identification phase as described in Section 3.3, to determine the basic components of the

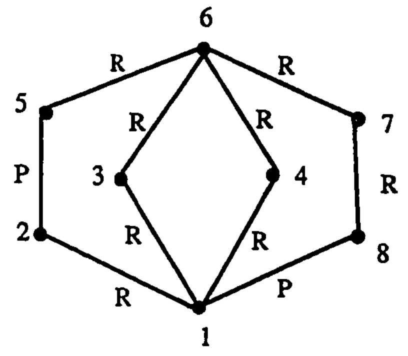


Figure 6. Eight–link structural graph

	1	2	3	4	5	6	7	8
1		R	R	R				P
2	R				R			
3	R					P		
4	R					R		
5		R				R		
6			P	R	P		R	
7						R		R
8	P						R	

Figure 7. Matrix Representation of Fig. 6

mechanism. To do so, it makes use of the adjacency matrix of the graph shown in Figure 7. Through the use of the identification rules, the system is able to identify what each entry in the matrix means. In this particular example, the system concludes that the mechanism contains three cranks (according to Rule 3 of Sec. 3.3). These are Links 2, 3, and 4. However, a prior assumption has been enforced to limit the number of cranks to one. The system in such a case, treats this design problem as three separate configurations each of which having a different crank with the other two links considered as connecting links. Each of these configurations must then be evaluated separately.

As an example, consider the first configuration where Link 2 is the crank. The results based on this configuration are shown in Table 3. Information based on those in the table are then passed on to the evaluation stage which applies the set of heuristic rules to determine the feasibility of this design configuration. The result of the evaluation stage of the this example is shown in Figure 8. This is followed by a list of the rules that contributed to the results. These rules are shown below and can be compared to those in Table 2 of the variable-stroke engine.

```
RULE #18
  IF :
     (1) Link 3 = " CRANK "
  and (2) [Link 3 is directly connected to a number of
         sliding pairs] > 0
  THEN :
         REJECT - Probability = 9/10
         " The crank is directly connected to a slider.
           This configuration results in difficulty in
           varying the stroke. "
```

values based on 0-10 system	value
1- REJECT	9
2- link number 1 =	Quaternary GROUND
3- link number 2 =	Binary CRANK
4- link number 3 =	Binary Connecting Link
5- link number 4 =	Binary Connecting Link
6- link number 5 =	Binary Connecting Link
7- link number 6 =	Quaternary Connec. Link
8- link number 7 =	Binary Connecting Rod
9- link number 8 =	Binary PISTON
10- Drive Loop =	1,2,5,6,4,1
11- Control Loop =	1,3,6,4,1
12- Power Loop =	1,4,6,7,8,1
13- Total number of links in system =	8
14- Total number of CRANKS in system =	1
12- Total number of SLIDING PAIRS in system =	2
13- One of the SLIDING PAIRS is PISTON	

Figure 8. Result of the evaluation stage of Fig. 6

```
RULE #36
  IF :
     (1) Loop 1 = " DRIVE_LOOP "
  and (2) [Number of sliding pairs in Loop 1] > 0
  THEN :
         REJECT - Probability = 9/10
         " To avoid sliding in this high-speed loop, the
           drive loop must not contain any sliding pair. "
```

Rule #18 states that if a link is a crank, then that link should not be directly connected to a sliding pair. This is to avoid difficulty in varying the stroke of the piston. Rule #36 states that there should not exist any sliding pair in the drive loop, so that sliding in this high-speed loop is eliminated.

Row # (Link #)	Matrix Analysis	Results
1	- Quaternary link - Connected to *links*$_{2,3,4}$ by R joints and to *link*$_8$ by a P.	Ground
2	- Binary link. - Connected to ground by an R,and to *link*$_5$	Crank
3	- Binary link - Connected to ground by an R, and to *link*$_6$ by a P.	Connecting link
4	- Binary link - Connected to ground and to *link*$_6$ by R's.	Connecting link
5	- Binary link - Connected to *links*$_{2,6}$ by R's.	Connecting link
6	- Quaternary link - Connected to *link*$_3$ by a P and to *links*$_{4,5,7}$ by R's.	Connecting link
7	- Binary link - Connected to *links*$_{6,8}$ by R's.	Piston Connecting Rod
8	- Binary link - Connected to ground by a P and to *link*$_7$ by an R	Piston

Table 3. Summary of matrix analysis of Fig. 9.

3.4.2 Example on the Automation of Robot-Hand Design

This section presents a second example on the operation of the system. In this example, the application problem is assumed to be the design of a two-fingered robot-hand. The schematic diagram and graph representation are shown in Figure 9 and Figure 10 respectively. The system follows the same sequence of steps as in the first example. Some final results of the evaluation stage are shown in Figure 11, followed by some of the rules which contributes to the final decision.

From the robot-hand rules presented in Table 2 it can be shown that the design presented in Figure 9 satisfies all of the evaluation rules and as a result it is considered a feasible design alternative. The following is a trace on some of the rules fired:

(*Rule #1*) It is a five-link mechanism with two gear pairs located between $link_2$ and $link_3$, and between $link_3$ and $link_4$.

(*Rule #2*) There is only one sliding pair located between $link_1$ and $link_2$.

(*Rule #3*) One gear rack represented by $link_2$.

(*Rule #4*) Turning pairs are permitted since gears are concentric.

(*Rule #5*) There is no slider-crank exists in this configuration.

(*Rule #6*) Number of R's and P's exceeds number of G's by the d.o.f. There are four R's and P's, two G's, and d.o.f is equal two.

(*Rule #8*) There are two fundamental circuits and two geared edges. Each circuit contains exactly one G.

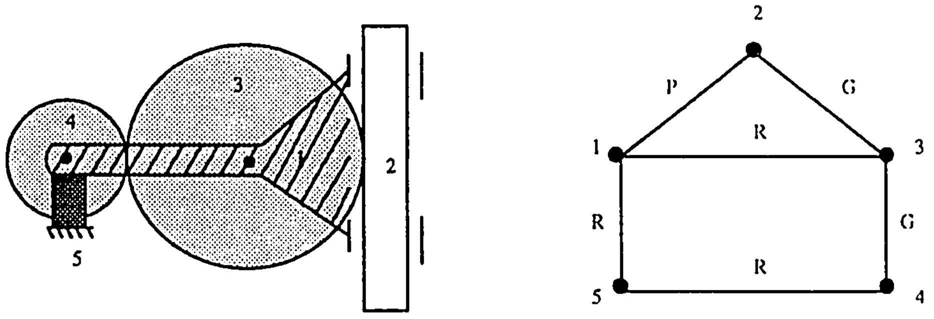


Figure 9. Design Representation of a Robot Hand

	1	2	3	4	5
1		P	R		R
2	P		G		
3	R	G		G	
4			G		R
5	R			R	

Figure 10. Matrix representation of Figure 9

values based on 0-10 system		value
1- PASSED ALL RULES, ACCEPT		10
2- Number of Links in Mechanism =	5	
3- Number of Joints in Mechanism =	6	
4- Number of Fundamental Circuits =	2	
5- Number of Turning Pairs =	3	
6- Number of Sliding Pairs =	1	
7- Number of Gear pairs =	2	
8- Degree Of Freedom for Mechanism =	2	
9- First Fundamental Circuit is	1231	
10-Number of Gears in First F-Circuit =	1	
11-Second Fundamental Circuit is	15431	
12-Number of Gears in Second F-circuit =	1	
13-Gear-pairs are link 3 and link 4		
14-Gear arm is link 1		

Figure 11. Analysis of the Design of Fig. 9

3.5 CONCLUSION

The work presented in this chapter has demonstrated the feasibility of applying existing knowledge-based system tools to the automation of conceptual design of mechanisms. This was accomplished through the augmentation of such tools with other conventional programming techniques, and through the use of an efficient knowledge representation.

The use of graph theory for the representation of the kinematic structure of mechanisms, has been proven to work very well, and has overcome the knowledge representation problem associated with the automation of highly complex domains such as mechanical design. By using graphs, the system then is capable of creating and evaluating mechanisms, in a systematic manner. A major advantage of using the graph representation, lies in its applicability to the design of a wide range of different types of mechanisms. A few examples of such applications include the preliminary design of variable-stroke engines, robot-hands, automatic transmissions, casement windows, and many others.

The automation process as a whole, has shown to provide a consistent method for the creation and evaluation of mechanisms. Large numbers of design alternatives were created and then tested for feasibility, and as a result one or more optimal design configurations have being selected. In such a process the user is completely informed with all the steps and actions taken with explanations of any given result. The user is also provided with an easy environment for experimentation of various heuristic rules. Through this environment, the user is able to change any of the design's specifications, re-evaluate the new design, and then compare it with other designs existing designs.

Deficiencies of Current Approach

While the system model presented in this chapter shows to have many benefits for aiding the designer at the conceptual design stage, it incurs a number of disadvantages and limitations. Some of these limitations include:

- The graph representation employed does not convey enough information to the designer. The designer must go through a tedious process of identifying the various components in the mechanism.

- To identify the components, a set of identification rules must be established. These rules are often difficult to construct and are based on observations of various mechanisms. Some of these rules are difficult to generalize and thus can be found contradictory in some cases.

- The design procedure employed in the system is based on exhaustive generation and evaluation of design alternatives. A design configuration is generated first then evaluated. This results in hundreds of design alternatives that must be evaluated separately.

Such disadvantages and many others play a major role in expanding our research efforts searching for a more efficient knowledge representation and a more intelligent approach in the design process. The remaining chapters are based on such motivation. In the following chapter we will explore an area of research focused on the Creative Design Process which we relate to other research in AI on analogical reasoning and learning. This is followed by a presentation of a new system model which takes advantage of reasoning by analogy in the reuse of past problem solving episodes.

CHAPTER FOUR

The Role of Analogy in The Creative Design Process

Highlights

- Tools for Creative Design
- Reasoning and Learning by Analogy
- Summary

The Role of Analogy in Creative Design

Chapter Outline

This chapter explores a very promising area of research focusing on creative design. While there have been some research efforts established in this area, we have not found much work that relates the creative design approaches to the research on Artificial Intelligence. While this dissertation includes only some background about this novel research area, we regard this direction as a very important result of our continued research. Expansion of this research area will be considered as one of the primary goals in the future.

In this Chapter we explore a number of existing tools used in the creation of mechanisms, and discuss the relationship between such tools and our research area. Emphasis is placed on tools that utilize various forms of analogical reasoning. Some of the tools described in this chapter include Design by *Implication*, Design by *Inversion*, design by *Empathy*, and *Systematic Design*. This is followed by a presentation of some of the underlying theories in Analogical Reasoning to provide a detailed study of this powerful reasoning technique. We will examine work on analogy as it relates to the Machine Learning research.

4.1 Creative Design

One major problem in the creative design process is the difficulty in determining a starting point during the conception of new ideas. This starting point marks the beginning of the *conceptual design stage* in which a large number of preliminary design alternatives are synthesized and evaluated for feasibility. If ill-structured, this stage could be both time and effort consuming, and most often will result in unsatisfactory products. However, to get started on the right path of a creative solution of a given problem, a designer must adopt some form of a well-structured and systematic design tool that could serve as a positive stimulant to his creativity.

The engineering design process begins with conceptually identifying a set of preliminary design configurations that satisfy some given structural and functional requirements. The conceptual design stage requires a great deal of creativity accompanied with a number of decision-making activities. The significance of this stage is that it requires little details and thus provides the designer with an environment for experimentation and creative thinking. However, the first problem that faces most designers at this stage, is where and how to get started in conceiving new design ideas. A designer is not always guaranteed to have a "flash-in-the-night" idea resulting in a creative solution. In addition to that, methods such as trial and error do not always produce satisfactory results and could eventually lead to abandoning the design problem. To avoid such uncertainty in design, designers rely on using certain design tools and methodologies to organize and structure the design process and to facilitate different means for creative thinking.

4.2 Tools for Creative Design

This section presents a brief discussion of some of the existing tools that are commonly used by engineering designers during the conceptual stage. Emphasis is placed on such tools that utilize analogical reasoning.

4.2.1 Brainstorming

The concept behind brainstorming is to gather a number of people (preferably ten or more) to work as a group in solving a given problem. All the activities and results of the working group is to be recorded and analyzed by a separate group of people.

To ensure successful results, a brainstorming session must follow certain restrictions and rules some of which are:

1. Group members must present any idea that comes to mind, no matter how ridiculous it may sound. This is an important rule since the goal of brainstorming is to gather as many ideas as possible.
2. Criticisms of ideas by group members are not allowed.
3. There must be a time limit for each idea and for the session as a whole.

4. Every action taken by the group members must be recorded first, and then organized and analyzed separately.

The disadvantage of brainstorming is that it can be applied only to design problems which do not require high precision and detailed analysis. A further disadvantage is that many good ideas could be lost due to the sudden and frequent suggestions by the different group members. Finally, brainstorming is a complex and ill-structured process. Therefore, it does not represent a tool that can be reasonably automated using AI methodologies. However, many researchers have used brain storming during protocol analysis to provide for better understanding of the cognitive behavior of designers (refer to the literature survey chapter).

4.2.2 Structured Approaches to Design

The structured approach (logical building-block approach) to design is used to organize the designer's thoughts and to solve a given problem using a step by step approach. The structured approach is similar to that used in software engineering and many other domains of expertise. The system model presented in this dissertation is based on such an approach. In general, this approach treats a design problem as a black box with given input, output, and functional requirements. This is followed by connecting the input and output through a variety of simple building blocks (mechanical components) so that it satisfies the functional requirements. Figure 12 summarize this approach graphically.

We can summarize the structured approach in a number of steps as follows:

1. Problem definition and establishing of problem boundaries.
2 Establish the output requirements of the problem to be designed based on the function that it must perform.
3. Establish the input requirements of the device. This include the type of input, number of input mechanisms, and the relation ship of such an input to the required output.
4. Determine and summarize the constraints on the device to be designed, such as space limitations and cost.
5. Summarize all the functional requirements of the device such as low friction, high rigidity, low wear rate and so on.
6. Select a number of simple building blocks that would satisfy the boundary conditions and the input/output requirements.
7. Using different sets of the selected components, connect to the input and output, sketch and test the resulting mechanism. This step is called the synthesis step.

The logical building block approach represents a genuine approach to creative design. It resembles in away the top down approach used in software engineering. In such an approach the designer has to use his/her knowledge and experience to conceive an acceptable design configuration. This knowledge includes complete understanding of the design problem at hand and experience in the application domain that includes the problem. The designer must also possess enough knowledge about the basic building blocks which are to be used to build a final design. The advantages of this approach is that such knowledge can be represented and utilized in a knowledge-based system such as that described in this dissertation.

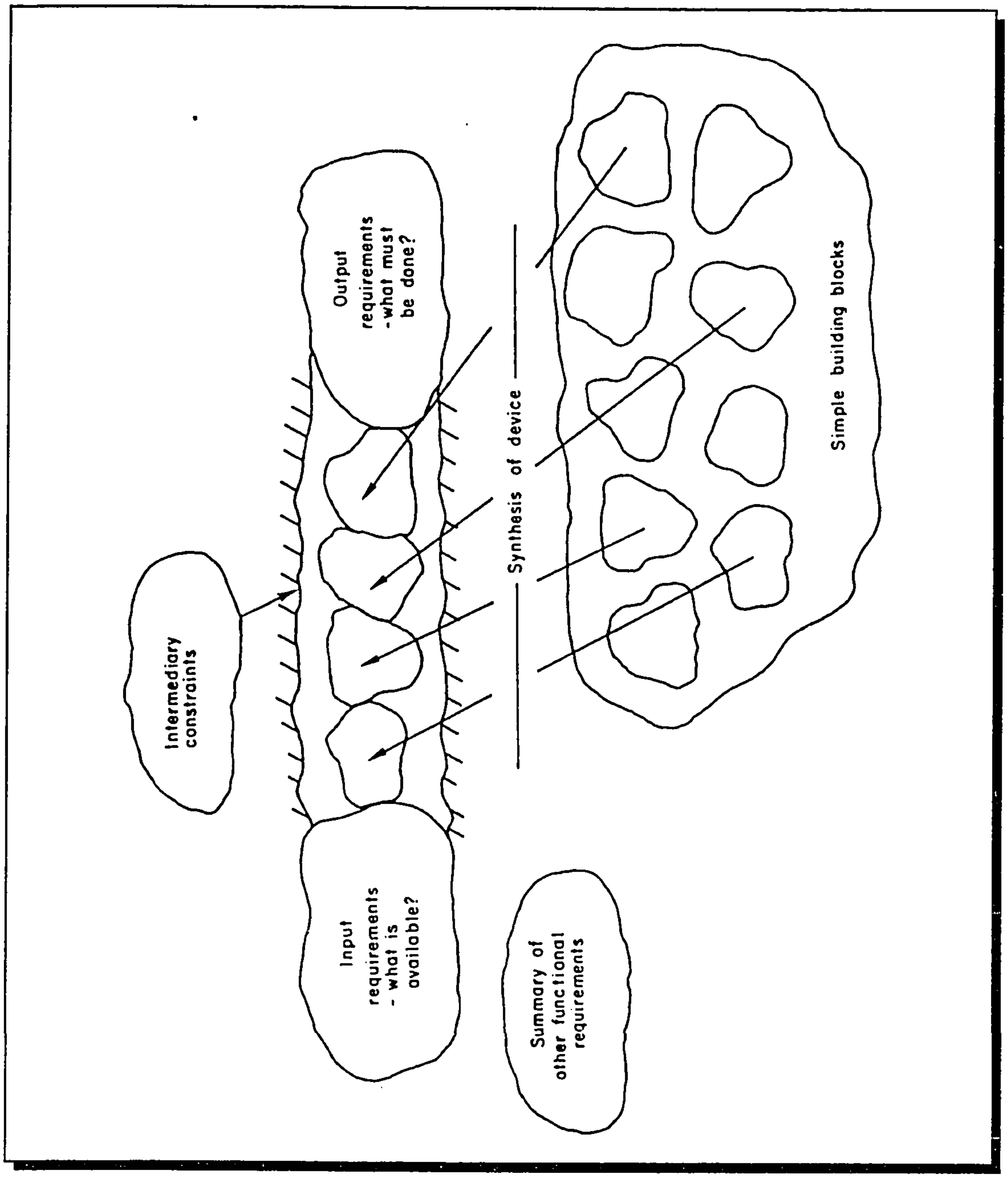

Output requirements - what must be done?
Intermediary constraints
Synthesis of device
Simple building blocks
Input requirements - what is available?
Summary of other functional requirements

Figure 12. Logical Building Block Approach to Design

4.2.4 Systematic and Morphological approaches

This approach relies on testing all the possible combinations of solutions of a given problem. The goal is not to miss any opportunity that could lead to a successful solution. Examples on this approach include breaking a design problem into several dimensions based on some functional and structural requirements [2,45]. Each dimension consists of several options or factors to be considered in the design. Possible solutions are then constructed by combining the different options from each dimension in all possible ways. For example, the design of an automobile fuel-consumption meter [45] may be broken down into three dimensions. 1) the structure of the meter, 2) indicator type, and 3) means of computation. Some of the options for the structure could be : mechanical, electrical, or hydraulic. The indicator type could be a digital, analog, or colorific display. The computation means could be continuous, analog, or interrupted approaches. The solution space in this case would be a three-dimensional volume covered by 54 combination of different solutions. Each of the solution must then be evaluated for feasibility.

An other method that falls within the systematic approach is the graph representation method for the synthesis of mechanisms. As described in Chapter 3, this approach relies on generating all the possible combination of a graph and then evaluates each configurations for feasibility.

Graphs are used to represent the kinematic structure of mechanisms in a systematic manner. The advantages of graph representation is that it provides a systematic approach for synthesizing new design configurations.

4.2.3 Design by Analogy

Analogical reasoning has always been part of the creative process in engineering design. It is used as a tool which facilitate knowledge transfer from domains which designers have experience to other domains with less experience. It is also a tool by which engineering designers can find solutions to problems that are analogous in nature. While the process of analogy may seem straight forward, it is questionable how much experience a designer must possess in the domains involved in the analogy process.

Analogy plays a major role in the conceptual phase of design in determining solutions for new problems based on ones that have been previously solved. It can be applied to problems which has similar application category such as, using an existing internal-combustion engine design, and producing a new one with fewer moving components, or one with less piston thrust and less friction. It can also be used with problems that fall into different application categories such as, using an existing engine design to produce new designs for a yoke riveter or an assembly line box-stamper.

Considering the application of analogy in Engineering Design, we can break it down into four related categories as follows: design by Implication, Design by Inversion, Design by Across-Domain analogies, and design by Empathy.

4.2.3.1 Design by Implication

Design by implication involves using an existing design solution to solve a new problem which bare some resemblance to the existing one. This could be to satisfy the need in developing a better solution to an existing problem, or could be one of taking advantage of an existing optimal design and attempt to duplicate it in some manner to satisfy a new problem.

The need for optimal solution serves as a stimulant to this type of activity. As an example, the desire for a shorter internal-combustion engine resulted in the development of the V-8 engine, instead of the longer inline engine.

Design by implication is one of the most commonly used form of analogy in design. It has also received considerable attention from researchers in the AI field.

4.2.3.2 Design by Inversion

Inversion is an important tool that facilitates the creation of new ideas. It is related in a way to analogical reasoning since it depends on using previously solved design problems. The process of discovering new ideas is not a simple task where it is highly probable that most of our new ideas are already exist somewhere. The intuition behind the inversion method however, is to take an existing idea and study it from different viewpoint to solve a given goal. This goal could be one of inventing a better solution than an existing one, or to solve a difficult problem which has no solution.

An example will help illustrate this method. The problem is to equip an assembly robot with a more sophisticated arm so that it can move more freely and can perform a number of different functions. However, a designer knows that many attempts to do so have failed in the past for one reason or another. This could be due to cost, complexity, size, and space limitations. The designer in this case may suggest to abandon the idea of working on the robot arm and, instead, focus on enhancing the robot hand by adding more fingers and more degrees of freedom to the hand. Examples on the discovery by inversion include the 1880 discovery by Pierre and Jacues Curie of piezoelectricity. The generation of electric charges in a nonconducting crystal under pressure and, conversely, the change in volume of certain crystals subjected to an electric field. This shows that ideas that do not lead immediately to a solution can be suspended and then later can be used for the discovery of new solutions.

4.2.3.3 Design by Empathy

Design by empathy is another form of analogy used by designers as an aid to visualize a problem or a situation from a different perspective. We all have used this form of reasoning one time or another when faced with a problem that is hard to understand. When using Empathy, we put ourselves into the situation we are attempting to solve. We then imagine that we are inside or as part of the problem, then seek an analogous paths of action for its solution. Empathy is a very useful tool not just to solve a particular problem, but as an explanation tool. Take for example the task of opening a wine bottle with a cork. One way of using Empathy is to imagine that you are inside the bottle, and think how you would push the cork out. The immediate solution is to exert pressure on the cork from the inside out. Such solution is then used to find an analogous real solution from the outside of the bottle. This could result in the idea of inserting a needle in the cork and push air inside the bottle which in turns provides for the required pressure to open the cork.

As you would imagine from the example above, empathy requires a great deal of creativity on the part of the designer. Such a task would not be easy to realize in a machine. However, with sufficient background knowledge and the use of first principle knowledge, tasks that could simulate the use if empathy could be accomplished.

4.2.3.4 Design by across domain analogies

Another form of using analogy in design is the synthesis of mechanical systems using electric circuits diagrams. The analogy between electric circuits and other domains such as fluids, hydraulics, and mechanical design has always been a major topic [67]. However, Johnson in his book "mechanical design synthesis" demonstrate the usefulness of this approach with a real-world industrial problem [22]. The advantages of this approach is in the compactness of the representation form of electric circuit diagrams, hence only the major components of a design are considered. Additionally, the simplicity of electric circuits diagrams provides an experimental environment for the designer from which creative solutions are possible. This is a result of the ease of manipulating such diagrams and ability of testing them with different circuit parameters. Using circuit diagrams actually reduces the complexity of design to that similar to the systematic approaches except for the need to transfer the circuit diagram back to a mechanical design.

4.3 Analogical Reasoning From the AI perspective and the Machine Learning Research

In the previous sections of this chapter we have presented a quick overview in which we demonstrated some of the applications of analogical reasoning to creative design. Most of the research cited on the topic of creative design treats analogy as an essential tool that stimulates the creativity of designers. The following sections study analogy from a different point of view. The study is based on recent advances of the research of machine learning. A number of issues and limitations of analogy and the solutions as implemented by a number of researchers are presented.

Analogical reasoning provides an essential mechanism for learning new concepts and solving new problems based on existing knowledge of similar experiences. Analogical inference is a process which proposes new conjectures about a target case, based on known facts about a source case. However, analogical reasoning does not allow one to conclude with certainty as is the case with deductive reasoning. For such a reason, any system that uses analogy must employ some methods for evaluating the analogical inference and for repairing faulty results. A discussion of a number of evaluation methods and learning strategies are presented to gain a full understanding of the overall analogical process.

4.3.1 Computational Analogy

The use of analogy in AI (computational analogy) have begun as early as 1968 by Evans [25] who developed a system called ANALOGY which reasons by analogy to find relations among geometric shapes. Figure 13 shows the type of problems that can be solved by ANALOGY. Given the shapes A and B, ANALOGY selects one of the choice shapes (shapes in second row) that corresponds to shape C and based on the relationship between A and B. This is accomplished by generating all the rules that transform A to B, generating all the rules that transform C to each of the choices, and finally selecting a rule that takes C into a choice while preserving the relation A:B.

This work was followed by the development of a theorem prover called ZORBA by Kling in 1971 [48]. Kling's system uses analogy to prove new theorems based on solutions of existing ones. The goal of ZORBA is to restrict or reduce the search for starting clauses (Da) that could be used in proofing a new theorem (Pa). This is accomplished by using an existing proof of an analogous theorem (P). By doing so, ZORBA can reduce the search effort substantially, thus eliminating the need to search the overall data base which contains all the possible clauses.

The work presented by Evans and Kling have incurred a number of limitations. The most obvious of these limitations is that both have assumed that the source case (existing experience) is already known. Determining a source case that is analogous to a target case is usually a difficult task and is one of the major problems that must be addressed by work on analogy. Other criticisms lie in the fact that both of the above systems relied mainly on surface similarities rather than on deeper or structured knowledge.

Despite the many limitations observed in the early work by Evans and Kling, it provided research on computational analogy with some directions. However, serious work on analogy and its role in machine learning did not actually take place until the early 1980's. This is due to the difficulty involved in understanding the analogy process, and the problems associated with distinguishing analogical reasoning from other machine learning techniques. The work presented by Winston in 1980 and 1982 marked the beginning of a new era of research on computational analogy [81,82]. He implemented a system which creates new and generalized rules based on constraints shared by a source case (a precedent) and a target case (an exercise). In 1983 and 1986, Carbonell [11,12] began some serious work on learning by analogy and developed the notion *transformational analogy* using a set of transformational operators to change the solution of a source case to one suitable to a given target case. The analogy in this approach is based on similarities between the initial state, goal state, and path constraints of both the target and source cases. Some of the limitations of this approach however, is that it relies on surface features present in the initial and goal states. Furthermore, the transformational operators did not have any constraints to limit the transformation process and to avoid meaningless results. To overcome many of the limitations in his previous work, Carbonell developed the idea of *derivational analogy*, which uses a deeper understanding of the analogy process and which learns from failure.

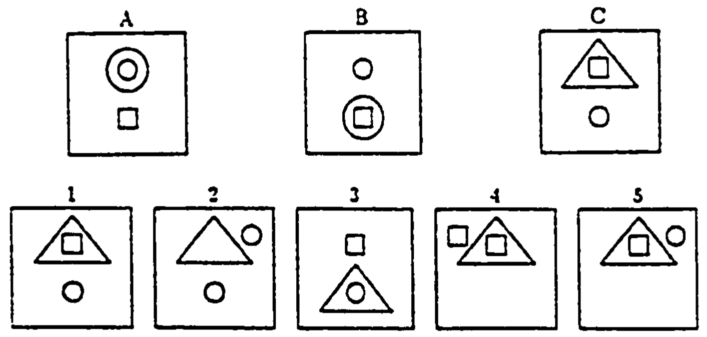


Figure 13. A geometric Problem solved by ANALOGY

In this new approach, the intuition is to rely on more information regarding the subject problems other than surface features. To do this the derivational analogy approach stores all the steps used in solving previous problems. This means that two problems are considered similar if much of the underlying plan for solving one problem can be used in solving the other.

Burstein [12] in 1986 discussed the concept formation by incremental analogy which uses multiple analogies drawn from different levels of abstraction. He used that concept to develop a system called CARL which learns the semantics of BASIC language. CARL depends on a direct feedback by the user to detect any problems in the analogy results and to correct such results by acquiring additional analogies or analogical hints.

Greiner [13] developed a system called NLAG which is given an analogical hint so that it finds an analogous case and find a solution to the problem at hand. The basic concept behind NLAG is that if we have experience in a some domain, then we could use that experience to solve analogous problems in a different domain. For example, if we have a good background in electricity and we are trying to solve a problem in hydraulics for which little knowledge exists, and if we are told that current (electricity) is like flow rate (hydraulics), then we could use such hint solve the hydraulics problem. Greiner proposed to use a set of heuristics to constraint the analogical mapping to only those useful analogies.

Kedar-Cabelli [47] introduced the concept of *Purpose* to constraint the analogical mapping from the source to a target case. His method has been referred to as the Purpose-Directed Analogy.

As a result of the research on computational analogy, a more operational definition of analogical reasoning has been established:

> "In terms of problem solving, analogical reasoning consists of transferring knowledge from past problem solving episodes to new problems that share significant aspects with corresponding past experience and using the transferred knowledge to construct solutions to the new problems. This of course must be proceeded by an efficient mechanism to recognize similarities between the existing episodes and the new problem."

4.3.2 Learning By Analogy

Machine learning is generally defined as a process by which a system improves its performance. This could be accomplished by acquiring and applying new methods and knowledge, or by improving existing methods and knowledge.

In terms of machine learning, Analogy combines both inductive and deductive reasoning. Determining analogies, between a target and a source case, based on similarity, is basically an inductive process, while transferring knowledge from source case to the target is a deductive process [57]. Figure 14 shows a possible classification of most of the machine learning approaches including learning by analogy as suggested by Mechalski.

Analogical reasoning can be viewed as an organizational framework for existing machine learning techniques. This is shown by examining the various steps involved in the overall analogical reasoning process. For example, recognizing analogies between a target and a source case without any supervision of a teacher, can be viewed as a form of learning by observation. The elaboration of the analogical mapping between source and target cases anticipates inductive summarization techniques in learning from examples. The evaluation of analogical inferences about the target domain resembles learning by discovery. Finally, the consolidation of analogical results uses inductive summarization, memory integration and indexing techniques used by various machine learning strategies.

Comparing learning by analogy with other learning approaches such as learning from examples and learning by observation or discovery, shows that analogy requires a simpler inference mechanism. This is a result of acquiring pre-existing knowledge a bout similar cases (source cases), which serves as a seed to start the analogical inference. However, the complexity of the inference mechanism for analogy is affected by the amount of assistance it receives to recognize the source case. If no assistance is given then it becomes closer to that of learning by observation.

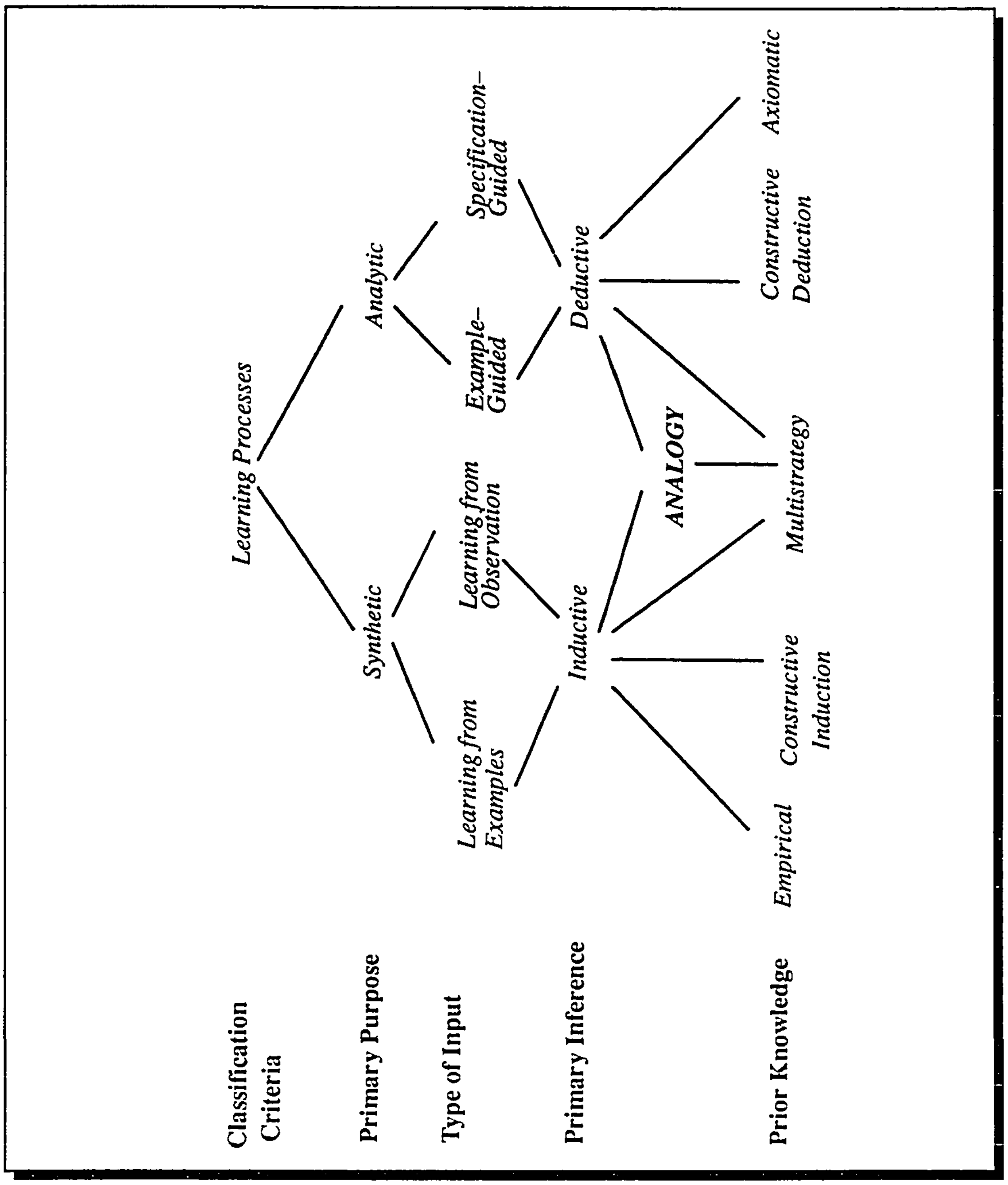


Figure 14. Classifications of various Machine Learning Strategies

Analogy can also be used with Explanation-Based learning in which a system can use previous explanations to explain new goal concepts without the need to perform a new explanation every time. This approach has been employed by Kedar-Cabelli [47] in which given a goal concept and a target example, the system first retrieve a source case that is an instance of the goal concept, and the explanation used in the source concept is then used and modified to prove that the target case is also an instance of the goal concept. Analogical reasoning could also show usefulness with an Explanation Based Learner that has a weak domain theory and that needs to use more than one example. Analogy will be useful in the sense that it provides a good tool for putting features of the examples in correspondence.

4.3.3 Learning Correct Results From Analogy

As with any inductive reasoning, analogical reasoning is a plausible form of reasoning. This means that the results are not guaranteed to be correct. In nature such plausibility is tolerated, especially since we learn in an incremental fashion. False results could be detected and corrected with gaining more knowledge. In addition to that, we always attempt to verify any conclusion that is a result of an analogy. The same argument could hold for learning machines, but whether they learn incremently or not, evaluation methods must be employed to reduce the amount of uncertainty present in the analogical inference. Moreover, methods must be used to limit the amount of knowledge that can be transferred between a source and a target case. This is especially important when working with domains that is rich with knowledge.

4.3.3.1 The evaluation problem

With a given target case and an analogous source case, the goal is to map correct and useful information between the two. The correctness of information means that it is consistent with what is already known about the target case and its existing domain theory. Insuring the correctness of analogy takes on a number of inter related steps. It begins first in the *elaboration* stage in which a set of constraints and specific goals are used to limit the mapping to only useful and correct analogies. Once the mapping from source to target is completed, the analogical inference is then need to be evaluated, confirmed and repaired if necessary.

Constraining the mapping

Several approaches have been used by many researchers on this topic. Kedar-Cabelli [46,47] proposed an approach to constraint the possible analogical mappings between a source and a target problem. Such work is intended to solve a major problem in many analogy situations in which the space of legal analogies is usually very large, while only a small set of this space is actually considered useful analogies.

Kedar-Cabelli's approach is based on using the concept of *purpose* to guide the analogical mapping and to constraint the number of allowable analogies. In his model two objects are considered analogous if a causal network of relations can be mapped from one object to the other while demonstrating how both can be used for the same purpose. For example, a "magic marker" is considered analogous to a pen if it can be shown that the former could be used for writing. In other words the magic marker can have the same purpose as the pen.

Kedar-Cabelli used his approach in "concept-learning by analogy" in which, given an unfamiliar concept, one can learn about it by mapping over many aspects of a familiar situation. This is like learning the concept of the atom from the concept of the solar system.

Gentner [35,36] proposed a model to constraint the analogical mapping called *structural-mapping*. Her model shows preference for mapping systems of predicates with causal relations, rather than mapping isolated predicates. In other words, a system that uses such model does not consider surface features of objects as necessary for analogy. It also means that two objects need not resemble one another to be considered analogous. Instead, emphasis is on causal knowledge connected as networks. Gentner also introduces the concept of *soundness*, which says that an analogy is sound when there exists a highly systematic relational structure that can be mapped into a target domain.

In summary, the systematicity principle is a good method for constraining the type of knowledge to be mapped from a base domain to a target domain. Only connected knowledge which forms causal networks are considered for mapping. Other surface features which are unimportant are considered irrelevant in the mapping process.

Greiner in his system NLAG [38] controls the mapping of useful information by the introduction of two sets of heuristics to find the best analogies from the search space of analogies. The first set is called *abstraction based heuristics*, while the second is called *minimal constraint heuristics*. The basic principle behind the abstraction based heuristics is that the analogy process must only consider knowledge which is represented as chunks of information with causal connection among them. Such chunks of knowledge, called abstractions, when generalized would constitute a connected sequence of steps that can solve problems either partially or completely. With such principle, unconnected knowledge would be considered irrelevant for the analogy process. The minimal constraint heuristics uses some specific rules to govern and to provide a preference criteria on the selection of analogies. For example, there is a rule which prefers the selection of an abstraction that would result in the least number of conjectures added to the initial theory. An other example, is a rule which prefers the selection of an abstraction that is instantiated by constants which has support by the domain theory. This is to avoid using meaningless terms for instantiating the target case.

Having constrained the analogical mapping, the next step is to measure the plausibility of such mapping as pertained to the target case or its domain.

Confirming Analogical Inferences

The analogical inference can be confirmed by either testing the validity of the resulting information regarding the target problem, or by providing justification to the analogical inference in the target domain based on similar existing justifications in the source domain. The complexity of the confirmation process depends on the representation used in representing both the target and source cases, and on the type of knowledge being transformed. Testing the validity of transformed information can be performed by acquiring some sort of expectations regarding the target case. This could be in the form of acquiring background and domain dependent knowledge. For example, Winston [80] uses what is called *transfer frames*, in which slots values of a source frame or frames are used to fill slots in the target frame. Winston uses a two step validation. First, a frame hypothesisation to determine which slots to be transferred from more than one source frame (constraining the mapping). Secondly, he justifies the validity of the target frame by testing the correctness of the slot instantiations. The justification used in Winston's system basically depends on the information supplied by the user.

The validation method used by Winston, is limited since it depends only on surface features of target and source cases rather than deeper knowledge. It also depends on heavily on the user for confirmation. A better approach has been presented by Kedar-Cabelli [47] which uses the purpose concept to direct and justify the analogical mapping. In his system the followings are given, a goal concept (eg. HOT-CUP), its purpose (eg. to enable drinking of hot liquids), and a domain theory. The system then attempts to show that a target example (eg. a styrofoam cup) is an instance of the goal concept and its purpose is to enable drinking hot liquids. This is accomplished by retrieving an existing source case (eg. ceramic cup), proving that the source case is a member of the goal concept, map the explanation (the proof) to the target case. The justification process takes the mapping over the target case (explanation) and attempts to justifying it. First, it attempts to show how the attributes of the styrofoam cup satisfy the structural and functional requirements of a HOT-CUP concept in the same way the source case (ceramic cup) was proven. If that fails, then it attempts to modify a portion of the explanation to show that the functional requirements can be satisfied by an alternate set of structural features. If that also fails, then it attempts to show that alternatives actions satisfy the goal actions. If all of these justifications fail, then the mapping is considered incorrect. In this example, the styrofoam cup satisfies most of the functional and structural requirements observed in the ceramic cup, except that the material, shape, and handle. Therefore, it will have slightly different explanation.

Burstein [9] in his learning system CARL, uses a feedback from a tutor regarding analogical inferences which includes corrections for wrong answers, and uses multiple analogies of a given concept to aid the system in concluding correct results. For example, in a tutorial session to teach CARL the concept of an assignment statement in BASIC programming language. An analogy was given to the system stating that a "variable" in BASIC is like a box. While this analogical hint works in simple cases such as "X=5", it results in a failure while reasoning about problems such as "X=Y" or "X=Y+1". When such errors are detected, a feedback for the user is supplied to the system. This feed back is in the form of extending the knowledge of CARL by supplying it with other analogies (multiple analogies) of the same concept. In this example, a second hint is given to the system from the algebra domain regarding the equality operation. Using multiple analogies therefore, permits the system to incremently extend and repair knowledge in the target domain. A trace of an interactive session between CARL and a teacher while learning the BASIC variable concept can be found in [9].

4.3.2.2 The Learning Process

Once the results of the analogy are evaluated and are proven either correct or not. The next step is to learn the new concept. This step is sometimes referred to as the Consolidation step. It involves refinement, generalization, and other techniques for the goal concept, followed by recording the results for later use. The learning could involve storing the target case (one solved by the analogy process), learning the analogy process, learning new rules, or even learning from failure.

The simplest form of consolidation is to directly record information successfully transferred from the source to the target domain. An example is shown in Greiner's system (NLAG). His system is basically a deductive system with an initial domain theory, that uses analogy to augment the initial theory with new knowledge learned from solving new problems. New conjectures resulting form the solution of the new problem and which does not already exist in the domain theory are added with out performing any generalization. While this simple form of learning may seem limited, if coupled with a powerful recognition and elaboration processes, it could achieve incremental performance improvements as the number of source cases cover a wider domain aspects.

Other more sophisticated approaches attempts to store the justification of the analogical mapping, so that they could be used in the justification of new problems with out too much effort. Winston's system described earlier, stores justifications frames supplied by the user.

Carbonell [12] uses an interesting method for consolidation in which he groups successful solution plans of similar concepts together, and also groups negative examples or failed plans and the causes of failure. Following that he generalizes over the positive examples without including any of the negative ones. The end results is a generalization of a solution plan that can be used for later use. In his later work, Carbonell stores all the derivations used in solving a problem. When a given plan leads to faulty results, the derivations are used to prevent repeating or following the same operator path resulted in the wrong conclusion. Once a solution plan is found, a generalization method, similar to that used in his transformational analogy, is applied. The generalization procedure constructs clusters of solutions that have common derivational ancestor.

4.4 Chapter Summary

This chapter began with a presentation of some of the tools used in creative design. From the studies of these tools, we've seen the importance of analogical reasoning as it plays a major role both directly or indirectly in the creative design process. Many of the tools presented can be realized into future systems. From the presentation we can conclude that our system is based on the Logical Building Block Approach, and employs both design by implication and Across Domain Analogies.

Following the presentation of these tools, the chapter then diverted to a different direction to present a detailed discussion on the research of analogy as a topic of Machine Learning to gain a better understanding to this reasoning technique. The chapter shows how analogical reasoning can be used with other learning techniques such as explanation-based generalization, to reduce the effort needed to produce explanation of a given example. We have shown that analogy resembles a general framework for machine learning that could include, implicitly or explicitly, other various machine learning strategies.

We have then explored the importance of constraining the mapping between the source and target cases to reduce the size of analogies being transferred. This is necessary so that the amount of effort used in evaluating and justifying the analogical inference is kept to a minimal. From the literature survey, it was evident that the most widely used method for constraining the mapping, has been to focus on knowledge that is connected together or ones which forms connected networks. Other methods used the concept of "Purpose" to direct the analogical mapping and to overcome the problem of ignoring relevant and disconnected knowledge which do not fall within a causal network or a structure. Directing the mapping using the "purpose" also takes care of the problem of mapping a connected structure which could be relevant for the source case but not for the target case.

We have also examined a number of analogy evaluation methods. This included the simple form of testing the slots instantiations of a target frame, using a domain-dependent and domain independent knowledge, and using direct feedback form the user. In general, the review of prior research shows that incremental reasoning by analogy and the use of multiple analogies would reduce the amount of uncertainty that could be present in the analogical results.

We finally reviewed some of the literature regarding the last stage of the analogical process; the consolidation stage. We've seen methods that basically stores specific knowledge about the target case, while using strong recognition and elaboration mechanisms. Other methods relied on storing the analogical mapping and its justifications so that they can be used in the future with similar cases. Other approaches produces an incremental generalization based on the number of stored specific cases. Thus, the more cases there is the better the generalization would be.

CHAPTER FIVE

System Overview

Highlights

- Architecture of System Model
- Control Strategy

Overview of System Model (SMARTs)

Chapter Outline

This chapter introduces the results of our research as implemented in a system called SMARTs. The chapter provides for a quick overview of this system with emphasis mainly on the Architecture and Control Strategy. The remaining part of this dissertation will elaborate on the various components of the system and the overall operation. Chapter Six presents the Knowledge Representation employed in SMARTs. Chapter Seven describes the details of each of the systems components. Chapter Eight presents a number of case studies to demonstrate some of the activities that take place during the problem solving process.

5.1 Overview of System

As mentioned earlier in the introduction chapter, the objective of our work is to develop a system model capable of aiding the human designer in an intelligent and efficient way. The work we presented in Chapter Three demonstrated one of our earlier attempts to use expert systems for the automation of design. The chapter also showed the usefulness of the graph representation in the synthesis of mechanisms. However, it was evident that such a system and the graph representation as it stands have incurred many limitations. As our objective states at beginning of this paragraph, we are in search for a methodology which demonstrates both usefulness and intelligence. As a result of our research on Creative Design and Analogical Reasoning, the consensus has been based on the concept of reusing of past design experience to aid in constructing solutions to future design problems. This is accomplished by having a knowledge base consisting of previous design cases accompanied with their description, solution and analysis. New problems are then solved through the application of analogy between the new problem (target) and existing cases. The result is possibly a number of cases from which one or more are selected based on deeper understanding of the cases, and by using some form of weighing criteria. Once a source case or cases are established, analogical transformation is applied to construct a solution to the target problem. Doing just that, the performance of a system utilizing this concept will clearly out weight the performance of other systems which solves new problems always from scratch.

Once a new case is solved, it is evaluated, and then stored in the knowledge-base using an efficient indexing method for later retrieval. This means that the knowledge base is dynamically changing every time the system is operating.

Unlike many other systems that have been developed, our system makes a use of two additional components that operate at the bottom of the case-based reasoner. A rule-based knowledge base consisting heuristic rules, and another knowledge base consisting of first principle knowledge on various mechanical components. The addition of these two components distinguishes our system form others, and results in a system with greater flexibility in terms of problem solving. The system switches from an upper level to a lower one depending on the difficulty and novelty of the problem at hand.

5.2 System Architecture

This section presents an overview of the system architecture including a brief description of each of the system's components. More details are presented in the next chapter.

SMARTs consists of three modules organized in a hierarchical structure as shown in Figure 15. Each module performs a specific task which contributes to the overall problem solving process. The system begins with the top level and control is passed to the lower levels upon need. Knowledge is then transferred from the lower levels upward. A brief description of each module is presented next.

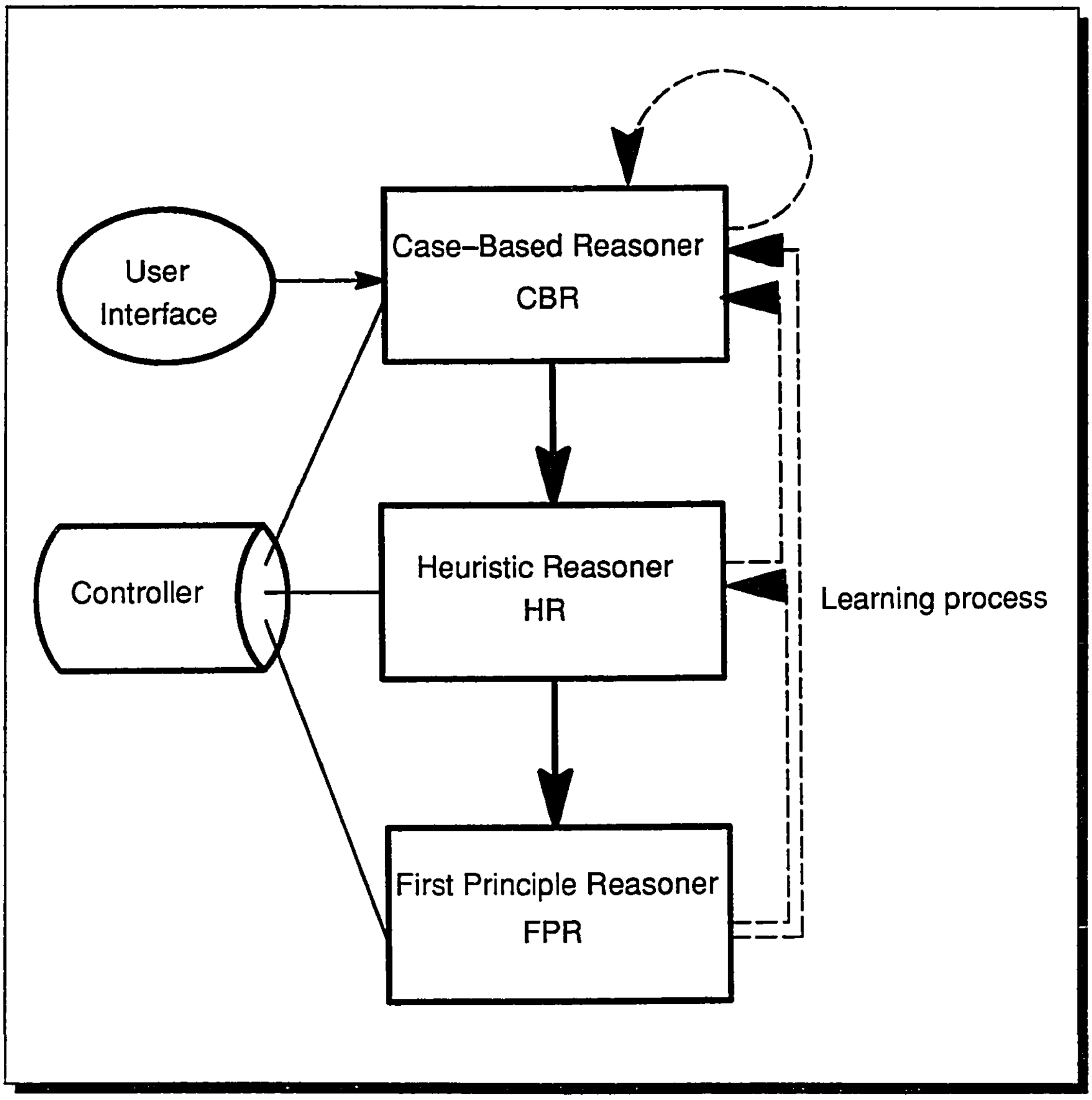


Figure 15. Architecture of SMARTs

5.2.1 The Case-Based Reasoner (CBR)

At the top level is the *Case-Based Reasoner* (CBR) which consists of a knowledge-base containing previously solved design cases. Each case is stored with its description, solution, rules, and explanation. Each case is then pointed to by an index pointer for easy retrieval. Based on the design specifications of a target case, the CBR retrieves the closest analogous case from its knowledge-base (source case). This followed by applying a set of transformation operators to reduce the difference between the source case solution and the target specifications. If the transformation procedure is completed successfully, then a solution for the target case is found.

5.2.2 The Heuristic Reasoner (HR)

At the second level is the *Heuristic Reasoner* (HR) which consists of background knowledge used to assist the CBR operation. The HR is a rule-based system consisting of design heuristic organized into several rule-sets based on the domain of the design application. These are rules of thump used by designers and based mainly on experience in design. This knowledge base consists of sets of rules, each of which is focused towards a specific design application. As an example, there will be a rule set for the design of robot hands, a rule set for the design of variable-stroke engines, a rule set for the automatic transmission design, and so forth depending on the number of application to be automated by the system. Such rules are used to determine the components of a design artifact, analyze the design, and determine the feasibility of such design. The heuristic reasoner could be considered just like a rule-based expert system with its own inference mechanism.

5.2.3 The First-Principle Reasoner (FPR)

The FPR on the other hand is a database system consisting of a library of mechanical components. This library consists of application-independent information describing the structure and function of several basic mechanical components. These are components which constitute basic building blocks of most design objects. The information about these components has no regard to the type of application they might be used in. It could information that includes the characteristics of a mechanical component, installation requirements, external factors which may limit the functionality of such component, and so on.

Along with the three major modules described above, there is a **user interface** module and a **controller**. The user interface provides means for interaction between the system and the user. It offers a number of option to the user to select from. The user interface also provides graphics to represent different designs. It could also provide some initial analysis and calculations. In order to control the three knowledge bases described above, an additional module which we call the controller is used. This module is similar to what is called a Black Board which is seen in many existing systems. Its duty is to pass control from one module to another when necessary, and to keep track of the events that take place in the system. It also controls the learning process and facilitate the propagation of knowledge from lower to upper levels. We therefore consider the controller to be a vital part of the system. Hence, by separating the control we relief the knowledge bases and their inference mechanisms from performing the extra task of switching from one level to another. The controller also makes the system more organized and modularized thus making our implementation clearer.

Knowledge residing in both the HR and FPR is initially supplied by experts in the mechanism field and is made accessible to the designer.

5.3 Systems Operation

The overall control strategy in SMARTs is depicted in Figure 16. The design loop consists of a design phase and a training phase. At the design phase, the user enters the specifications for the conceptual design of a mechanical device (target case). Then the CBR takes control and searches its knowledge-base for an existing design case (source case) that shows some resemblance to those specifications. If the CBR successfully retrieves the solution of an existing case, it solves the new design problem by transforming the solution of the source case. The transformation process is accomplished by relying on the design specifications of the target case, design rules in the source case, and on control knowledge stored in the CBR. If the CBR encounters the presence of an unknown design component in the target case, it will switch to the HR and if necessary to the FPR to extract any necessary information on that component. The final result may contain one or more solutions to the target case which are subsequently ranked according to their feasibility or desirability. The chosen solutions are then indexed and stored for future use.

If the CBR is unable to find any analogous source case, then a training example must be constructed by the designer. The training phase involves a number of activities many of which are designed to reduce the designer's effort and to ensure the correctness of the training example. SMARTs automatically accesses its background knowledge searching for applicable design rules. Rules that are found, along with their explanations, are then inserted into the appropriate locations for each of the design components. SMARTs also checks for erroneous and contradictory specifications by relying on generic knowledge available in the CBR. The resulting design is then illustrated by a connectivity graph representing the structure of the mechanism. Finally, SMARTs provides a firsthand analysis of each of the design components as well as the overall resulting mechanism. Once the training example is completed and evaluated, it is stored and indexed for later use.

```
Main-Menu: Repeat until choice = quit
case choice of:
    1:Design-loop
    2:HR-access
    3:FPR-access

Design-loop: Repeat until (choice = quit or choice = design-phase)
case choice of:
    1:Training-phase
    2:Design-phase

Training-Phase
    Enter design specifications
    Enter Connectivity of mechanism
    Draw connectivity graph
    Describe each component
    For each component search HR for relevant rules
    If rules found then replicate and store in component
    If no rules found then
        Prompt user for rules
        Store rules in HR and in component
    Evaluate resulting design using extracted rules
    If pass then Index and Store in CBR
    else Prompt user for modification and Repeat Training-phase.

Design-phase
    If CBR has no previous cases then Access Training-phase
    else
        Enter design specifications for Target-case
        Retrieve most relevant case from CBR
        If design specifications contain an unknown component then
            Switch to HR and search for any relevant rules
            If no rules are found then
                    Switch to FPR
            If unknown component is not described then prompt user
        Transform retrieved case
        If transformation results in more than one case then
            Generalize
        Index and Store case(s).

HR-Access
    Build a new knowledge-base
    Update an existing Knowledge-base
    Fire rules to evaluate a design case.

FPR-Access
    Use various database activities to enter, update, display information on design
    components.
```

Figure 16. Control Strategy in SMARTs

CHAPTER SIX

Design Representation in SMARTs

Highlights

- Kinematic Structures of Mechanisms
- Representation of mechanisms
- Other forms of representations

Design Representation in SMARTs

Chapter Outline

Before going into greater detail in the presentation of the our system model, this chapter describes the representation of knowledge employed. This includes the representation of a design case as well as other forms of knowledge representation. Since the knowledge manipulated by the system is based on the kinematic structures of mechanisms, it is important to present the necessary background in this topic. Some background on kinematics has been presented in Chapter Three.

6.1 The Automation and Representation of the Conceptual Design Stage

Our primary work in engineering design has been centered around the conceptual or preliminary design stage. Conceptual design is a very vital stage in the design life cycle because of its effects on the total design process. Conceptual design is basically a decision making activity which results in rough designs with many approximations but closely resembles the intended design.

Studying mechanical systems from the conceptual point of view, enables designers to determine the feasibility of a given design before getting involved with details. The conceptual process begins with a set of specifications and requirements of a certain design, a selection of a proper representation tool such as schematic diagrams, and finally applying experience and reasoning capabilities to the study and analysis of the structural and functional aspects of the preliminary design. Such analysis is usually less detailed, but sufficient enough to determine the feasibility of the design. Once a preliminary design is thought feasible, the designer uses the results for the implementation the actual detailed design. Conceptual design is therefore considered a necessity to reduce the cost and efforts of design. Thus it is not feasible for designers to begin the design cycle with the actual and detailed designs, then finding out later that their design would not work and have to repeat the whole process over and over again.

A Designer performing conceptual design must posses a proper knowledge of the basic components that makes a mechanical system, and be able to speculate on how such components would interact together from one configuration to another. For example, a typical mechanical system consists of a number of basic components such as gear pairs, sliding pairs, turning pairs, several types of linkages, and so on. These components are organized and structured together in many different ways to produce a higher level of components or devices such as cranks, pistons, gear racks and so on. Upon connecting these components together, a complete system is produced. The structure and connection of such components determine the performance of the resulting system. A mechanical hand or an engine for instance are kinematic systems. That is, they must have moving and rotating parts to be able to perform their intended tasks. To do so, the moving or rotating parts have to work properly and be free of high friction, vibration, excessive side thrust, and any other negative side effects. A designer is able to determine most of these factors by conceptually studying his designs and before involving complicated design equations and calculation. Before any attempts to build a system to automate any of the stages of design, a knowledge engineer must first study and develop a suitable form of knowledge representation.

6.2 Design Representation

One of the first and major stumbling blocks that is usually encountered in the automation of conceptual design, has to do with finding a proper method for knowledge representation. For such method to be considered useful, it must be capable of representing the design artifacts and their components in a systematic manner. This requirement is to ensure that the representation method can be suitable for computer implementation, and would always produce consistent results. One widely used method in the design domain is schematic diagrams. Nevertheless, while schematic diagrams are easily understood by human designers, they incur several difficulties to be understood by computer programs.

Our goal thus becomes one of finding a representation scheme that is compatible to schematic diagrams, while at the same time, have very little overhead in terms of implementation. As a result of our research, such methodology has been established using graph representations. Graph representation have been used in the mechanical design domain for quite some time. It can be used for the representation and creation of the kinematic structures of mechanisms in a systematic manner using graph theory. This means that the structural aspects of a design can be directly represented by a distinct graph containing sufficient knowledge for determining its connectivity and feasibility. With such representation a computer program need not to bother in understanding and analyzing schematic diagrams. Instead, reasoning is applied to the graphs which are easily recognized by simple programs. The graph representation therefore, offers a very efficient method for representing a wide variety of mechanisms with little regard to the application type been used.

We have been able to determine the feasibility of applying the graph representation to design-expert systems, through the implementation of two previous expert systems. One system is for the development of variable-stroke combustion engines, while the other is for the development of mechanical hands (robot hands) [14,69].

As a result of these two systems, we have gained more insight on the usage and advantages of the graph representation. We have then modified the graph representation to provide for more details and to represent a more informative knowledge representation. The new modifications were later implemented in our recent system (SMARTs). Before going into any details with the graph representation, a brief background in kinematics is necessary.

6.3 Background in Kinematics

The representation method described in previously in Chapter 3 is being used to represent the kinematic structures of mechanisms. Such knowledge representation is based on a number of concepts kinematics.

Kinematics: is a subbranch of solid mechanics which deals with the study of relative motion. Ampere defined kinematics as "the study of the motion of mechanisms and methods of creating them". The first part of this definition deals with kinematic *analysis* which is basically used to determine the performance of a given mechanism. The second part of Amperes definition deals with kinematic *synthesis* in which the required *motion* is given for which a mechanism is to be found. Thus kinematic synthesis deals with the systematic design of mechanisms for a given performance.

A *mechanism*: is a mechanical device that has the purpose of transferring motion and/or force from a source to an output. A *linkage* consists of links (or bars), generally considered rigid, which art connected by joints such as pins (or revolute) or prismatic joints to form chains. Such *kinematic chains*, with at least one link fixed are called mechanisms if at least two other links retain mobility. If a kinematic chain does not retain any mobility it is then called a *structure*. In other words, a mechanism permits relative motion between its rigid links; a structure does not. As far as our work is concerned, we are interested only in mechanisms, ie., kinematic chains with motion, and we will consider structures which permit no motion as unfeasible design configurations.

Motion: as mentioned above, a mechanism must exhibit some type motion. There are two types of motions seen in mechanisms. A *planar* motion or as called two-dimension or plane motion, is one produced by links which move in parallel planes. Planar motion consists of *rotation* about axes perpendicular to the plane of motion and *translation* -

where all points move along identical straight or curvilinear paths and all lines embedded in the body remain parallel to their original orientation. *Spatial* motion on the other hand, is a three dimensional motion. It is a combination of rotation about three nonparallel axes and translation in three directions depending on the constraints imposed by the joints.

Kinematic Diagrams: It is often difficult to visualize the movement and operation of complicated mechanisms especially when other components appear in the diagram. Instead, designers rely on sketching equivalent kinematic or skeleton diagrams. Kinematic diagrams serves a purpose similar to that of electrical schematic or circuit diagram that displays only the essential parts needed for the analysis.

6.4 Understanding the representation method

Using graphs as an aid in conceptual design has been a major issue in the mechanical domain for a long time now. Work on this topic has been established since the 1960's and is still carried on until current days. Such work has not been limited to educational institutions only, but has been a major concern of many giant industrial corporations including General Motors.

To understand this representation method we will rely on some results that have already been established by a number of leading researchers in this field, such as

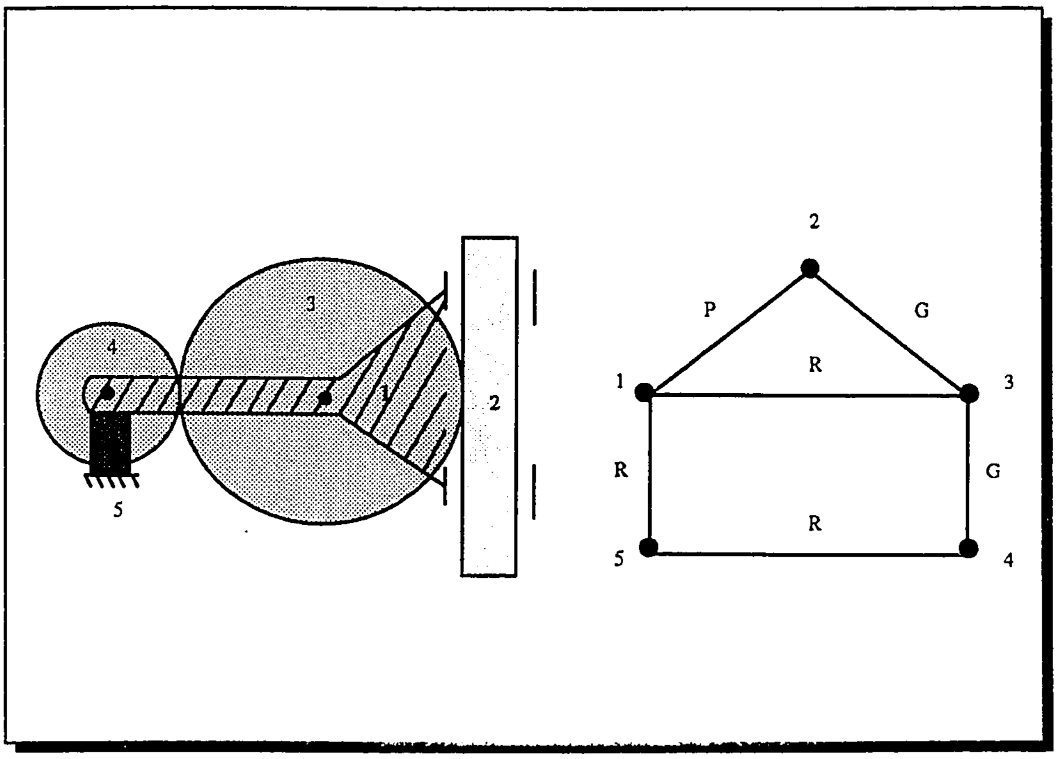


Figure 17. A schematic diagram of a two-fingered robot hand with its graph representation

Freudenstein [32-34], Woo 1967 [83] Dobrjanskyj and Freudenstein 1967 [20], and Buchsbaum and Freudenstein 1970 [8].

According to the work by the above researchers, kinematic structures can be defined precisely with the aid of graph representation. This means that the structure of a mechanism and its resulting functionality could be represented by a graph containing sufficient information for analysis and evaluation.

Figure 17 shows this type of representation. The figure includes both the schematic diagram and its corresponding graph representation. The figure represents the structure of a one finger of a two-fingered robot hand. The relation between the schematic diagram and the graph representation is usually, but not always, a one to one relation. Hence, given a schematic diagram, exactly one graph can be constructed that represent it. On the other hand, given a graph, there is a possibility to get more than one schematic diagram which satisfy the graph representation. This is usually not a major concern to designers since all the resulting designs of a single graph are very closely related.

Studying a kinematic structure from a high level of abstraction, we find that it consists mainly of two basic components: *LINKS* and *JOINTS.* And therefore, a mechanical system consists of a number of links connected to one other by several joints. A *link* is part of a mechanical system representing devices such as cranks, pistons, connection rods, and so on. Links are classified according to their number of connections. For example, a singular link can only be connected to one other link, a binary link can be connected to two other links, and so on. A *joint* on the other hand, is the part of a mechanical system which connects the links together. Different types of joints must allow for various types of movement (eg.. the joints in your fingers or arm), and they are classified according to the types of permissible movement. The number of permissible types of movements by a joint$_i$ is referred to as the *degree of freedom* f_i. For example, a sliding pair (P) joint which basically slides back and forth has only one degree of freedom. A turning pair (R) joint also has a single degree of freedom since it can only perform rotational movement. A gear pair (G) joint, on the other hand, has two degrees of freedom since it can perform both rotational and translational movements.

Examining the schematic of the mechanism in Figure 17, we see that it consists of links each of which is given a distinct number and is connected together by joints. There are a total of five links in the diagram, each of which is of a different type. For example, link_1 which looks like a *funnel* represents a *gear-arm*. Link_2 shown as an up right rectangle, represents a sliding mechanism (could be a piston). Link_3 shown as the larger circle, represents a gear. Link_4 shown as the smaller circle, represents another gear. Finally, Link_5 is considered as a common ground.

All the above links are connected to one other by various types of joints. Turning pair joints (R) represented by dots, connect link_1 to Link_3 and to Link_5. A sliding pair joint (P) shown in the figure as the connection between Link_1 and Link_2. Finally, a gear-pair joint shown as the contact between Link_3 and Link_4. Every joint in the figure except gear-pair joints has a single degree of freedom ($f_{i=1}$). A gear-pair joint has two degrees of freedom. The mechanism in Figure 17 contains five links and six joints.

Examining the graph representation in the figure, note that, links are represented by graph vertices, while joints are represented by graph edges. The graph has five vertices corresponding to five links, and six joints each with a label corresponding to six joints. A vertex i connected to a vertex_y by an edge labeled X, corresponds to a link_i connected to link_y by a joint of type X. Tracing the connectivity of the graph, you will see that it exactly matches that of the schematic diagram.

The above example illustrates the simplicity and efficiency of using the graph representation. Through the use of such representation, a major problem in the automation of a highly complex domain such as design is considered solved. The graph representation could be used to represent the designs of a wide variety of different applications such as robot-hands, engines, transmissions, casement windows, and many others.

Advantages and Disadvantages of Graph Representation

The graph representation has a number of advantages that makes it suitable for automation, however it also has some limitations. The representation of mechanisms using graphs is a compact form of representation requiring very little overhead. Graphs that correspond to mechanisms are easily generated and manipulated. Such representation is based on the study of kinematics and thus capable of representing mechanisms which are then evaluated using various rules and constraints in the kinematics field. The use of graphs thus provides for a systematic procedure that can be readily implemented in a computer program.

The simplicity of the graph representation however results in a number of limitations. Graphs are not easily interpreted to mechanisms. Graphs do not reveal all the needed information about a given mechanism. The various components that make up a mechanism have to be identified before applying evaluation rules. The identification of components is not a straight forward process. It relies on heuristic rules based on experience in this field. the result is that some of these rules may not be accurate or consistent to identify various components successfully. Moreover, although the use of graphs results in a systematic procedure, it is based on exhaustive generation of design configuration. This may result in numerous design configurations that have to be later evaluated. Such a procedure therefore lacks the basic elements of intelligence that we are aiming for in our research.

To overcome such disadvantages we have developed a representation method using a different form of graphs. The new representation is based on the traditional graph representation and conforms with all the kinematic rules. The new representation method is designed so that it can be useful in the transformation phase of the analogy process. It should also show more information regarding the mechanisms being represented.

The design representation employed in SMARTs is designed to satisfy at least two objectives. First, it must be capable of representing a complete design of a mechanism that is consistent with the definitions described earlier. Secondly, this representation must facilitate the use of analogical reasoning in finding and transforming previous design cases with minimum effort. The overall knowledge representation in SMARTs includes frame-like structures containing design specifications, graphs to represent the connectivity of mechanisms, design rules using an expert system shell designed specifically for SMARTs, and database tables containing descriptions of various mechanical components.

6.5 Representation of a Design Case

The knowledge representation employed in SMARTs is designed to satisfy at least two objectives. First, it must be capable of representing a conceptual design of a mechanism using the kinematic structure of the mechanism. Secondly, this representation must facilitate the use of analogical reasoning in finding and transforming previous design cases with minimum effort. The overall knowledge representation in SMARTs includes frame-like structures containing design specifications, graphs to represent the connectivity of links (components) of mechanisms, design rules using an expert system shell designed specifically for SMARTs, and database tables containing descriptions of various mechanical components.

Figure 18 shows the representation of a design case of an eight-link Variable-Stroke Engine. The description of a design case such as that in the figure consists of:

1) *General specifications.* This includes the Application type of the mechanism such as an engine or a robot hand design. It also includes information about the input and output requirements. That is, the type of input needed to produce a specific output. For example, in an engine design the rotation of a crank (input) produces a translational motion that moves the piston (output). The general specifications also includes and important element called Constraints. A constraint is a requirement imposed on either the input or output to behave in a certain way. As an example, the variation of the piston stroke in the variable-stroke engine is a constraint on the output.

2) *Structural specifications.* These are the structural requirements which affects the connectivity of a given mechanism. The description of these specification has been presented in the earlier sections of this chapter.

3) *Structure of a mechanism.* The structure is represented by a graph which shows the connectivity of the various mechanical components or links. Graph nodes represent the mechanical components used, while graph edges represent the joints connecting the components. Each component in the graph includes information about the component itself and other adjacent components. It also includes a set of rules that govern the connectivity of the component to the rest of the system. These rules play a major role during the transformation phase in the analogy process.

1) General Specifications:

Application: Variable-Stroke Engine
Input: Rotation /* Input requirement */
Output: Translation /* Output requirement */
In_comp: Crank /* Component used in generating input */
Out_comp: Piston /* Component used in generating output */
Constraints: Stroke Variation /* Constraints on input/output */

2) Structural Specifications:

Motion: Planar /* Type of motion supported by mechanism */
DOF: 1 /* Degree of Freedom of mechanism */
Links: 8 /* Number of links (components) in mechanism */
Joints: 10 /* Number of joints used to connect links */

3) Structure:

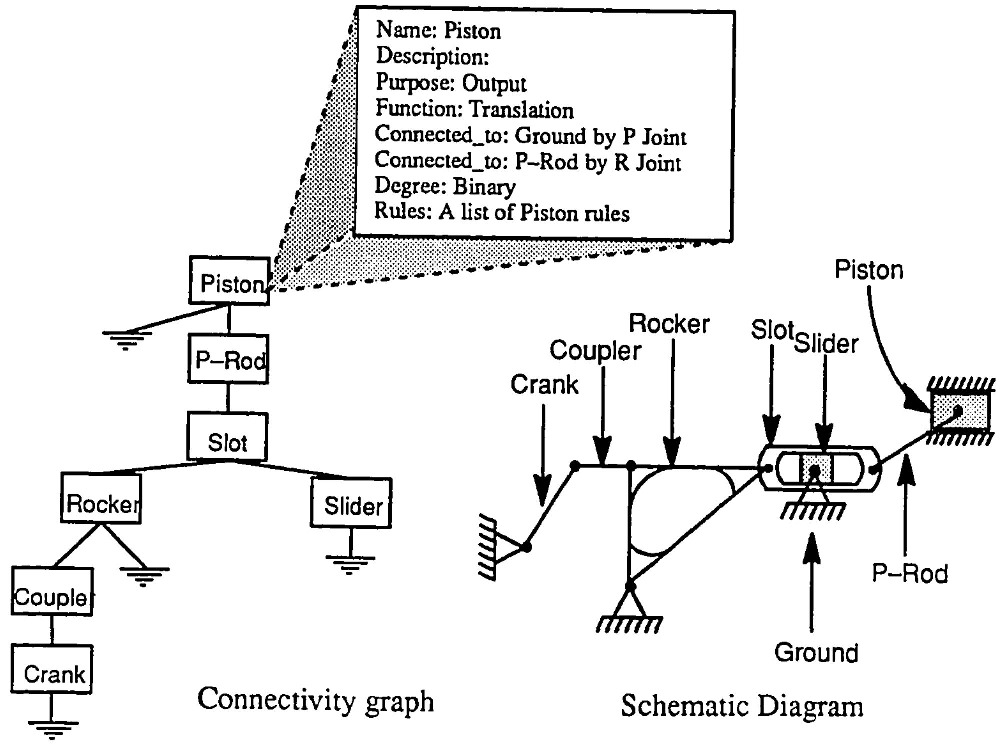

Name: Piston
Description:
Purpose: Output
Function: Translation
Connected_to: Ground by P Joint
Connected_to: P-Rod by R Joint
Degree: Binary
Rules: A list of Piston rules

Figure 18. *Design Representation in SMARTs*

6.6 Other Forms of Knowledge Representation Employed By SMARTS

6.6.1 Representation of Heuristic Knowledge

SMARTs consists of two types of heuristic knowledge. The first set of heuristics are what we refer to as control knowledge in the CBR. These heuristics are based on general knowledge in mechanisms design and are used to assist the CBR in its decision making activities. Control knowledge includes rules to determine when to add or delete design components, other rules to keep track of the design specification in case of any contradiction.

The second set of heuristics is the knowledge present in the Heuristic Reasoner. These are rules based on experiential knowledge in specific design applications. A typical rule in the HR is represented in the form of IF-THEN-ElSE rule. Each rule includes a description and a certainty factor.

For example, a typical rule in the variable-stoke engine application is stated as follows:

"The Crank should not be directly connected to a sliding pair. This is to avoid high piston thrust and difficulties in varying the piston stroke."

Such a rule is represented in the HR in the form of IF-THEN rule as follows:

IF ((link_name Crank) (Connected_to Piston))

THEN (structure Illegal)

Probability: 9/10

Explanation: The Crank should not be directly connected to a Piston. Such configuration will result in excessive side thrust and high Piston Friction.

6.6.2 Representation of First Principle Knowledge

The incorporation of First-Principle knowledge provides SMARTs with the capability to reason about and solve more difficult problems. First-principle knowledge is detailed knowledge that does not relate to any particular application, but applies to most mechanisms in general. In out system, first-principle knowledge is used to provide for description of various basic mechanical components. To provide for ready access to this type of knowledge, a database system with several tables has been incorporated. When the system needs the description of a of a specific component, it accesses the FPR and retrieves the needed information which then reformulated in to a form understandable by both the Heuristic reasoner and the Case-based reasoner. More detail regarding the FPR is presented in the next chapter.

CHAPTER SEVEN

Organization of SMARTs

Highlights

- Architecture
- Operation

Organization of SMARTs

Chapter Outline

This chapter presents the overall organization of our model. It will incorporate the various components presented in the earlier chapters. The chapter presents each of the system's modules in detail while pointing out the main characteristics and operations that take place.

Organizing SMARTs into three separate modules brings several advantages, among which are: hierarchical organization to provide a separation of different knowledge representations and inference mechanisms; modular structure, wherein a module is only accessed when needed; ready access to the various types of knowledge; easier and more efficient implementation; provision for a measure of system performance by monitoring the number of accesses from one module to another. This chapter will now elaborate on the structure and function of each of the system's modules.

7.1 Case-Based Reasoner (CBR)

The CBR can be viewed as an episodical memory containing previously solved design cases. It is equipped with an inference engine to perform a number of functions which include: search and retrieval, source case transformation, generalization, and indexing and storage. Each of these functions is described in the following sections. Figure 19 shows the structure of the CBR. It consists of an area to hold existing design cases; a System Index which organizes these design cases into proper classes and sub-classes and which controls the access to the design cases. The CBR also contains a set of control knowledge and rules to govern the various activities that will take place. It also contains an area called the Component Store which holds the description of various mechanical components that have been previously used by the system.

CBR Architecture

1. *Memory.* The system's memory is used to refer to the knowledge base containing existing design cases along with the system index which provides access to that memory. SMARTs does not employ any memory management strategy. Instead memory is viewed as a collection of design cases indexed based on their features. Based on the specifications of a target case, the index is used to search for an existing case that closely matches the target case. Once a new design case is created, it is indexed properly and placed in memory. The system index contains pointers which points to the various existing design case.

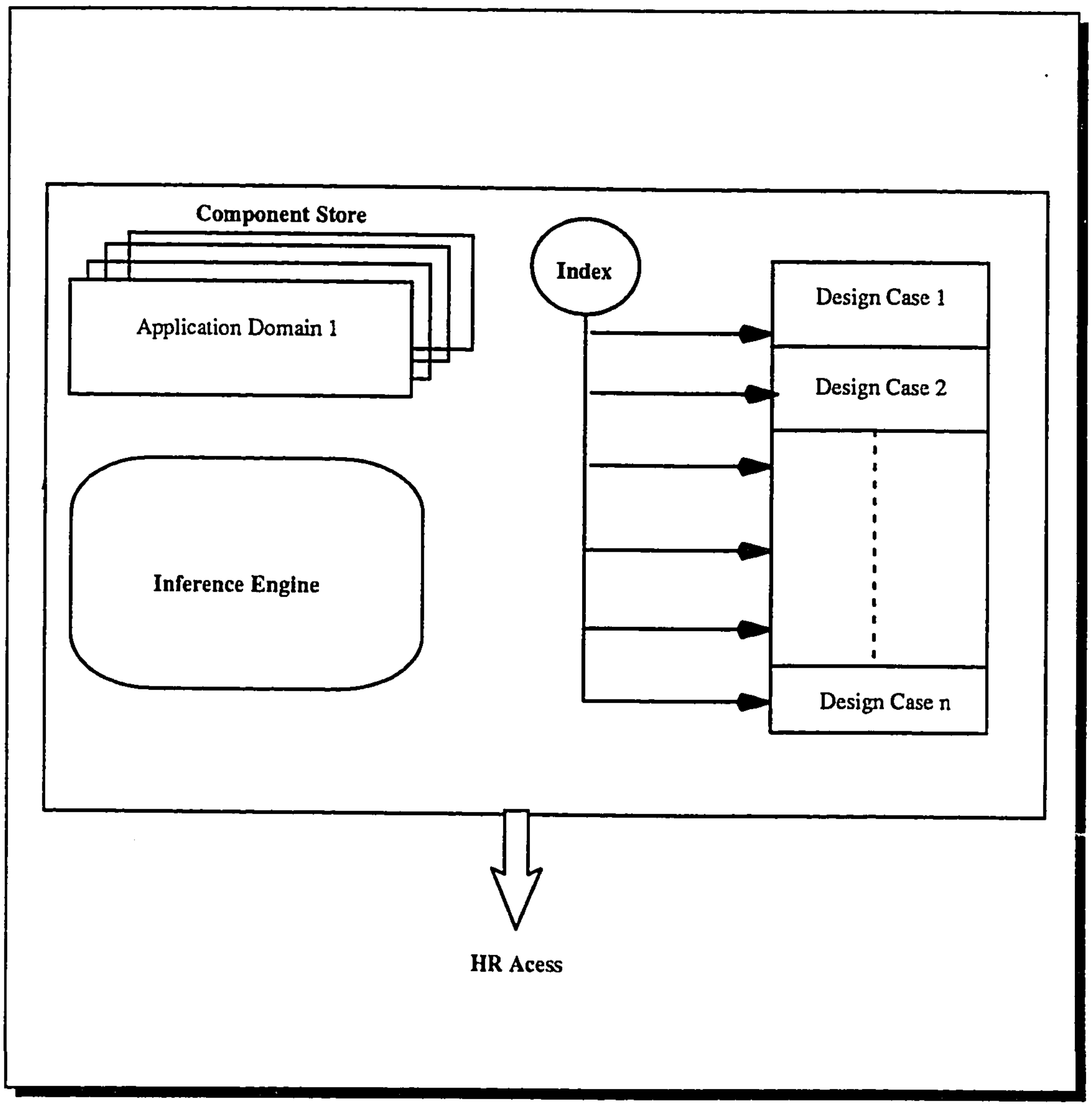


Figure 19. *The Structure of the CBR*

2. *Inference Engine.* The inference engine in the CBR is a collection of procedures and rules used to control the various operations taking place in the CBR. There are a number of complex procedures each is designed to perform a specific task. Some procedures are for rule understanding, addition of components, deletion of components, checking the specification, comparison of design cases and so on.

3. *Component Store.* The component store is a collection of various design components that have been used by the CBR. Along with the list of components, the component store contains description of each component and the type of cases that have used such component. The CBR contains a component store for each application domain. The need to maintain such a list stems from the problem of adding design components to existing design cases during the transformation phase (described later). When the CBR needs to add a number of components to a design case, it needs to know the purpose and description off hand. Having the Component store eliminates the need to search for such knowledge from the lower modules.

Analogy Frame-Work Used in the CBR

New design cases are solved in the CBR using analogy. The overall operation of the CBR can be summarized in the following steps:

1. Search and Retrieval: based on the specifications of a target case, retrieve an analogous case from memory.
2. Elaboration: If more than one case are retrieved, then use features that are not part of the initial search to determine the best matching case.
3. Transformation: Transform the solution of the source case to one that satisfies the target case.
4. Evaluation: Use the rules present in the HR to evaluate the correctness of the solutions.
5. Learning: Once a solution is found store it back in memory for later use. If more than one solution are generated then a simple generalization procedure is invoked.

7.1.1 Search and Retrieval

To facilitate the storage and the subsequent retrieval of cases, SMARTs contains a system index with a number of features that are a subset of the initial specifications. These features are organized into a hierarchy based on the degree of discrimination that a feature presents. Figure 20 shows an index with a number of cases. Each level in this index is considered as one level of abstraction. At the top level, the highest level of abstraction, a mechanism is considered as a black box with a given input and a required output. This means that two mechanisms are considered analogous at this level if both have similar input and output. Moving down the hierarchy more detail is added to the cases being compared until reaching the bottom level which contains pointers to specific design cases.

The CBR begins its search using the target specifications and comparing them with those in the index, one level at a time. This is carried out by comparing a specific target feature with all the nodes at that level while calculating a similarity metric for each of the nodes. The similarity metric is a heuristic function based on experience in the mechanism field. One such heuristic is to give a higher rank to a case with the same but opposite I/O components to that of the target case (eg. {Gear, Crank} vs. {Crank, Gear}), as opposed to a case with only one matching I/O component. Another heuristic is to prefer a source case with an equal or a larger number of components. This is due to the fact that it is easier to delete links from a source case than adding new ones. A node with features closest to those of the target case, is assigned the highest metric value, and is considered for expansion in the next step. The system also employs a one-level look-ahead mechanism to examine the expansion of nodes that do not necessarily have the highest metric value but are very close to it. This provides the search process with greater flexibility in selecting the best path that would lead to a matching source case. If the this search results in more than one design case, then additional and more detailed information that were not part of the initial search will be used to select the best matching case. Such information, for instance, may include the preference of a design case that contains a specific mechanical component or a specific joint type.

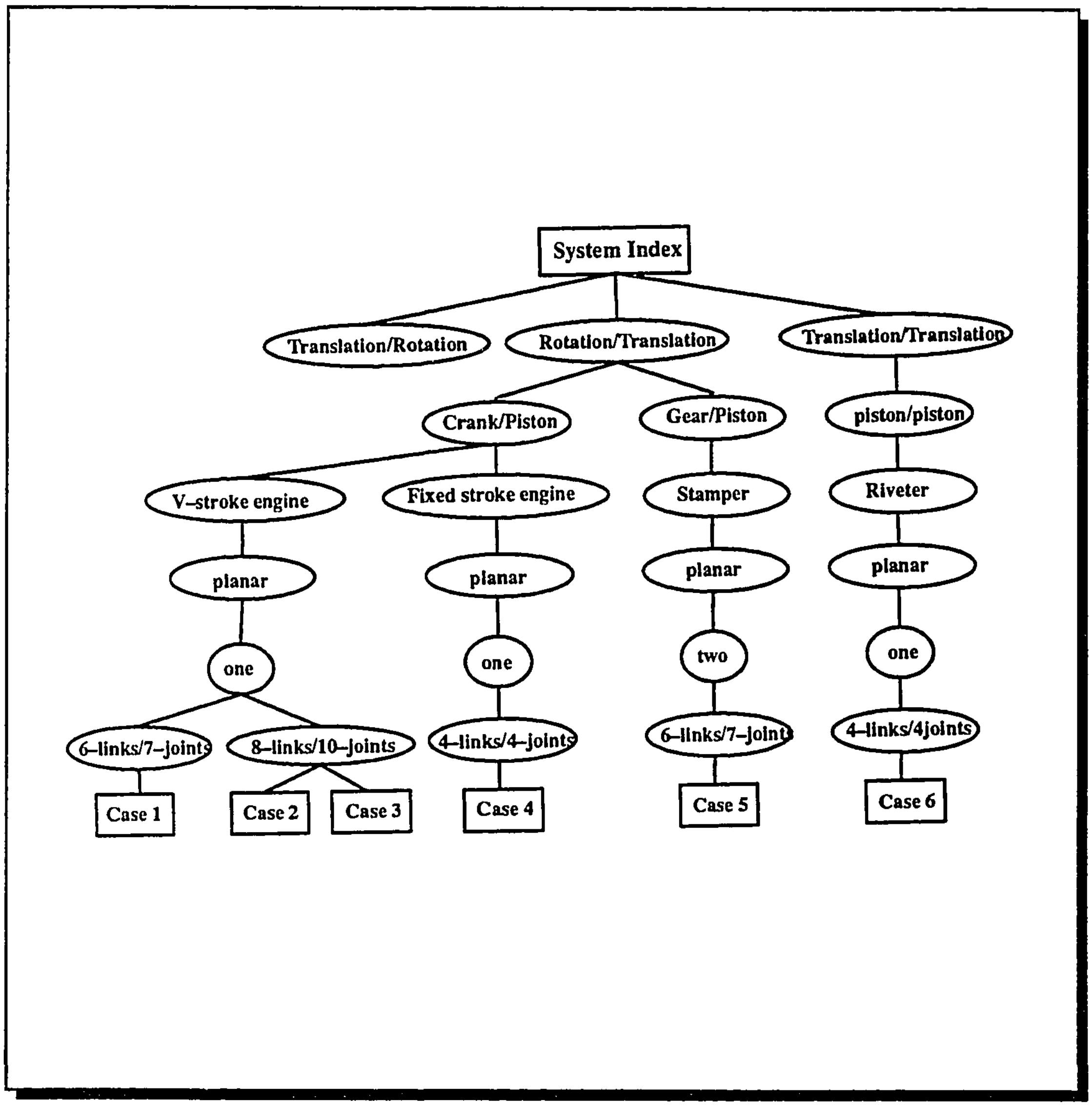


Figure 20. System Index Based on a Set of Discriminating Features

In summary, the search algorithm is guaranteed to retrieve an existing case if one exists. If more than one exists then it will retrieve the "best" matching case. The degree of similarity and consequently the correctness of the analogy transformation depends on the level at which the first mismatch is encountered.

As an example consider a target case with the following specification:

I/O :	Rotation/Translation
I/O Components:	Spring/Piston
Application:	V-stroke engine
Motion:	Planar
DOF:	1
Links/Joints:	7/9

Based on the above specifications the search begins from the top level with the I/O requirements. In this case the node with Rotation/Translation is assigned the highest Metric value, since it is a perfect match. This node will be selected for expansion with out the need to look ahead. Reaching the next level, we find that there are three nodes that will be assigned the same metric value since all of them match the output component. In such a case, the next level is checked and based on the comparison the results is added to the metric value. In this example, since the application domain is a V-stroke engine, that node will be selected. The search continues until reaching the level containing the links/joints. Since the target specifications at the level do not match either node (7/9 vs. 6/7 and 8/10), a heuristic function is used to find the most preferred node of the two. The heuristic function in this case prefers cases with larger number of links and joints. This means that the node with 8-links/10-joints is the one to be expanded.

Having done so, the system finds that there are more than one case. These two cases are then retrieved and the next step which we call the *elaboration* step is to select the case which mostly resemble the target case. Such a task is accomplished by using the designer's assistance. For, instance, preference may be given to a design case which contains a specific design component(s), or a certain structure. The results of the search procedure is one source case which is then goes through the transformation procedure to find a solution that satisfies the requirements of the target case.

7.1.2 Transformation Process

In SMARTs a solution for a target case is found by transforming the source case based on the differences between the target and source case specifications. The transformation of a source case takes place in a number of ways which include: adding new components or deleting existing components, replacing existing components with new ones, and reorganizing its kinematic structure based on some target domain rules.

The transformation procedure is illustrated in Figure 21 and is carried out with the following steps:

Step I. Replace unmatching Input/Output components of the source case. This situation occurs when the target case requires a different set of I/O components than those existing in the source case. Before replacing any of the components in the source case, the CBR must find a description for any new component. To do so, the CBR attempts to retrieve relevant information that describes such components from the HR. If the HR has no knowledge regarding such components, then the CBR calls the FPR. If no help is derived from the FPR, then the designer must supply the missing information. Once all the new components are described, the CBR confirms the replacement step and the result is a source case with a new set of input and/or output components.

For example, if the target case requires Gear/Piston for its I/O components while the source case contains a Crank/Piston, then the CBR must replace the Crank with a Gear. However, the CBR must describe the Gear component before proceeding with the replacement and before it proceeds with the transformation. If the CBR has never experienced a case containing a Gear, then it switches to the HR trying to extract any available rules that describe the Gear. If that fails, then it switches to the FPR, and finally to the user. In summary, the CBR cannot proceed with the transformation process until it has a full description of each of the components that exist in a design case.

Step II. If required, the next step is to reorganize the structure obtained from the source case. The reorganization takes place under the following two conditions. First condition is when the application domain of the target case is different from that of the source case, and the HR contains a rule-set for the target application domain. For example if the target case is a variable-storke engine while the source case is a yoke-riveter, and the HR contains a rule set that describes the variable-stroke engines. The second condition is when both the target and source cases belong to the same application domain, and the corresponding rule-set in the HR has been updated after the source case was first created.

The reorganization procedure under both conditions is similar, except that the former is executed during the transformation phase, while the latter is executed only if the rules in the HR have been modified. Under both conditions, the rules that exist in each component of the source case are replaced with rules that exist in the HR that have matching conditions. This is followed by adding any new rules from the HR that describe that specific component. After the replacement and/or addition of rules, the structure obtained from the source case is then modified to satisfy the new rules. This step ensures that the rules contained within a case are the most current version of rules.

The reorganization stage may also include the replacement or addition of new design components. Such actions depend on the new rules obtained from the HR. If one of the new rules explicitly states that an existing component must be connected to a specific component that does not exist in the source case. In order to satisfy such a rule the new component must be added to the source case in a way that satisfy this rule and any other related rule. Of course, the new component must be fully described by either the HR or the FPR.

Step III. Following the reorganization step, the CBR checks for any difference in the constraints. A constraint is a requirement imposed on either the input or the output of a mechanism to behave in a certain manner, such as, the means to vary the stroke of the piston in a Variable-Stroke Engine application. A constraint is usually satisfied by one or more components within the mechanism. If the target case has none or has a different constraint, then the components that support the constraint in the source case are marked for potential deletion in the next step. This step is used to eliminate components that serve no purpose in the target case. In the case where different constraints are required, the components in the source case will be replaced by different components that will satisfy the new constraint. If such components are not available in the system's component store, then the user will have to specify exactly what components should be added, if any. The replacement of constraint components must proceed the next step (Addition/Deletion step). This is due to the fact that the replacement of component is not always a one-to-one process. New components may require more or less joints, and some components may them selves consist of other sub-component such as the Gear component.

Step IV. The final step in the transformation phase is to check for any differences in the number of links and joints between the source and target case. If no difference exists, then the intermediate structure from the source case represents a possible solution to the target case, and the transformation procedure is then terminated. If the number of links in the source case is greater than that in the target case, then the CBR reduces the number of links to match the target requirements. This is accomplished by first eliminating those links that have been marked for deletion by the previous step, and then proceeding by searching for links that can be deleted based on the rules existing within each link. This procedure is repeated until the number of links and joints remaining match the target case requirements. If the resulting structure contains marked links that are not deleted, then these links may provide for an unnecessary constraint. In such a case the links will remain marked and the decision is then left to the designer to replace these links with others, if needed.

Should the number of links in the source case be smaller than that in the target case, the CBR accommodates this difference by adding more links to the source case. The addition of new links is a more difficult problem than the deletion process. This is due to the fact that the system has to determine the type and source of components to add, and where such components are to be added. This problem is solved partially by maintaining a list of design components, called the *Component Store*. This list contains the description of distinct components that have been previously used by the system. There is a Component Store for each design domain. During the addition procedure, the CBR uses the component store of the target domain, source domain, or a combination of the two. If neither the Component Store of the source domain nor the target domain solves the addition problem, then the CBR will request assistance from the designer. The selection of components from the Component Store depends on the degree of assistance provided by the designer. In the worst case, where no assistance is available on the types of components, the system uses a combination of the components available (usually this a small number), as well as the rules existing in the source case and in the components selected to perform the addition.

One of the important aspects of the transformation process is that it is made easier by including the design rules with each mechanical component of the mechanisms. These rules guide the CBR during the various forms and stages of the transformation phase by confirming the legality of any move before it takes place. During both the addition and deletion, the CBR starts by checking the components in the source case one at a time. If the process is deletion, then the CBR takes the first component and attempts to delete it. However, before confirming the deletion, the CBR checks the rules present in that component and the rules present in the adjacent components. The CBR also checks the purpose of this component. For example, a component which serves as an input, output, or a constraint should not be deleted. If a component has no rules to prevent it from being deleted, it will be removed and the CBR will then proceed searching for another component to delete. This process is repeated until the remaining components and accordingly the remaining joints match the requirements of the target case. The result of this process could be more than one design configuration based on the type of components that have been deleted. Naturally, the greater the number of components that exist in the source case, the more different design alternatives are produced by the deletion process. The addition of components is similar to the deletion process, however the CBR in the addition process searches for a point to add or insert components. Before any addition of any components the CBR also checks the rules existing in the components of the source case and the rules existing in the new components to be added.

```
Let $T$=Target Case; $S$=Source Case.
Compute difference between $T$ and $S$
case difference of
   I/O Components:
      Replace $S$.I/O with $T$.I/O
      Describe new I/O components
      Find applicable rules in HR, FPR, or User.
   Application Domain:
      If HR contains a rule-set for $T$domain then
        For each rule in $S$
         Find rules in $T$domain with matching conditions and replace
         Find new applicable rules in $T$domain and add to $S$
         Transform $S$ to satisfy the new rules.
   constraints:
      For each link in $S$ do
         If a link produces a constraint on I/O then
            Mark for possible deletion.
   Links:
      If ($T$.links < $S$.links) then
         $N$ = $S$.links - $T$.links
         repeat until (no more marked links) or ($N = 0$)
            Delete a marked link; $N = N - 1$
         If ($N = 0$) then a solution is found
         else
            Generate combinations of $N$ links in $S$ for deletion
            Test the deletion of each combination
            If no contradiction then Confirm deletion
      else If ($T$.links > $S$.links) then
         N = $T$.links - $S$.links
         Using $T$domain.comp_store, $S$domain.comp_store, User
            Build a component add_list
         Generate combinations of $N$ links in add_list for addition
         Test the addition of each combination
         If no contradiction then confirm addition
```

Figure 21. Transformation Process in CBR

7.1.3 Generalization, Indexing and Storage

The learning process in the CBR is a case-based learning, that is, it depends on learning new specific cases rather than a new general concept of the case. However, since the transformation process in the CBR may result in more than one solution, a simple form of generalization is used to reduce the number of specific cases. The learning process in the CBR consists of the following steps:

1) **Evaluation:** New design cases must be proven feasible before being stored in the CBR for later use. However, since the transformation procedure is controlled by the design rules present in each of the components of a mechanism, and since it keeps track of the various target-case specifications and any general rules in mechanisms available in the CBR, then any solution resulting from the transformation must at least satisfy the necessary requirements to be considered a feasible solution. Since the rules present in a design case are a subset of those that exist in the HR, then to further refine the feasibility of the various solutions, the designer has the option of evaluating the resulting designs by executing the rules in the HR. The final result would be one or more design cases ranked according to their degree of feasibility. The ranking of design cases is described in more detail in the next section.

2) **Generalization:** The generalization procedure is a simple form of generalization with the objective of reducing the number of new design cases to be stored in the CBR. This procedure consists of two steps. First, all the design solutions are compared and grouped into several sets based on the type of components, type of joints, and rules used in each design case. Second, from each solution set, only the one with the highest rank is selected for storage. This procedure substantially reduces the number of possible solutions since it eliminates design alternatives that are either structurally equivalent or readily deducible from another.

3) **Indexing and Storage:** The final stage of the learning process is to store feasible design solutions for later use by the system. The objective of this step is to index the new design cases properly so that they are readily accessible when needed. In SMARTs, what is not important is where a design case is stored in memory, but how to get to it. Generating a proper index for a design case is actually accomplished during the search and retrieval phase. The index for a target case will follow the same path as the source case until the first mismatch was encountered. In other words, the path of the target-case solution will branch off that of the source case at the point where the specifications of the target case mismatched those of the source case.

7.2 The Heuristic Reasoner (HR)

The HR is a rule-based system with its own knowledge-base and inference engine. It is designed to contain background knowledge in the form of design heuristics. Such background knowledge is used to aid the CBR during its training and transformation phases. This is accomplished by providing the CBR with access to the HR knowledge-base and permitting it to search for any applicable rules based on the components present in the design at hand. As mentioned earlier, the HR is also used as an evaluation tool to test the feasibility of mechanisms based on their kinematic structure and function if rules are available in the knowledge-base.

Knowledge in the HR is represented in the form of rules which are provided by experts in the mechanism field. The experts build this knowledge-base by entering application-oriented rules, based on experience. A typical rule consists of an IF part, a THEN part, a certainty factor, and an explanation. The format of the rules is designed to be compatible with the CBR which needs to understand them and be able to reason accordingly. An example of a simple rule that describes the piston component in the Variable-Stroke Engine application is illustrated below:

```
IF: ((COMP_NAME PISTON) (DEGREE BINARY))
THEN: (STRUCTURE LEGAL)
ELSE: (STRUCTURE ILLEGAL)
CERTAINTY: 0.8
EXPLANATION:The Piston must be binary. It must be connected
```

to only two other links. This requirement is to avoid
high side thrust and any excessive friction.

The Heuristic Reasoner is divided into several sets of rules each of which is oriented towards a specific design application. According to the application of the design at hand, the proper rule set is activated if it exists. Figure 22 shows the basic components that constitute the HR. The figure shows the various interfaces between the HR, CBR, FPR, and the user.

7.2.1 The HR Access

As presented earlier, the HR is a knowledge-based system located between the CBR, and FPR. This means that the HR must be equipped with communication links which ties it with other system's modules and the user. There a number of ways to access this module:

1) *Direct User Access*. Just like accessing any expert system shell, the HR has facilities to interact with the user. The user can access the HR to add rules, update rules, and as an evaluation tool. The HR is a complete expert system which can be accessed individually without going through the overall system's operation.

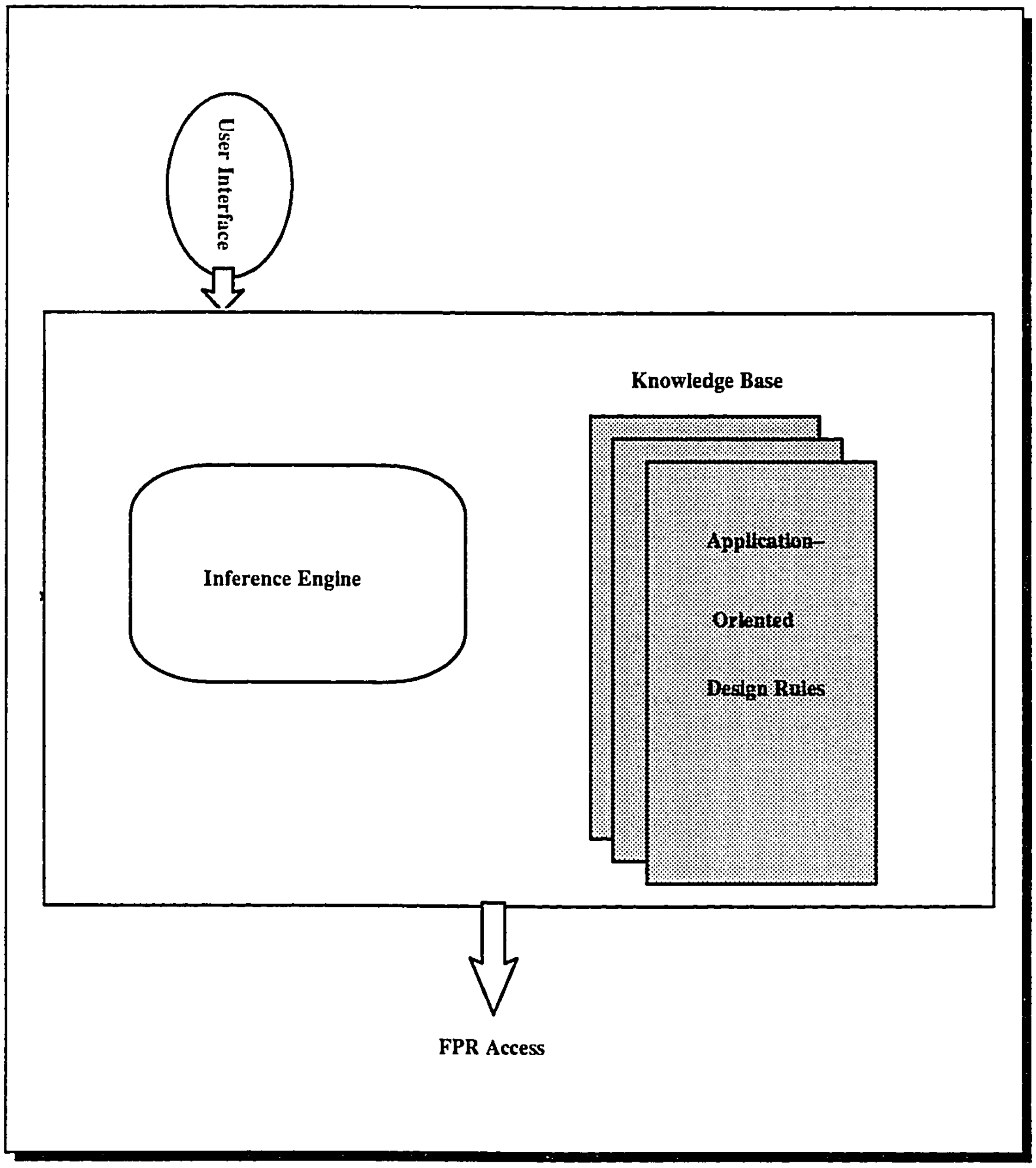


Figure 22. The Architecture of the Heuristic Reasoner

2) *Access through the CBR.* Knowledge in the HR is made accessible to the CBR whenever needed. During the training phase and the transformation procedure, the CBR sends a request to the HR through the Controller searching for rules that describe some component required for the construction of a given mechanism. Upon receiving such a request, the HR searches its knowledge base and sends any relevant rules to the CBR. Another form of indirect access to the HR occurs during the operation of the CBR (training and transformation). If the user decides that some rules for a given case should be modified or new rules should be added, then a copy of these rules will be placed in the HR.

3) *Access to the FPR.* Although request for information from the FPR is usually generated by the CBR, knowledge that are extracted from the FPR is formulated and also stored in the HR. The reason behind this duplication of knowledge is to reduce any future need to access the FPR. This becomes apparent in a situation where the FPR was accessed earlier, however the case which was solved based on that knowledge does not represent a source case for a new target case. In such situation, since the FPR information has been stored in the HR, then there is no need to switch again to the FPR.

7.2.2 Evaluation and Ranking Design Configurations

One of the important features of the HR is that it uses a certainty factor to rank the various design configurations. Such ranking is maintained during the execution of the rules. The system relies on the ranking of design configuration during its Generalization procedure to reduce the number of specific cases that are stored in the CBR memory. The calculation of the certainty factors is based on the Certainty Theory presented by Shortliffe and Buchanan in 1975 [73]. This theory was developed for use in expert systems. Given a factual statement S, the certainty measure is C(S). If S is known to be true, then C(S) would be equal to one. Each rule in the knowledge base is assigned a certainty factor CF which indicates the reliability of the rule. To calculate the certainty measure for a conclusion X in the presence of a condition A; C(X/A) the following equation is used:

$$C(X/A) = C(X) + [CF * [1.0 - C(X)]]$$

Based on the rules incorporated in our system we assume that all the conditions present in any rule are known to be true; that is $C(A) = 1$; while the conclusions are always set initially to a Zero value; $C(X) = 0$. Let's consider the following example with two rules that have been fired:

R1) If ((Link$_i$ = "Crank") and (Link$_i$ is directly connected to a sliding-pair))

Then REJECT-MECHANISM with CF:0.9

R2) IF ((Loop$_i$ = "Drive-loop") and (Loop$_i$ Contains a sliding-pair)

Then REJECT-MECHANISM with CF:0.6

Our goal is to determine the final Value for the Conclusion REJECT-MECHANISM assuming that it was initially set to Zero.

Let C(REJECT-MECHANISM) = 0.0

Let A denotes the condition in R1, and B denotes the condition in R2.

Let C(A) = 1.0, and C(B) = 1.0; Based on our assumption.

Using R1:

C(REJECT-MECHANISM/A) = 0.0 + [0.9 * [1.0 - 0.0]]

= 0.9

Using R2:

C(REJECT-MECHANISM/B) = 0.9 + [0.6 * [1.0 - 0.9]]

= 0.96

Hence the final certainty of REJECT-MECHANISM = 0.96

Knowledge in HR is dynamic and changes with repeated use of the system. The HR learns new rules in several ways. First, by a direct access from the designer. Second, during the training phase, new rules are added to the HR based on the application type. Finally, the HR learns new rules by accessing the FPR. Knowledge from FPR is transformed into rules and stored in the HR, and thus reducing the number of switches to the FPR.

7.3 The First-Principle Reasoner (FPR)

The incorporation of first-principle knowledge in knowledge-based systems is an important factor that has been neglected by many expert system builders. While many implementations have used this type of knowledge mixed with heuristics and without any distinction between the two types of knowledge and between the reasoning strategies necessary to make a use of this knowledge. We believe that first-principle knowledge should be explicitly differentiated than other types of knowledge in a given system.

The First Principle Reasoner is located at the lowest level of SMARTs, and is used as a last resort in the problem solving process. It is called when the transformation process is interrupted by the presence of mechanical components that are neither described by the CBR nor by the HR. The FPR is a database consisting mainly of basic design knowledge regarding several mechanical components that are used as building blocks for many mechanical systems. This knowledge is fundamental to most mechanisms and therefore does not depend on any specific design application. Knowledge in the FPR is entered either by domain experts prior to the operation of the SMARTs or by the user during the reasoning process when more information is required. The duty of the FPR is then to complete the missing information and pass it up to the CBR so that it can proceed with its solving process. Information needed by the CBR is derived from the FPR according to the type or functionality of the unknown components. This information is then translated into rules compatible to both the HR and the CBR.

7.3.1 The Architecture of the FPR

Currently, the FPR consists of three tables describing the functionality, structure, and constraints of a list of mechanical components. Figure 23 shows the three tables with data representing a Gear component.

The Function Table

The function of a component may be described by the type of motion that is provided by the given component. Different components have different functions such as Rotation or Transnational motion. Examples on some components that support transnational functionality include: Pistons, Sliders, Gear Racks, and Springs. Others that support rotational functionality include: Gears, Rollers, and Cranks.

The function table consists of a list of different components indexed by the name and also indexed by the function. This permits the CBR or HR to retrieve a specific component or a list of components that provide the required functionality.

The Structure Table

The structure of a component describes the connectivity of the component (in terms of other subcomponents). The structure table contains important information required to connect a component to a design case. Many mechanical components consists of various subcomponents connected by certain joints. Such connectivity must be provided to the CBR if it is to add a component consisting of a number of subcomponents to an existing design case. Based on this table the CBR determines the number of subcomponents, the number of joints, the degrees of freedom for each joint, and accordingly constructs a subgraph representing such information. This subgraph which could be a mechanism on its own rights, is later connected to the design case in hand.

The Constraints Table

Constraints are requirements that have to be satisfied when a component is to be used in a mechanism (note that this constraint differs from that presented earlier in the design representation which served as a constraint on the input or output). Without the information present in the constraint table, the CBR may have to make a guess on how to connect a retrieved component to an existing system. Constraints are rules established based on experience in using a given component in the construction of mechanisms. While such knowledge is not exactly first principle knowledge; more heuristics than first principle, it is essential to eliminate the element of trial and error of connecting a component to a mechanical system. When a component is retrieved from the database, its constraints are converted to a form that matches that of the HR and CBR.

A request for information is passed to the FPR either in the form of a component name or a required functionality for which a component description is to be supplied. Figure 23 is shows a representation of a Gear component. The rows in the structural table represent the connectivity of the internal subcomponents within the Gear.

As an example, the first row in the table shows that a Gear consists of two Gear Plates (Gearl and Gear2) connected together by a G-joint which has two degrees of freedom. When this table is accessed all the rows concerning the component "Gear" are read and used to connect the component to a new mechanism. Information in the constraints table are also read and translated into rules compatible with the HR rules and are used to monitor the connection of the component to the rest of the system.

7.3.2 Accessing the FPR

One of the important characteristics of our system is that it is an open system. The knowledged existing in the system is dynamic and changes when one module accesses another or by the user who is given complete access to each module. Each module is constructed to behave as a complete system and as a part of an overall system. This means that a user interface and subsequently a complete access to each module becomes an important factor that must be considered. A good example to this consensus is the access provided by the FPR. The FPR has access facilities to the remaining two modules and to the user.

1. *Direct User Access.* As a database system, the FPR provides the user with many features for readily data access. Such capabilities include: adding new components to the FPR, updating existing components, deleting components, searching for a specific component, and listing the various components. In addition to these basic functions, a user can use the facilities of the FPR to generate various reports which provides the user with better understanding to the contents of the database.

Linkname	Function	Description
Gear	Rotation	The gear is a component consisting of two gear plates connected by a gear-pair joint.

(a)

Linkname	**Link1**	**Link2**	**Joint**	**DOF**
Gear	Gear1	Gear2	G	2
Gear	Gear1	G-Arm	R	1
Gear	Gear2	Ground	R	1
Gear	Gear2	G-Arm	R	1
Gear	G-Arm	Ground	R	1

(b)

Linkname	**Fact**	**Certainty**
Gear	Must not be concentric	0.8
Gear	Must be connected to Ground	0.9

(c)

Figure 23. Representation of a Gear component in FPR:
(a) Functional Table; (b) Structural Table; (c) Constraints Table

2. *Access to the CBR and HR.* The duty of the FPR is to pass information to both the CBR and HR. Therefore, the FPR has no access to the upper modules except for passing information. As mentioned earlier, knowledge retrieved from the FPR must be processed first before it can be useful. Constraints are processed as rules compatible to those in HR and CBR, and then is transferred upwards. The structure of a component is processed into a subgraph to serve as part of a complete mechanism.

CHAPTER EIGHT

Application of SMARTs to Mechanisms Synthesis

Highlights

- Training Example
- Design Examples

Application of SMARTs to Mechanisms Synthesis

Chapter Outline

This chapter presents a number of practical examples as applied to our system. The examples are chosen to demonstrate some of the different activities that take place in SMARTs during the synthesis of mechanisms based on existing design cases. The examples also show how this system incorporates its background knowledge to solve problems which otherwise will be difficult to solve by the Case-based Reasoner alone. These examples are by no means cover all the possible situations that may arise in the system, however, they are real examples extracted from a number of references and in some cases represent actual U.S. patents.

8.1 Training Example

Assuming that the CBR knowledge-base contains no previous cases, or that the existing design cases are too different from the given target case. Then the first step for the designer is to begin training the system.

Using the design representation discussed earlier, a typical training example includes various design specifications, connectivity graph describing the structure of a mechanism, and a description of each design component including rules that govern the structure and connectivity of each component. The system aids the user in building the training example by providing any existing information regarding the application domain of the training example. One such help is to search the HR knowledge-base for any applicable rules. If such rules do not exist the system may switch to the FPR or may request the information directly from the trainer. The system also checks the entries supplied by the trainer by enforcing general rules such as the degree-of-freedom equation. Some portions of a training example for an Eight-link Variable-Stroke Engine is shown in Figure 24. Once the trainer enters the connectivity of the mechanism as in Step 6 of Fig. 24, the system constructs the connectivity graph representing the mechanism. In this step, SMARTs describes the degree of each component (the number of joints on the component, eg. Binary) and shows the connectivity of the component. The trainer however must enter the type of joints to be used to connect this component. Following that, SMARTs searches the Heuristic Reasoner for any applicable rules. The rules will be checked against the components and their connectivity. If any contradiction

1- I/O Requirements
Input: ROTATION
Output: TRANSLATION
InputComp: CRANK
OutputComp: PISTON

2- Application: Variable-Stroke Engine

3- Constraints: CONTROL OUTPUT

4- Structural Specifications:
Motion: PLANAR
Degree Of Freedom: 1
Links: 8
Joints: 10

5- Connectivity:
Connects(PISTON, GROUND, P-ROD)
Connects(P-ROD, SLOT)
Connects(SLOT, ROCKER, SLIDER)
Connects(ROCKER, COUPLER, GROUND)
Connects(SLIDER, GROUND)
Connects(COUPLER, CRANK)
Connects(CRANK, GROUND)

6- Components Description*:
Describing the PISTON:
BINARY(PISTON)
CONNECTED_TO(GROUND) **BY P**
PURPOSE(OUTPUT)
FUNCTION(Trans)
RULES:
```
1. IF ((COMP_NAME PISTON)
     (DEGREE BINARY))
     THEN: (STRUCTURE LEGAL)
     ELSE: (STRUCTURE ILLEGAL)
     CERTAINTY: 0.8
   EXPLANATION: To avoid high piston
                side thrust.
```

7- Feasibility of Mechanism:
STRUCTURE(LEGAL)
FEASIBILITY DEGREE(0.9)
VIOLATED RULES: NONE

* Only the description of one component is shown here

Figure 24. *Training Example in SMARTs*

exists, the trainer is notified. The trainer can then add to or change these rules or change parts of the training example. In the final step of the training phase, the system shows the feasibility of the resulting design and lists any violated rules. If the trainer agrees to this design, it is indexed and stored in the knowledge-base of the CBR for later use.

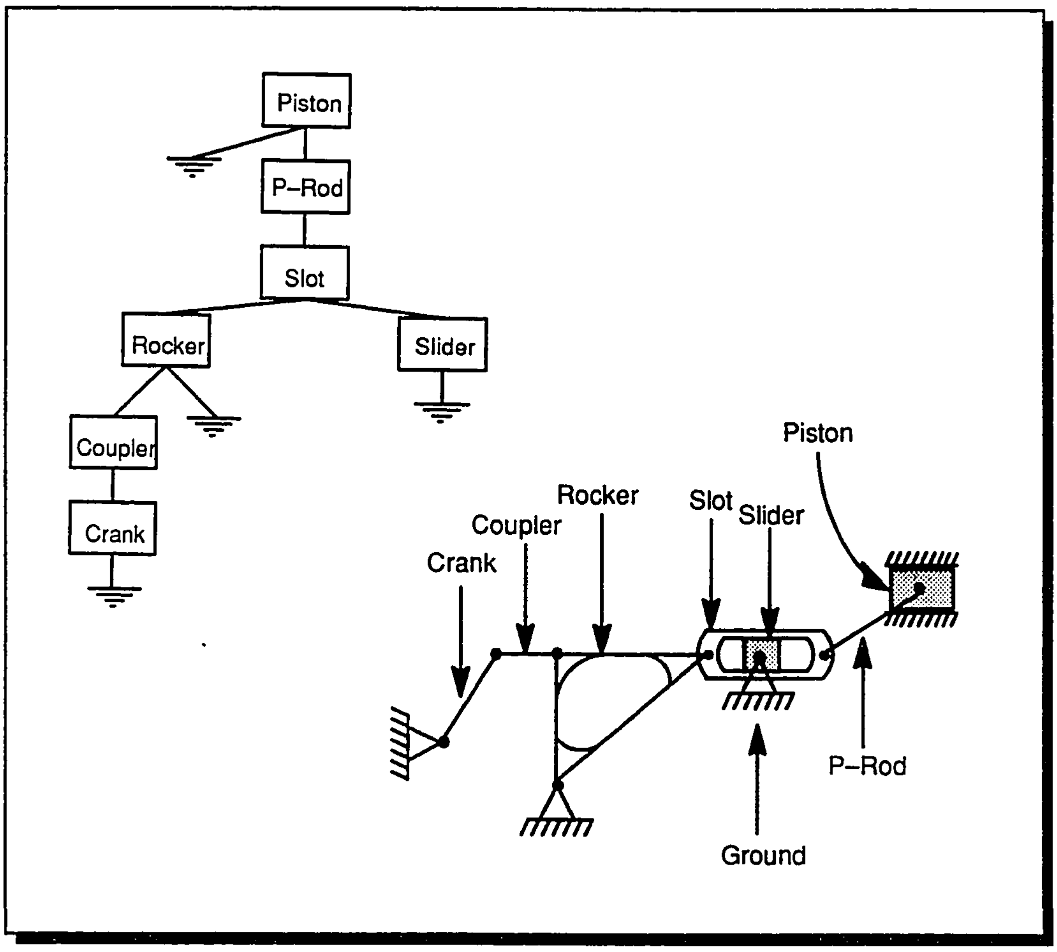


Figure 25. Graph representation and Schematic Diagram for Training Example

8.2 Design Case 1

Application: Design a Six-link Variable Stroke Engine

Target Case Specification:

I/O Requirements: Rotation/Translation
Input Component: Crank
Output Component: Piston
Application: Variable-Stroke Engine
Constraints: Control Output
Motion: Planar
Degree of Freedom: 1
Links: 6
Joints: 7.

Source Case: Eight-link variable stroke engine shown in Figure 25.

Objective: The objective of this example is to demonstrate how the training example presented earlier is used to produce a new design for a target case. In this example we present one of the simplest transformation scenarios. This is due to the fact that the only difference between the target case and source case is the number of links and number of joints.

Assumptions: The training example is the only existing case in the CBR.

Solution: Based on the difference between the target and source cases, the CBR concludes that it must reduce the number of links by two and the number of joints by 3.

Such deletion must be accomplished while maintaining the requirements of the target case and without violating any rules. The CBR begins this process by examining each component in the source case while testing the purpose, function, and rules of each component. The reasoning process based on the components of the source case can be described in the following steps. Only those rules that show immediate effect are included in this example. The rules are usually more explanatory than they appear in the example.

1) Test the PISTON:
Purpose: Output
Rules: Must be binary; must be connected to Piston-Rod (P-Rod)
Action: Do not delete.
Reason: The piston satisfies the Output requirement in the target case.

2) Test the P-Rod:
Purpose: Support Piston
Rules: Must be binary; Must be connected to Piston; Should not be connected to the Crank.
Action: Do not delete.
Reason: The P-Rod cannot be separated from the Piston.

3) Test the Slot:
Purpose: Control Output
Rules: Must be connected to a slider
Action: Do not delete
Reason: The Slot has the purpose of controlling the output; a constraint present in the target case.

4) Test the Slider:
Purpose: Control Output
Rules: Must be connected to either Ground or a slot
Action: Do not delete
Reason: Slider is also used to satisfy the required constraint

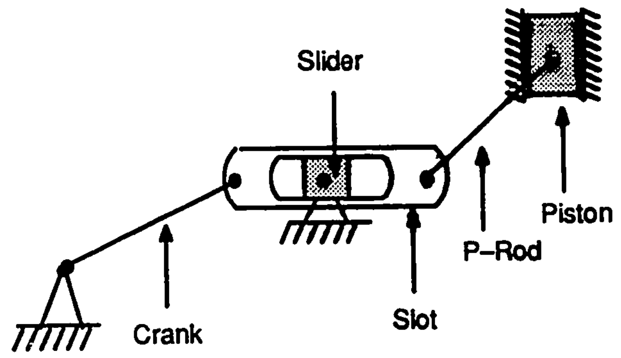
5) Test the Rocker:
Purpose: Support
Rules: Must be ternary and connected to Ground

Action: Delete
Reason: The Rocker in this case does not directly affect the target case specifications. There are no rules present to prevent deletion.
Note: Deleting the Rocker results in deleting one connection to Ground.

6) Test the Coupler:
Purpose: Support the Crank
Rules: Must be binary
Action: Delete
Reason: The Coupler in this case does not directly affect the target case specifications. There are no rules to prevent deletion.

7) Test the Crank:
Purpose: Input
Rules: Must be binary
Action: Do not Delete
Reason: Matches the required input in the target case.

Figure 26 shows the result of these steps. Having deleted the required number of links, the next step is to reconnect the mechanism. The reconnection is accomplished almost in the same way as the deletion. The CBR attempts to connect components together while checking their rules and the target specifications. For example, If the CBR attempts to connect the slider to the Crank it will find that there is a rule which prevents such connection from taking place. This rule is present within the Crank it self. The CBR then tries to connect the Crank to the Slot. Such connection is feasible since there are no rules present in either the Crank or the Slot to prevent such connection. The CBR also keeps track of the number of joints to ensure that they are reduced to seven joints as required by the target case specifications. The remaining number of links and joints, and the resulting degree of freedom of the mechanism must also match the required specifications.



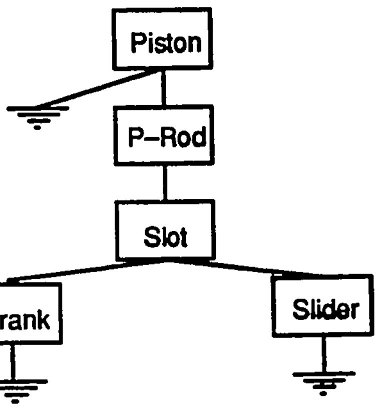


Figure 26. Resulting design for the Six-link variable stroke engine (Case 1)

8.3 Design Case 2

Application: A Six-link Assembly Line Stamper

Design Specifications for Target Case:

I/O Requirements: Rotation/Translation
Input Component: Gear
Output Component: Piston
Application: Stamper
Constraints: None
Motion: Planar
Degree of Freedom: 1
Links: 6
Joints: 7.

Source case: Eight-link variable stroke engine shown in Figure 25.

Objective: This example shows a problem requiring the design of an assembly line box-stamper. This is a mechanism that is used in factories to stamp the name of the manufacturer or other information on boxes that are moving along an assembly line. The designer's goal is to arrive to a conceptual design of the required mechanism. This example demonstrate the use of the FPR to provide needed information.

Assumptions: As an illustration, we will assume that the system's index in SMARTs contains only the training example described earlier for a Variable-Stroke Engine (Figure 25). Assume also that the HR has no rule set describing the Stamper application.

Given the design requirements of the Stamper shown above, SMARTs searches its index and finds that the only existing and therefore most analogous source case is the training example for the eight-link variable stroke engine. Having found a source case, the next step is to compute the difference between target case and the source case. The difference is as follows:

1. Difference in the input components. Source case has a Crank while target case has a Gear.
2. Application domain. Source case is a variable-stroke engine while target case is a stamper.
3. Source case has a constraint on input (Control Output). Target case has none.
4. Source case consists of 8 links (components) and 10 joints while target case requires 6 links and 7 joints.

To perform the transformation process, the CBR concludes, based on the above differences, that it must:

1. Replace the input component of the source case (Crank) with a Gear.
2. Delete two links and three joints from the source case to satisfy the target requirements.
3. Since the target case has no constraints on I/O, begin the deletion of the control links first.

However, before the CBR begins any transformation it must acquire information about gears. It does that by switching to the HR searching for any rules on Gears under the Stamper rule set. If this rule set does not exist or has no rules describing a Gear, then the search continues under the Variable-Stroke Engine rule set (source case domain). Any applicable rules that are found will be transferred to the CBR and will be used to complete the transformation process. In this case we assume that no rules are found and therefore, the system must switch to the FPR attempting to define the unknown component "Gear". If this process is successful, the required information is then transformed into rules that are passed to the CBR. According to the database tables shown previously, the CBR concludes that:

1. A Gear consists of three links, Gear1, Gear2, and a G-Arm.
2. The two gears are connected by a G-joint with two degrees of freedom.
3. The gears are also connected by the G-Arm through Revolute joints.
4. One of the gears must be connected to Ground.

Based on the Gear description above, the CBR concludes that it must delete three additional links and three joints from the source case to accommodate for the Gear component. This results in a total of five links and six joints that must be deleted from the source case to be transformed to the target case. However before the CBR deletes any links, it will check the purpose of each component and conform with the rules that exist within each component. It will also keep track of the target specifications. In this case, the CBR detects that using the Gear component will result in a mechanism with two degrees of freedom, which is due to the presence of the G-joint. Therefore, before any actual transformation takes place the user must be notified with this contradiction, and is given the choice to continue or change other parts of the specifications that would eliminate this contradiction. Referring to the connectivity graph in Figure 25, the flow of reasoning of this step is shown below:

1) Test the PISTON:
Purpose: Output
Rules: Must be binary; must be connected to Piston-Rod (P-Rod)
Action: Do not delete.
Reason: The piston satisfies the Output requirement in the target case.

2) Test the P-Rod:
Purpose: Support Piston
Rules: Must be binary; Must be connected to Piston; Should not be connected to the Crank.
Action: Do not delete.
Reason: The P-Rod cannot be separated from the Piston.

3) Test the Slot:
Purpose: Control Output
Rules: Must be connected to a slider
Action: Delete
Reason: There is no constraint present in the target case.

4) Test the Slider:
Purpose: Control Output
Rules: Must be connected to either Ground or a slot
Action: Delete
Reason: Slider is also used as part of the constraint.

5) Test the Rocker:
Purpose: Support
Rules: Must be ternary and connected to Ground
Action: Delete
Reason: The Rocker in this case does not directly affect the target case specifications. There are no rules present to prevent deletion.
Note: Deleting the Rocker results in deleting one connection to Ground.

6) Test the Coupler:
   Purpose: Support the Crank
   Rules: Must be binary
   Action: Delete
   Reason: The Coupler in this case does not directly affect the target case specifications. There are no rules to prevent deletion.

7) Test the Crank:
   Purpose: Input
   Rules: Must be binary
   Action: Replaced by the Gear
   Reason: Based on the input component for the target case.

In summary, the CBR does not delete the piston because it matches the output component of the target case. The piston must be connected to a P-Rod according to the rules of both Piston and P-rod, therefore the P-Rod will not be deleted. On the other hand the Slot and Slider are deleted because they serve as Output Control; a constraint not found in the target case. The Rocker and Coupler are also deleted because they contain no rules that would prevent this deletion. Finally the Crank must be replaced with the Gear to satisfy the input requirement of the target case. The final decision taking by the CBR is to determine how to connect the Gear to the remaining components. This process is similar to the deletion process above; the CBR must still conform to the existing rules within each component to be connected with the Gear. If no rules would be violated it will be considered as a legal design configuration. A feasible design configuration that results from this example is shown in Figure 27.

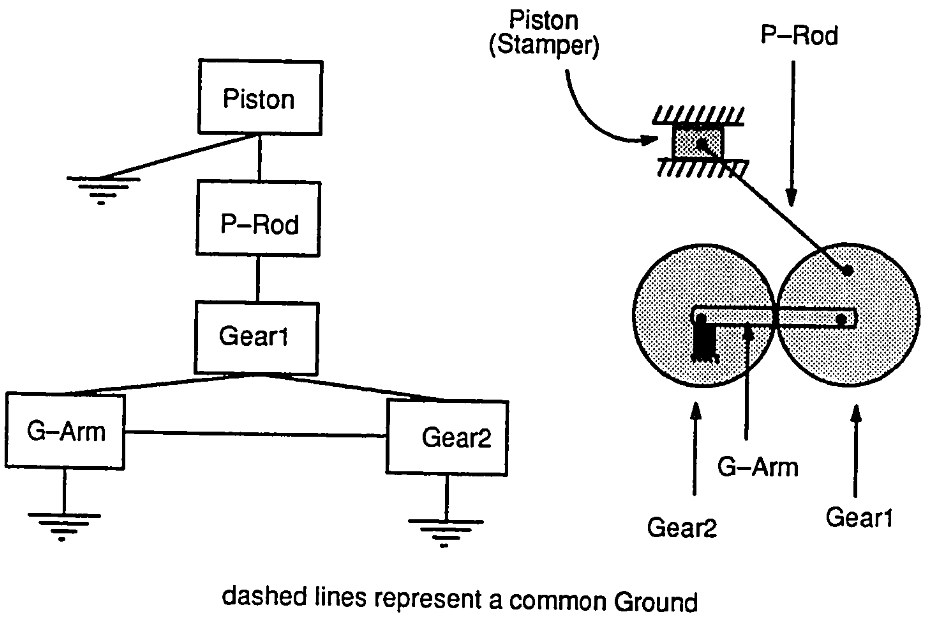


Figure 27. A connectivity graph and schematic diagram for the Stamper design case

8.4 Design Case 3

Application: An eight-link Variable Stroke Engine.

Design Specifications for Target Case:

I/O Requirements: Rotation/Translation
Input Component: Crank
Output Component: Piston
Application: Variable-Stroke Engine
Constraints: Control Output
Motion: Planar
Degree of Freedom: 1
Links: 8
Joints: 10.

Source Case: Six-link variable stroke engine design shown in Figure 26.

Objective: To show how the System transforms a six-link source case to a solution for a target case requiring larger number of links and joints. This transformation requires invoking the Addition Procedure, and makes a use of the Component Store. Note that both source and target cases belong to the same application domain.

Assumptions: Assume that the CBR contains Design Case #1 and the Source case. Assume also that no assistance is provided by the designer on what components to add.

Solution: Based on the specifications of both the target and source cases, the CBR concludes that it must increase the number of links by two, and the number of joints by three while maintaining other specifications. The addition of new components usually take place to produce more stable mechanisms. Although this means that the resulting mechanism is more complicated and more expensive. For example, the addition of a Rocker component to a variable stroke engine shows to provide a more stable mechanism since it isolates the Crank from the rest of the system and thus absorb excessive vibrations which may affect the piston.

Using the Component Store

Since our assumption above states that no assistance is being provided by the user on what components to add, SMARTs relies on its Component Store for the selection of new components. The Component Store includes the description of various components that have been used within the variable-stroke engine application. Since we assumed that the system contains two different six-link designs, the component store will include the following components:
Piston, P-rod, Slot, Slider, Rocker, Coupler, Crank.

Using a Selection Heuristic and Constructing an Add List

The selection of these components is first based on a heuristic which prefers the selection of components that are not currently present in the source case. This heuristic based on the fact that in most of our designs it is very rarely to see repeated components within the same mechanism. When we design a variable stroke engine, we are actually designing a prototype with one Piston only instead of six or eight pistons. The selection is then based on the function and purpose of each of the components in the Component Store. Based on the addition heuristic, the CBR constructs a potential Add List containing the following components: Rocker, Coupler.

The remaining components have not been selected since they are already present in the source case. The next step is to check the purpose and function of each of these components and then decide where to add them to the source case. The system takes each component separately and attempts to connect it into various locations within the source case. At the same time the system will check the rules present in the new components and those present in the existing components. It also keeps track of the design specifications such as the number of joints and degree of freedom. The reasoning process for this addition using the source case is illustrated as follows:

Reasoning Process for Addition of New Components

1. Add to the Piston:
   Rules: Piston must be binary.
   Action: Do not Add at this position.
   Reasons: Violates the piston rules.

   Insert between Piston and P-Rod:
   Rules: Piston must be directly connected to P-Rod.
   Action: Do not Insert
   Reasons: Violates Piston and P-Rod Rules

2. Add to the P-Rod:
   Rules: P-Rod Must be binary.
   Action: Do not add.
   Reasons: Violates P-Rod Rules.

   Insert between P-Rod and Slot:
   Rules: No rules prevent Insertion
   Action: Insert Rocker, Connect Rocker to Ground based on Rocker rules.
   Reasons: No rules to prevent insertion.

3. Add Coupler to Slot:
   Rules: Coupler must be connected only to a component with Purpose = Input.
   Action: Do not Add.
   Reasons: Coupler is a device that couples the input to the output of a mechanism.

4. Insert Coupler between Slot and Crank:
   Rules: No rules to prevent insertion.
   Action: Insert Coupler
   Reasons: Insertion results in a configuration which matches the purpose of the coupler.

5. Insert Coupler between Slot and Slider:
   Rules: Slot must be connected to a Slider. Coupler must be connected to a component with Purpose = Input.
   Action: Do not Insert.
   Reasons: Violation of Slot rules and Coupler rules.

The above steps show the procedure by which the CBR reasons about adding new components to an existing case attempting to produce a solution to the target case. From these steps, it is clear that there are other alternatives that could be produced. These alternatives are a result of testing the addition of the Rocker to other components and then connecting the coupler based on that. Figure 28 shows the results of the steps above and some other additional configurations .

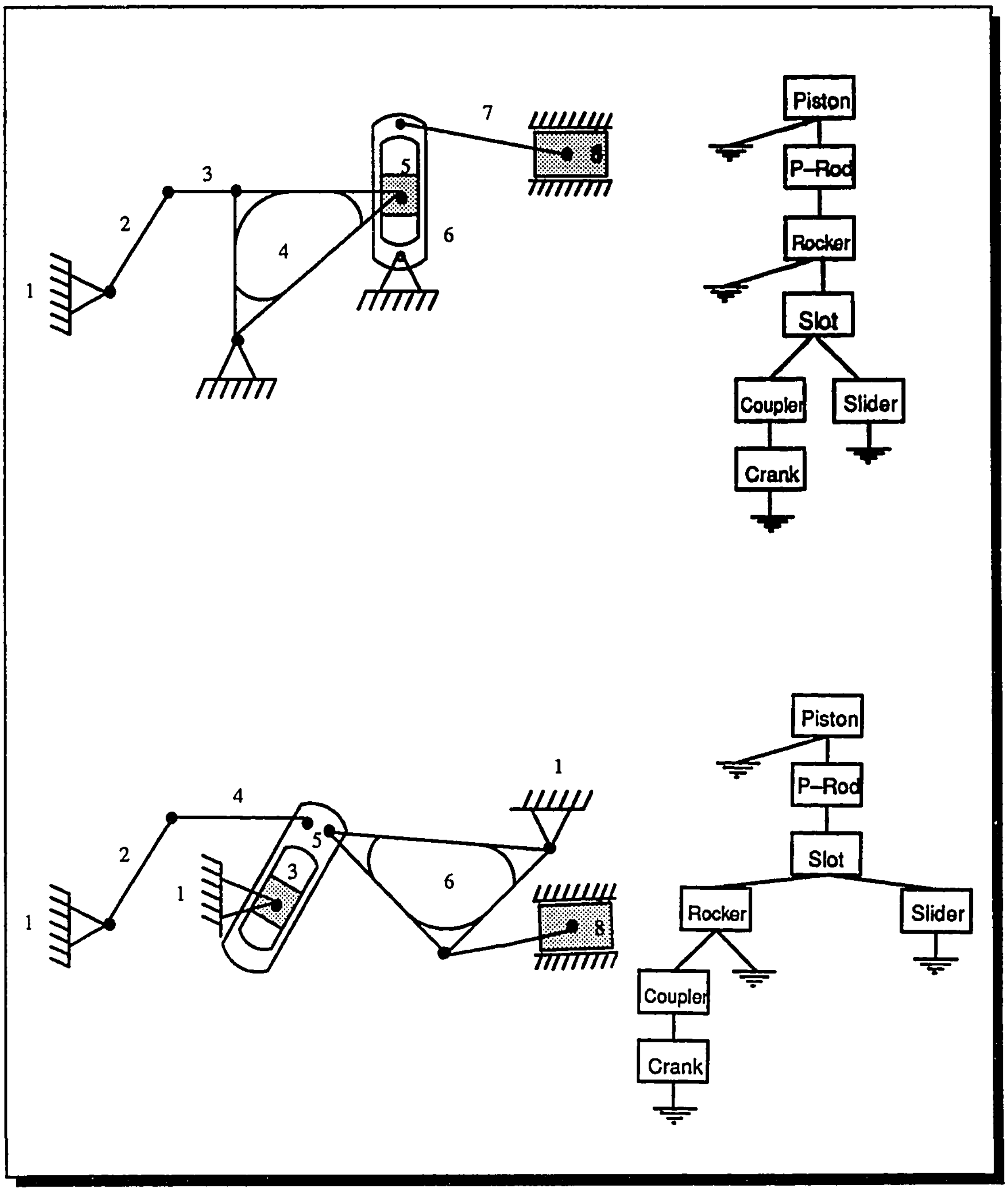


Figure 28. Results of transforming six-link variable stroke engine to an eight-link mechanism.

8.5 Design case 4

Application: An eight-link Yoke Riveter

Background: A yoke riveter is a common mechanism used in industry to attach sheets of metal using riveters instead of soldering. A typical Yoke riveter consists of a large size piston; called power piston, and a smaller piston called the Hammer. Moving the power piston in and out forces the hammer to move up and down and thus pressing a piece of aluminum rivet. As expected in such a mechanism, there must be a high force amplification between the power piston and the rivet die. Such power amplification must be considered in the design.

Design Specifications for Target Case:

I/O Requirements: Translation/Translation
Input Component: Piston
Output Component: Piston
Application: Yoke Riveter
Constraints: None
Motion: Planar
Degree of Freedom: 1
Links: 6
Joints: 7.

Source Case: Eight link variable-stroke engine. Figure 25.

Objective: Interaction of designer with system during the transformation phase.

Assumptions: HR contains a rule set describing the design of Yoke Riveters.

Solution: The CBR first determines the difference between the target and source case specifications. The difference includes:

1. I/O requirements: Source case requires Translation/Rotation while target case requires Translation/Translation.
2. Input component: Source case requires a Crank while target case requires a Piston.
3. Application: Variable stroke engine vs. Yoke riveter.
4. Constraints: Source case has a constraint while target case has none.

based on the above differences the CBR proceeds with the transformation procedure as follows:

1. Replace Crank with a Piston.
2. Reorganize the source case based on rules in HR.
    a. Connect output piston to ground.
    b. Replace the Coupler with a P-Rod. (based on Piston rules.)
3. Check constraints. Mark slot and slider for deletion.
4. Check number of links and joints. Delete marked links and connect the P-rod of the Power Piston to the rocker.
5. Check the resulting number of links, joints, and degree of freedom.

The Resulting mechanism is shown in Figure 29. As you can see from this example, the reorganization step calls for the restructure of the mechanism based on existing rules. In this case the restructuring was limited to the addition of a P-Rod for the second piston. In a more complex situation where more rules are present, this step may result in a dramatic change to the original structure.

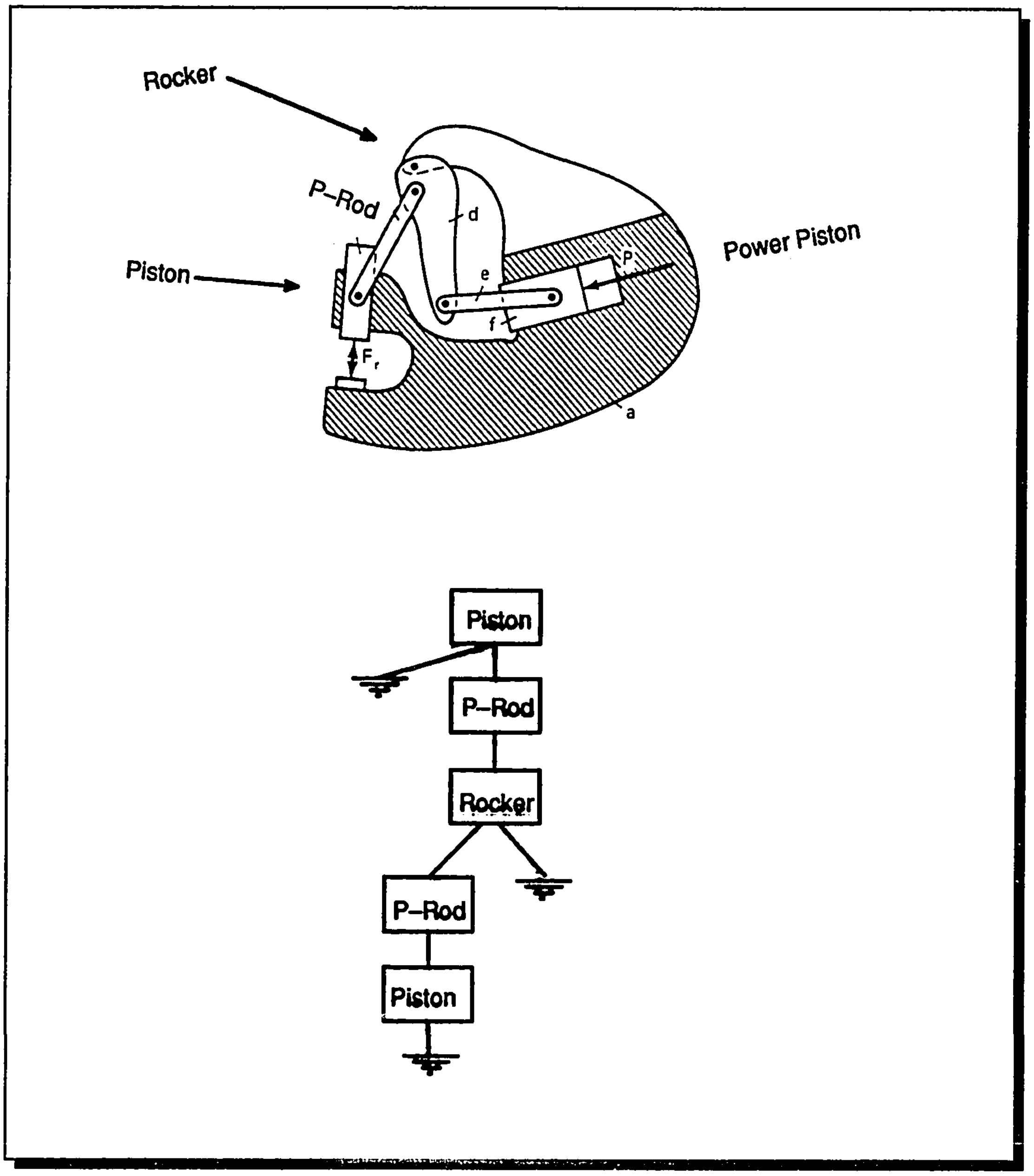


Figure 29. The resulting design of a Yoke Riveter

8.6 Design Case 5.

Application: A Five-link Internal Mechanism for a Two-Fingered Robot Hand.

Background: Robots that are used in manufacturing must be equipped with efficient mechanical hands which are able to manipulate different types of objects. In recent years, attention has been focused toward the development of more sophisticated mechanical hands to reduce the amount motion required by the robot arm. This design case illustrates the how our system can synthesize the internal mechanism needed to support a two-fingered mechanical hand.

Design Specifications for Target Case:

I/O Requirements: Rotation/Translation
Input Component: Gear
Output Component: Gear Rack
Application: Robot Hand
Constraints: None
Motion: Planar
Degree of Freedom: 2
Links: 5
Joints: 7.

Source Case: Six-link Stamper shown in Figure 27.

Assumptions: The HR Contains a rule set describing the design of Robot Hands.

Objective: To show how the system solves a target case using an existing case with a different application domain. To show the need for the reorganization stage in the transformation process.

Solution : Using the source case for the Stamper, the CBR first determines the difference between the target case specifications and those for the source case. The differences are as follows:

1. Output components do not match. Source case has a Piston while target case requires a Gear Rack.

2. Application domain. Source case is a Stamper while target case is a Robot Hand design.
3. Source case has a constraint on the output (Control Output). Target case has no constraints.
4. Source case consists of 6-links and 7-joints while target requires 5-links and 6-joints.
5. Source case requires a one degree of freedom while target case requires two degrees of freedom.

Having determined the above differences, the CBR begins the transformation process by:

1. Replacing the Piston with a Gear Rack.
2. Delete one link and one joint from the source case.
3. Attempts to eliminate the links that contribute to the constraint in the source case.
4. Keeps track of the degree of freedom and make sure it will equal to two degrees.

Before any transformation takes place the CBR must first find a description for the new component "Gear Rack". It accomplishes that by switching to the HR looking for rules describing such a component.

Some the rules describing the Gear Rack are listed below:

R1. Gear racks are not permitted for hand designs with more than five components.

R2. Only one Gear Rack is permitted for five-link designs.

R3. A Gear Rack cannot be connected to a revolute joint.

R4. A gear rack should be connected to one of the Gear Plates by a Gear-pair joint.

R5. Gear rack must be supported vertically by the extending the Gear Arm while permitting transnational functionality. This means that the Gear Rack must be connected to the Gear Arm by a sliding pair.

Using these rules the CBR begins the transformation as follows:

1. Delete the Piston component from the source case.
2. Delete the Ground link which was connected to the piston.
3. Delete the P-Rod since one of its rules requires that a P-Rod must be connected to the Piston.
4. Connect the Gear Rack:

   4.1 Connect the Gear rack to Gear1 by a G-joint

   4.2 Connect the Gear Rack to the to the Gear Arm by a sliding pair.

The results of these steps is a mechanism that corresponds to the requirements. The resulting mechanism is illustrated in Figure 30.

Note in this example that since the Gear has been previously used in the source case the CBR need not switch to the other reasoner looking for a Gear description.

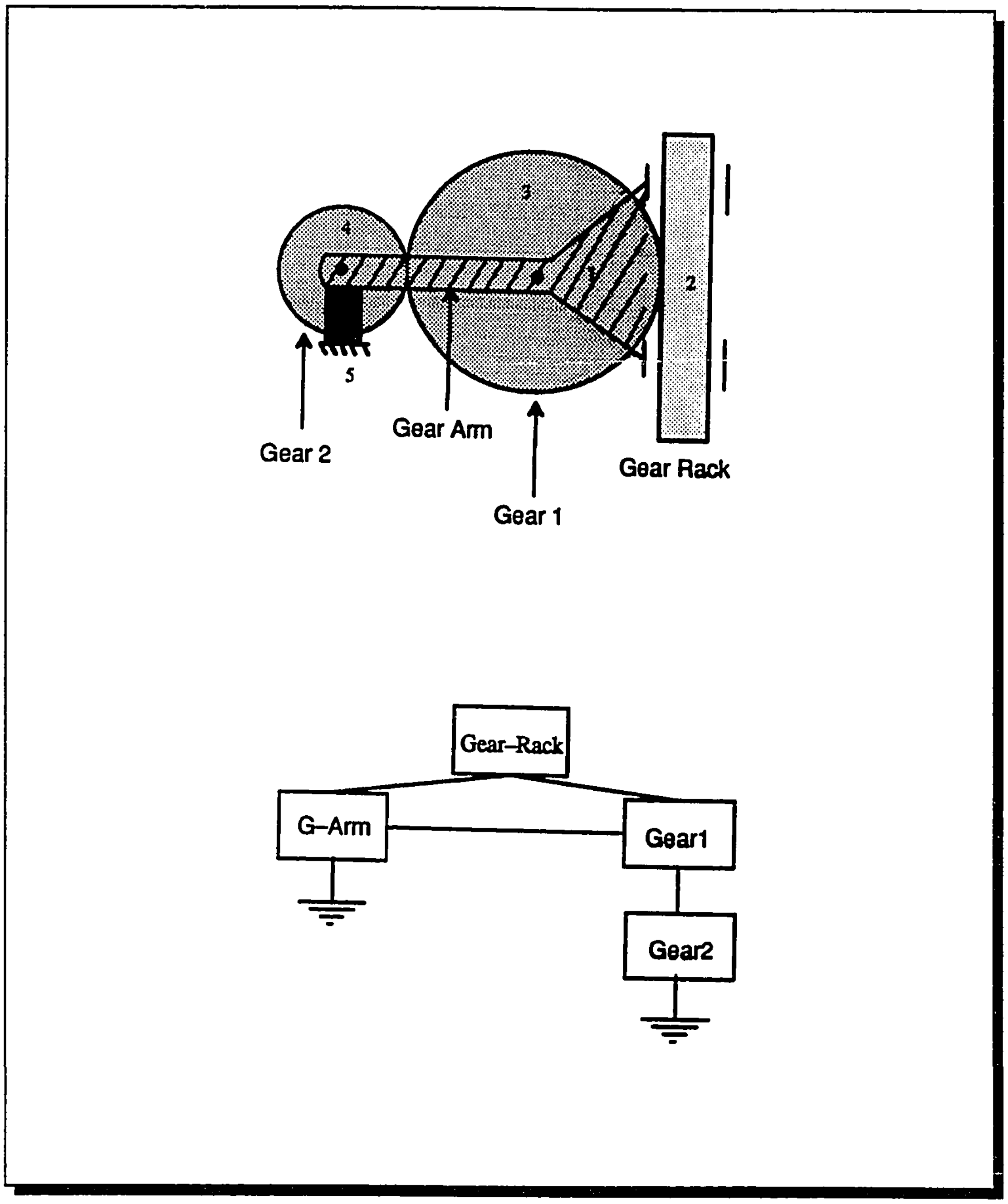


Figure 30. Resulting design of a Two-Fingered Robot hand for Design Case 5.

CHAPTER NINE

Summary and Conclusion

Highlights

- Summary of thesis
- Limitations of Current Research
- Significance of our Research Area
- Contributions of our Research
- Future Directions

Summary and Conclusion

9.1 Summary of Thesis

The work presented in this dissertation has focused on two main directions. The first direction is the study of engineering design. Under this direction, our work has centered on understanding the engineering design process, understanding the various methods employed by designers, studying the various tools used in creative design, and finally, incorporating the results into an application area within the mechanical design domain. Our work in the mechanical design has been based on studying the conceptual design stage for the creation of mechanisms based on the graph representation of kinematic structures. This application area represents a well established method for systematic synthesis of mechanisms that has been used for many years. Using kinematic structures as the basis for the design method provides for an excellent environment during the conceptual design stage. In this method, the structure and the function of a mechanism are treated separately, and the details of a design is left for later stages. The result is a well-defined and readily controlled form of design representation that is feasible for computer implementation.

The second and major direction of our research has been to integrate artificial intelligence techniques with design to produce a class of expert systems that can reason intelligently and efficiently for the automation of mechanical synthesis. We have implemented a number of systems based on integrating rule-based systems with graph representation and procedural languages [14,69,70]. The results of these implementations demonstrated the advantages of employing experiential knowledge as an aid for mechanical designers. It also showed the advantages of using graphs as a representation method. However, what we learned from these systems is that they have limited reasoning capabilities and thus lack many of the elements that would classify them as intelligent systems. The overall process was based on exhaustive generation and evaluation of design alternatives. This has been and remains the case of most of the application oriented expert systems that exist today. Our goal then has been to incorporate elements such as decision making during the design process, the ability to reuse existing solutions, the ability to learn from experience, and the ability to use multiple reasoning strategies and multiple knowledge representation. Such needs have directed our research towards a number of areas that include analogical reasoning, case-based reasoning, and first-principle reasoning.

The outcome of such studies has been the development of a system called SMARTs [44] which integrates three different forms of reasoning strategies for the reuse of mechanism design. An analogical reasoner followed by a heuristic reasoner, and a first principle reasoner.

The results of applying SMARTs to the creation of mechanisms using analogy has been compared with our previous implementations using traditional approaches to the development of expert system. In our previous work, the system relies on generating all the possible combinations of a design using a graph representation, and then applies a set of evaluation rules to determine the feasibility of each design configuration. When that system was used in the creation of the eight-link variable-stroke engines, the resulting combination produced 270 different design configurations. Out of this large number of design alternatives only three mechanisms were considered feasible. While this exhaustive method is guaranteed not to miss any design configuration, it is clearly a very costly approach, bearing in mind that the evaluation criteria has to be applied to each of the 270 design configurations. However, when the same problem was tested using SMARTs starting with a training example, the result presents only two feasible design configurations in addition to the training example. This result clearly shows the advantage of employing the concept of reusing past experience through analogical reasoning where the system uses an existing design to generate other design cases in different application domains.

We have presented a system that attempts to follow a natural path of reasoning resembling the reasoning process used by human designers. The system is capable of solving new design problems using analogy and shows capabilities of learning from training and from solving new problems. SMARTs has been applied to the mechanism field based on the method of separating the structure from the function of mechanisms, which was found to be an important tool in the conceptual design of mechanisms. This methodology of separating the function from structure has proved to be valuable in practice as well as in research.

SMARTs explicitly separates its knowledge and reasoning capabilities into three modules, thus making it more flexible and easy to use. Each of these modules serves as part of the overall problem solving process. Whenever the analogical process in the

Case-Base Reasoner is unable to arrive at a solution, it switches to the Heuristic Reasoner, and then to the First-Principle Reasoner attempting to derive the required information. This organizational framework provided SMARTs with the capability of solving new designs in different design applications including variable-stroke engines, robot hands, riveter, casement windows, as well as deployable space structures.

The indexing method used by SMARTs is based on a hierarchy of a number of discrimination features and allows the system to readily search and retrieve the most appropriate existing design cases. It also guarantees that the system will always find an existing design if one exists, even if the degree of similarity between the existing case and the target case is weak. The same index was also used to properly address the newly solved design cases. The indexing in our case eliminates the need for an expensive generalization procedure.

Using first-principle knowledge (deep knowledge) provides SMARTs with the ability to solve problems that are otherwise difficult to solve by relying only on the CBR and HR. Research on this area [13] shows the need for utilizing deep knowledge in today's expert systems. Such needs stem from the fact that heuristics alone may not provide for sufficient explanations or the competency to solve problems that are difficult in nature.

Some of the limitations observed in SMARTs include its inability to understand the explanation part of the design rules. Overcoming this limitation will provide SMARTs with a stronger reasoning capability and will permit for a more meaningful method of transforming the rules from one design to another. Another limitation that needs to be addressed, although inherent in the design method itself, is the lack of functional requirements and functional rules. While many of the structural rules directly affect the function of a mechanism, more functional rules will provide a more efficient and capable transformation process. Functional requirements can be used along with structural requirements to retrieve existing structures that would satisfy both requirements.

9.2 Limitations of Current Research

The limitations of current research can be placed into two categories. First, existing expert systems attempting to integrate AI with engineering design show great deficiencies in their reasoning capabilities. The majority of the research has been and is still focused on the application of rule-based systems. Rule-based systems clearly do not fully support design problems. Even when integrated with other tools or with procedural languages, the results remain less than adequate for complex problems such as design. In addition to these problems, and as consequence of them, much of the application oriented research has been focused on the problem of diagnosis. This is due to the nature of rule-based systems which are more suitable for diagnosis rather than design.

The second deficiency lies in the research on analogical reasoning, which has ignored the application side of this powerful reasoning strategy. The literature clearly shows this problem. There have been many recent studies on analogy, but very little has been accomplished in presenting complete models for this reasoning strategy. Chapter 2 illustrates that most of the research on this topic presents models that focus only on specific parts of the Analogy process, thus assuming the rest of the stages. Additionally, many of the accomplishments have relied on trivial examples with little real use. Simple examples do not provide for a real measure of the success of such systems when applied to real-world problems. This is clearly observed when studying much of the research on Machine Learning [57]. Such work shows that there have been many real issues that were either substantially simplified or just assumed. For example, the issue of background knowledge, which is important in real-world problems, has not been addressed effectively.

While the integration of expert systems and Engineering design has been considered an important issue in both AI and Engineering domains, the number of systems that employ reasoning techniques such as analogy or first-principle knowledge is relatively small. This suggests that the application area for such research is in great need of new attempts to demonstrate reasoning capabilities applied to practical and useful problems.

9.3 Significance and Contributions of our Research

"The proper study of mankind is the science of design."

Simon 1969 [1]

"Analogy pervades all our thinking, our everyday speech and our trivial conclusions as well as artistic ways of expression and the highest scientific achievements." Polya 1957 [2]

To show the significance of our work, we look at three different aspects of our research area. The *usefulness* of the research area, *contributions* to the field of Computer Science and other fields, and the *potential for future contributions*.

The work presented in this dissertation represents a comprehensive attempt to integrate two important aspects of human achievement and intelligence: engineering design, a highly decision making activity requiring a great deal of creativity; and analogy, which is considered a central issue in human intelligence.

There is no doubt of the significance of engineering design and its effects on our daily life. Consequently, our efforts to develop intelligent systems to aid human designers in this complex domain of expertise will certainly be of a great use in the future.

The research in AI especially in the direction of Expert Systems has been carried on for many years. However, the traditional approaches to the development of expert systems have failed in many cases to convince domain experts of potential of this technology. Our goal in this research is to demonstrate the usefulness of this technology as applied to a practical and complex domain of expertise such as the Mechanical Engineering Design. Our objective is to show the feasibility of implementing a system that can demonstrate a number of intelligent capabilities, such as the ability to reason effectively and the ability to learn from experience. Such topics have been studied within the AI research but not to the extent of application. To satisfy our objectives, we have carefully selected the mechanism design approach described in this dissertation. Many of the examples that we have selected in Chapter 8 to demonstrate our system have been used previously in real-world design problems. The design method we have adopted represents a method that is currently employed in both academia and industry. Companies such Kodak and General Motors have been using the same design method.

The system model that has been described in this dissertation distinguishes itself from many of the existing systems by the diversity and flexibility of its reasoning capabilities. This is accomplished through the hierarchical system architecture, which consists of three levels of reasoners. Through such an architecture, SMARTs can handle many design episodes which otherwise would not be solved by analogy alone. The multi-level reasoning and the incorporation of background knowledge reduces the uncertainty present in analogical reasoning. The operation of the system covers all the aspects of analogy as suggested by many researchers [40,46]. The overall architecture of SMARTs represents a general model that can be used in the application of various domains of expertise. The hierarchy and operation of the three modules represent a complete system which can utilize analogy, heuristic, and first-principle reasoning. The success of our system shows that there is an eminent need to use systems with multi-reasoning strategies applied to real-world design problems. As a general model, many of the limitations which were bound by implementation can be realized in the future.

The significance of our research is not limited to the development of a system that uses a sophisticated reasoning capability in a highly complex domain, but goes far beyond that. Our research represents a diverse area which spans a number of issues, such as studies in Engineering design, Knowledge representation, and Machine Learning studies. More importantly, our studies in the area of Creative Design present a new direction which has not yet been fully explored by AI researchers. There has been much research directed at understanding, design and our Literature Survey covers a number of such studies (Chapter 2). However, the area of creative design is a research area with its own rights existing in the engineering design research. This area is different from other studies because it is focused directly on the study of creative tools in design. Our attempts to integrate this research area with AI will facilitate many opportunities for successful future research.

Our contributions can be summarized in the following points:

1. Demonstrating the need for application-oriented intelligent systems that must operate in the real-world.

2. Presenting a comprehensive comparative study of analogical reasoning and its limitations. Emphasizing the need to study the practical aspects of analogy and how to incorporate it in real-world systems.

3. The development of a reference frame-work which can be used as a model for the construction of intelligent systems that use previous knowledge.

4. Presenting the need for and the feasibility of integrating AI techniques with Creative Design.

5. Exploring the advantages of employing graph representation of kinematic structures for the automation of mechanisms synthesis.

9.3 Future Directions

The first step in our future work will be to expand our model. The expansion will involve two different phases. The first is to account for and treat the various limitations presented in our model, and second, to enhance the performance and versatility of the system. Our model shows that there is a need to understand the semantics of the explanation part of the design rules. While this may not be an easy task, it is an essential step for the generalization of rules. If the semantics of the explanation are understood, then rules can be easily transferred between different cases across multiple domains. This step can be further expanded for the automatic creation of domain dependent rules. After the generalization of rules based on a specific domain, a specialization phase could be followed based on a new case in a new domain with little or no background knowledge. The result of such a procedure could be a number of new specific rules covering the new domain and which is derived from previous examples.

The next future consideration is to capitalize on first-principle knowledge. We would like to incorporate more knowledge in the FPR than the already existing components description. Fundamental knowledge that includes procedures, hints, casual knowledge, and essential domain independent knowledge, will increase the system's capabilities to reason more constructively and to solve problems which may exist in a larger spectrum of application domains.

Another important future consideration is the addition of more functional requirements as part of the overall design procedure. Incorporating functional requirements results in a more meaningful design method. Such functional requirements will then play a major role in the retrieval, transformation, and indexing stages of the analogy process.

Finally, and most importantly, we will consider a comprehensive study of Creative Design which will be focused on the development of future methods based on our current model. Emphasis in our future work will be placed on the topic of Design by Inversion.

References

1. Acosta, Ramon D. and Huhns, Michael N., "Alternative Explanation Structures for Explanation-Based Learning," *MCC Technical Report No. ACA/AI-CAD-079-88*, March 1988.
2. Adelson, B., "Cognitive research: Uncovering How Designers Design; Cognitive Modeling: Explaining and Predicting How designers Design," *Research in Engineering Design*, vol. 1, No. 1, pp. 35-42, 1989.
3. Ashley, Kevin D. and Rissland, Edwina L., "Dynamic Assessment of Relevancy in A Case-Based Reasoner," *Proceedings The 4th Conference on AI Applications*, pp. 208-214, March 1988.
4. Barr, Avron and , Edward A. Feigenbaum, *The Handbook of Artificial Intelligence,* 2, William Kaufmann, 1982.
5. Bennett, J.S. and Engelmore, R.S., "SACON: A Knowledge-Based Consultant for Structural Analysis," *Proceedings of The 6th International Joint Conference on AI IJCAI*, pp. 47-49, 1979.
6. Brown, D. C. and Chandrasekaran, B., "A Class of Mechanical Design Activity," in *Knowledge Engineering in Computer-Aided Design*, ed. J.S. Gero, pp. 259-263, North-Holland , 1985.
7. Brown, J.S., Burton, R.R., and Kleer, J. de, "Knowledge Engineering and Pedagogical Techniques in SOPHIE I,II,III," in *Intelligent Tutering Systems*, ed. D. Sleeman and J.S. Brown, Academic Press, 1982.
8. Buchsbaum, F. and Freudenstein, F., "Synthesis of Kinematic Structure of Geared Kinematic Chains and Other Mechanisms," *Journal of Mechanisms*, vol. 5, pp. 357-392, 1970.
9. Burstein, Mark H., "Concept Formation By Incremental Analogical Reasoning and Debugging," in *Machine Learning, An Artificial Intelligence Approach*, ed. R.s. Michalski, J.G. Carbonell, and T.m. Mitchell, vol. 2, pp. 351-369, 1986.
10. Carbonell, Jaime and Veloso, Manuela, "Integrating derivational analogy into a general problem solving architecture," *Proceedings Case-Based Reasoning Workshop*, pp. 104-123, May 1988.
11. Carbonell, Jaime G., "Learning by Analogy: Formulating and Generalizing Plans From Past Experience," in *Machine Learning An Artificial Intelligence Approach*, ed. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp. 137-161, Morgan Kaufman Publc. Inc., 1983.
12. Carbonell, Jaime G., "Derivational Analogy: A Theory of reconstructive Problem Solving and Expertise Acquistion," in *Machine Learning, An Artificial Intelligence Approach*, ed. R.s. Michalski, J.G. Carbonell, and T.m. Mitchell, vol. 2, pp. 371-392, 1986.
13. Chandrasekaran, B. and Mittal, Sanjay, "Deep versus compiled knowledge approaches to diagnostic problem-solving," *International Journal Man-Machine Studies*, vol. 19, pp. 425-436, 1983.

14. Chew, M., Shen, S.N.T, and Issa, G., "Application of Knowledge-Based Systems to the Conceptual Design of Mechanisms," *Accpeted for publication at the ASME Journal on Mechanical Design, presented at the 21st Biennial Mechanisms Conference in Chicago*, Sept. 1990.

15. Chouraqui, E., "Construction of a model for reasoning by analogy," in *Progress in Artificial Intelligence*, ed. Ellis and Horwood, pp. 169-183, 1985.

16. Cohen, Paul R. and , Edward A. Feigenbaum, *The Handbook of Artificial Intelligence*, 3, William Kaufmann, 1982.

17. Datseris, P. and Palm, W., "Principles on the Development of Mechanical Hands Which can Manipulate Objects by Means of Active Control," *Journal of Mechanisms, Transmissions, and Automation in Design*, pp. 1-9, 1984.

18. Dershowitz, Nachum, "Programming by Analogy," in *Machine Learning An Artificial Intelligence Approach*, ed. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, vol. 2, pp. 395-423, 1986.

19. Dixon, John R., "Artificial Intelligence and Design: A Mechanical Engineering View," *Proceedings AAAI-86 5th National Conference on AI.*, vol. 2, pp. 872-877, August 1986.

20. Dobrjanskyj, L. and Freudenstein, F., "Some Applications of Graph Theory to the Structural Analysis of Mechanisms," *Journal of Engineering for Industry*, vol. 89, pp. 153-158, Feb. 1967.

21. Duda, R., Gasching, J., and Hart, P.E., "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in *Expert Systems in the Micro-Electronic Age.*, ed. D. Michie, Edinburgh, 1979.

22. Edel, D. Henry Jr., in *Introduction to Creative Design*, Prentice-Hall, Inc., 1967.

23. Eliot, Lance B., "Analogical Problem-Solving and Expert Systems," *IEEE Expert*, pp. 17-26, Summer 1986.

24. Erman, L.D., Hays-Roth, F., Lesser, V.R., and Reddy, D.R., "The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, vol. 12, no. 2, pp. 213-253, 1980.

25. Evans, T. G., "A program for solution of geometric analogy intelligent test questions," in *Semantic Information Processing*, ed. M. Minsky, MIT Press, Cambrige, MA, 1968.

26. EXSYS, inc., *EXSYS Expert System Development Package.*, EXSYS inc., P.O. Box 75158 Contr. Sta. 14, Albuquerque, NM 87194, 1985.

27. Fenves, Steven J. and Norabhoopipat, Thanet, "Potentials For Artificial Intelligence Apllications in structural Engineering Design and detailing," in *Artificial Intelligence and pattern recognition in Computer Aided Design*, ed. J.C. Latombe, pp. 105-119, North-Holland 1978.

28. Fikes, Richard E. and Nilsson, Nils J., "STRIPS: A New Approach to The Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, vol. 2, pp. 189-208, 1971.

29. Fikes, Richard E., Hart, Peter E., and Nilsson, Nils J., "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, vol. 3, pp. 251-288, 1972.

30. Finger, Susan and Dixon, John. R., "A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Processes," *Research in Engineering Design*, vol. 1, pp. 51-67, 1989.

31. Fink, Pamela K. and Lusth, John C., "Expert Systems and Diagnostic Expertise in the Mechanical and Electrical Domains," *IEEE Transactions on Systems, Man, And Cybernetics*, vol. SMC-17, NO. 3, pp. 340-349, May/June 1987.

32. Freudenstein, F., "The Basic Concepts of Polya's Theory of Enumeration, with Application to the Structural Classification of Mechanisms," *Journal of Mechanisms*, vol. 3, pp. 275-290, 1967.

33. Freudenstein, F. and Maki, E. R., "The Creation of Mechanisms According to Kinematic Structure and Function," *Enviroment and Planning B*, vol. 6, pp. 375-391, Sep 1979.

34. Freudenstein, F. and Maki, E. R., "Development of an Optimum Variable-Stroke Internal Combustion Engine Mechanism From the Viewpoint of Kinematic Structure," *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 105, pp. 259-268, June 1983.

35. Gentner, Dedre, "Structural Mapping: A Theoretical Framework for Analogy," *Cognitive Science*, vol. 7, no. 2, pp. 155-170, 1985.

36. Gentner, Dedre, "Analogical Inference and Analogical Access," in *Research Notes in AI (Analogica)*, ed. Armand Prieditis, Pitman/Morgan Kofmann, 1988.

37. Gero, J.S., Radford, A.D., Coyne, R., and Akiner, V.T., "Knowledge-Based Computer-Aided Architectural Design," in *Knowledge Engineering in Computer Aided Design*, ed. J.S. Gero, pp. 57-81, 1985.

38. Greiner, Russel, "Learning by Understanding Analogies," *Artificial Intelligence*, vol. 35, pp. 81-124, 1988.

39. Grimson, W. E. and Patil, Ramesh S., *AI in the 1980s and beyond*, pp. 8-25, Massachusetts Institute of Technology, 1987.

40. Hall, Rogers P., "Computational Approaches to Analogical Reasoning: A Comparative Analysis," *Artificial Intelligence*, vol. 39, No. 1, pp. 39-120, 1989.

41. Hammond, Kristian, "Case-Based Planning," *Proceedings Case-Based Reasoning Workshop*, pp. 17-20, May 1988.

42. Harty, N., "An Aid To Preliminary Design," in *Knowledge Based Expert Systems in Engineering: Planning and Design*, ed. D. Sriam and R.A. Adey, pp. 377-392, Computational Mechanics Pub. 1987.

43. Huhns, Michael N. and Acosta, Ramon D., "Argo: A system for design by analogy," *Procedings the 4th Conference on Artificial Intelligence Applications*, pp. 146-151, march, 1988.

44. Issa, G., Shen, S.N.T, and Chew, M., "SMARTs: Synthesis of Mechanisms using Analogical Reasoning Techniques," *Submitted for publication as a feature article in IEEE Expert*, 1992.

45. Johnson, Ray C., "Introduction to Creative Design," in *Mechanical Design Synthesis*, pp. 29-33, Krienger Publ. Co., 1978.

46. Kedar-Cabelli, Smadar, "Analogy From a Unified Perspective," in *Analogical Reasoning*, ed. David H. Helman, pp. 65-103, Kluwer Academic Publ., 1988.

47. Kedar-Cabelli, Smadar, "Towards a Computational Model of Purpose-Directed Analogy," in *Research Notes in AI (Analogica)*, ed. Armand Prieditis, pp. 89-107, Pitman/Morgan Kaufmann Publs., 1988.

48. Kling, Robert E., "A Paradigm for Reasoning by Analogy," *Artificial Intelligence*, vol. 2, pp. 147-178, 1971.

49. Kolodner, Janet L., *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model,* Lawrence Erlbaum Associates, Publishers, 1984.

50. Kolodner, Janet L. and Simpson, Robert L. Jr., "Problem Solving and Dynamic Memory," in *Experience, Memory, and Retrieval*, ed. J.L. Kolodner and C.K Riesbeck, pp. 99-113, 1986.

51. Kolodner, Janet L., "Extending Problem Solver Capabilities Through Case-Based Inference," *Proceedings of the 4th int'l Machine Learning Workshop*, 1987.

52. Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A., and Lederberg, J., *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project.*, McGraw-Hill, N.Y., 1980.

53. Maher, M. L. and Fenves, S. J., "HI-RISE: An Expert System For Preliminary Structural Design Of High Rise Buildings," in *Knowledge Engineering in Computer-Aided Design*, ed. J.S. Gero, pp. 125-140, North-Holland, 1985.

54. Martin, James and Oxman, Steven, *Building Expert Systems,* pp. 45-150, Prentice Hall, 1988.

55. Mayorian, M. and Freudenstein, F., "The Development of an Atlas of Kinematic Structure of Mechanisms," *ASME Journal of Mechanisms, Transmissions and Automation in Design*, vol. 107, 1985.

56. McDermott, Drew, "Circuit Design as Problem Solving," in *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, ed. J.C. Latombe, pp. 227-245, North-Holland 1978.

57. Mechalski, R.S. and Kodratoff, Y., "Research in Machine Learning; Recent Progress, Classification of Methods and Future Directions," in *Machine Learning: An Artificial Intelligence Approach III*, ed. Y. Kodratoff and R.S. Michalski, pp. 3-30, Morgan Kaufman Publ., 1990.

58. Mittal, Sanjay and Araya, Agustin, "A knowledge-Based Framework for Design," *Proceedings aaai-86, Fifth national conference on AI*, vol. 2, pp. 856-865, Aug. 1986.

59. Mostow, Jack and Barley, Mike, "Automated Re-Use of Design Plans," *Proceedings of the 1987 Inter'l Conference on Engineering Design*, vol. 2, pp. 632-647, 1987.

60. Navinchandra, D., "Case Based Reasoning in CYCLOPS, A Design Problem Solver," *Proccedings Case-Based Reasoning Workshop*, pp. 286-301, May 1988.

61. Pierick, Jeff, "A knowledge Representation Technique for Dealing with Hardware Configuration," *Proceedings aaai-86, fifth national conference on AI*, vol. 2, pp. 991-995, Aug. 1986.

62. Polya, G., *How to solve it: Anew Aspect of Mathematical Method*, Princeton University Press, NJ, 2nd ed., 1957.

63. Pople, H., "The Formation of Composite Hypotheses in Diagnostic Problem Solving," *The 5th International Joint Conference on AI IJCAI*, pp. 1030-1037, 1977.

64. Reiter, Raymond, "A Theory of Diagnosis from First Princibles," *Artificial Intelligence*, vol. 32, pp. 57-95, 1987.

65. Rychener, Michael D., "Research in Expert Systems for Engineering Design," in *Expert Systems for Engineering Design*, ed. Michael D. Rychener, pp. 1-33, 1988.

66. Rychener, Michael D., "Research in Expert Systems for Engineering Design," in *Expert Systems for Engineering Design*, ed. Michael D. Rychener, pp. 1-33, 1988.

67. Sandler, Ben-Zion, in *Creative Machine Design*, Paragin House Publ., 1985.

68. Schank, R., *Dynamic Memory: A theory of reminding and learning in computers and people*, Cambridge University Press, 1982.

69. Shen, S.N.T, Chew, M., and Issa, G., "Expert System Approach Using Graph Representation and Analysis for Variable-Stroke Internal-Combustion Engine Design," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 4, no. 3, pp. 389-408, 1990.

70. Shen, Stewart N.T., Chew, Meng-Sang, and Issa, Ghassan F., "Expert System Approach For Evaluating Engine Design Alternatives," *SPIE proceedings on Applications of Artificial Intelligence VII*, vol. 1095, pp. 533-543, March 1989.

71. Shinn, Hong S., "Abstractional Analogy: A model of Analogical Reasoning," *Proceedings Case-Based Reasoning Workshop*, pp. 370-387, May

1988.

72. Shortliffe, E., *Computer-Based Medical Consultation: MYCIN*, Elsevier/North-Holland, 1976.

73. Shortliffe, E.H. and Buchanan, B.G., "A model of inexact reasoning in medicine," *Mathematical Biosciences*, vol. 23, pp. 351-379, 1975.

74. Simon, Herbert A., *The Science of the Artificial*, The MIT Press, 1969,1981.

75. Sussman, G.J., "Electrical Design: A Problem for Artificial Intelligence Research," *The Fifth International Joint Conference on AI IJCAI*, 1977.

76. Sussman, G.J., "Science: At the Boundary Between Analysis and Synthesis," in *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, ed. J. Latombe, North-Holland, 1978.

77. Tomiyama, Tetsuo and Yoshikawa, Hiroyuki, "Requirements and Principles for Intelligent CAD Systems," in *Knowledge Engineering in Computer Aided Design*, ed. J.S. Gero, pp. 1-23, 1985.

78. Uicker, J. J. and Raicu, A., "A Method for Identification and Recognition of Equivalence of Kinematic Chains," *Mechanism and Machine Theory*, vol. 10, pp. 375-383, 1975.

79. Ulrich, K.T. and Seering, W.P., "Computational Approach to Conceptual Design," *Proceedings of International Conference on Engineering design ICED 87*, August 1987.

80. Winston, P.H., "Learning by creating and justifying transfer frames," *Artificial Intelligence*, vol. 10, no. 2, pp. 147-172, 1978.

81. Winston, P.H., "Learning and Reasoning by Analogy," *Communication of the ACM*, vol. 23, no. 12, pp. 689-703, 1980.

82. Winston, P.H., "Learning new principles from precedents and excercises," *Artificial Intelligence*, vol. 19, pp. 321-350, 1982.

83. Woo, L.S., "Type Synthesis of Plane Linkages," *Journal of Engineering for Industry*, vol. 3, pp. 159-171, 1967.

Autobiography

Ghassan F. Issa

Date Of Birth: December 20th, 1959

Place Of Birth: Kuwait

Education:

Ph.D. in Computer Science, May 1992, Old Dominion University, Norfolk, Virginia.

Master of Science in Computer Science, Dec. 1987, Old Dominion University, Norfolk, Virginia.

Bachelor of Science in Electrical Engineering, Nov. 1984, Tri-State University, Angola, Indiana.

Bachelor of Engineering Technology in Electronic Engineering, Aug. 1983, University of Toledo, Toledo, Ohio.

Titles Held:

- *Division Chairman, Information Science.* Commonwealth College, Virginia Beach, Virginia.
- *Instructor of Computer Science.* Commonwealth College, Portsmouth, Virginia.
- *Research and Teaching Assistant.* Old Dominion University, Norfolk, Virginia.
- *Programmer.* Archimedes Laboratory Systems, Rowlett, Texas.
- *Technician.* Al-Taneeb Trading Co., Kuwait.

Publications

S.N.T Shen, M. Chew, and G. Issa, "Expert System Approach for Generating and Evaluating Engine Design Alternatives," *Proceedings of SPIE, Applications of Artificial Intelligence VII*, Miami Fl., vol. 1095, March 1989, pp. 532-543.

M. Chew, S.N.T. Shen, and G. Issa, "Application of Knowledge-Based Systems to Kinematic Structural Synthesis of Mechanisms," *Proceedings of ASME, 21st. International Conference on Mechanisms*, Chicago, Il., Sept. 1990, pp. 323-330.

M. Chew, G. Issa, and S.N.T. Shen, "Expert System for Robot Hand Design Using Graph Representation," *Proceedings of the Fifth International Conference on CAD/CAM, Robotics and Factories of the Future*, Dec. 1990, Norfolk, Virginia.

S.N.T Shen, M. Chew, and G. Issa, "Expert System Approach for Generating and Evaluating Engine Design Alternatives," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 4, No. 3, May 1990, pp. 389-408.

M. Chew, S.N.T. Shen, and G. Issa, "Application of Knowledge-Based Systems to Kinematic Structural Synthesis of Mechanisms," Accepted for publication in the *ASME Journal of Mechanical Design.*

M. Chew, S.N.T. Shen, and G. Issa, "Incorporating different Knowledge Representation for the Conceptual Design of Mechanisms," Accepted for Publication in the ISMM Conference, Miami Fl.

G. Issa, S.N.T. Shen, and M. Chew, "SMARTs: Synthesis of Mechanisms using Analogical Reasoning Techniques," Submitted for Publication as feature article in IEEE Expert.

G. Issa, S.N.T Shen, and M. Chew, "Artificial Intelligence in the Automation of Creative Design Tools." To be submitted to the Intelligent Computer Aided Design Conference (ICAD '92).

Professional Affiliations

ACM	Association of Computing Machinery.
ASME	American Society of Mechanical Engineers.
AAAI	American Association of Artificial Intelligence.
IEEE	Computer Society.
SIGART	Special Interest Group for Artificial Intelligence.
SPIE	International Society for Optical Engineering.