INVESTIGATIONS INTO DISTRIBUTED ARTIFICIAL INTELLIGENCE TECHNIQUES FOR DESIGN WITH APPLICATIONS TO INSTRUMENTS

BY

HOOMAN KHOUI

A THESIS SUBMITTED FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN ELECTRICAL, ELECTRONIC AND
INFORMATION ENGINEERING

THE CITY UNIVERSITY
DEPARTMENT OF ELECTRICAL, ELECTRONIC AND
INFORMATION ENGINEERING
LONDON

MARCH 1995

Table of Contents

CHAPTER 4
TOWARDS AN INTELLIGENT AND ADAPTIVE DISTRIBUTED PROBLEM-SOLVER FOR DESIGN OF INSTRUMENTS

CHAPTER 5
GENETIC ALGORITHMS FOR DESIGN OPTIMIZATION OF INSTRUMENTS

CHAPTER 6
CLASSIFIER SYSTEMS FOR THE AUTOMATION OF INDUCTIVE-REASONING IN THE DESIGN PROCESS

CHAPTER 7
CONCLUSIONS

APPENDICES

ABSTRACT

This Thesis is concerned with the application of Distributed Artificial Intelligence techniques for the design of instruments.

In this thesis it is argued that, the early stages of the design process can be automated by the use of Distributed Artificial Intelligence systems that are contractual in their communication and control.

A Distributed Problem Solver is proposed, and implemented, for the purpose of conceptual design of instruments. The system consists of a community of knowledge-based agents, with expertise on design of instrument sub-systems. The agents, use a task-sharing form of cooperation for dynamic problem decomposition and sub-problem distribution phases of the design problem solving. New design concepts are generated by suitable combination of partial solutions.

To incorporate learning capabilities into our Distributed Problem Solver, we have proposed the use of Classifier System Modules as inductive knowledge-based agents. The application of Classifier Systems and Genetic Algorithms in the context of a number of concrete instrument design problems is investigated.

A normalized formulation is applied to the multi-modal design optimization of a Linear Variable Differential Transformer. A number of important proposals for the application of classifier systems to the design automation of instruments are detailed. In particular, an implemented classifier system is used for the purpose of design heuristic extraction for corrugated diaphragms, using a set of dimensionless curves. In this application, the classifier system has produced a set of useful design heuristics by direct interactions with the specified mathematical model.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude and appreciation to my supervisors, Dr. Mohsin. K. Mirza and Professor Ludwik Finkelstein, whose guidance and encouragement was a great help to me throughout the project.

Professor Ludwik Finkelstein and Dr. Mohsin. K. Mirza will always stay in my memory as the best educators I have ever had, and I will always think of them with great respect.

I would, also, like to thank all my friends and colleagues for their help.

Finally, I wish to express my thanks to the Science and Engineering Research Council, who provided the financial support for this research work.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Design is the creative process by which one moves from a perceived need to a realized solution. It is a process which moves from the abstract to the concrete. Developments in conventional computer aided design systems have mostly provided environments for drafting and numerical analysis. Computer aids for generating ideas from which a design may evolve are not generally available (Finkelstein & Finkelstein, 1983; Mirza et. al., 1990b). In recent years, development of computer aids based on artificial intelligence techniques, for conceptual design of engineering systems, has become an active area of research (Gero, 1990; Adeli, 1988; Topping, 1989; Sriram, 1987).

The conventional use of computers for engineering design problems has been mostly directed at the later stages of the design process. They include tasks which can be described in terms of procedures and calculations. These computer based techniques are mostly used for analysis of candidate designs after they have been generated by designers. They include finite difference method, finite element method, and boundary element method.

However, the early stages of the design process can not be supported by conventional computing approaches. The early stages of the design process requires creativity and innovation and involves conceptualization and synthesis. Therefore, the early stages of the design process are dominated by inductive type of reasoning, and A.I. techniques combined with research in design methodology provide powerful means for the study and automation of this process.

This thesis is primarily concerned with investigations into the application of Artificial Intelligence techniques for engineering design problems in general, and for the design automation of instrument transducers in particular.

During the course of this thesis, it will be argued - by means of surveys, a number of proposals and implementations - that, complex engineering design problems, which have essentially inductive characteristics, can be automated by the use of Distributed Artificial Intelligence (DAI) systems which are contractual in their communication and control.

Distributed Artificial Intelligence (DAI), is concerned with the collaborative solution of global problems by a distributed group of entities. The entities may range from simple processing elements to complex entities exhibiting rational behaviour. The problem solving is collaborative in the sense that mutual sharing of information is necessary to allow the group as a whole to produce a solution, or to successfully accomplish a global task. The group of entities is distributed in that both control and data are logically, and often geographically, distributed.

Def

1.2 Thesis Organization

The thesis is organized in the following sequence:

In Chapter 2, we will investigate the application of Artificial Intelligence techniques for the engineering design problems. It is argued that complex engineering design problems are best tackled by using Distributed Artificial Intelligence (DAI) techniques. An overview of DAI, advantages and related research issues are presented. A critical review of current DAI systems applied to engineering design problems is, also, given.

Chapter 3 is concerned with the development of an adaptive intelligent system for the conceptual design of instruments using Distributed Artificial Intelligence (DAI) techniques. The implemented Distributed Problem Solver (DPS) system, for the conceptual design of instruments, consists of a community of Agents - an agent provides expertise on a particular aspect of the problem or solution of a sub-problem. An agent may be a complete expert system in its own right. Co-operation between agents is based, primarily, on Contract Net (CNET) approach.

Def.

Def.

Def.

Our objective, in chapter 4, is to investigate A.I. techniques which are promising for the design and implementation of adaptive DAI systems which are capable of improving their performance. At a coarse-grained level (i.e. at the level of inter-agent interactions), this is achieved by using task-sharing and result-sharing forms of cooperation. Task-sharing is implemented using the Contract-Net approach. At a fine-grained level (i.e. at a single agent setting), the overall performance of the DPS system hinges on the capabilities of each single agent (i.e. the knowledge-based expert system). This issue is further investigated by presenting a critical review of machine learning methods developed for single agents. These investigations led us to propose a theoretical DPS framework, in which an agent is considered as a Classifier System module.

In chapter 5, we will study the genetic algorithms which have been proposed to support inductive mechanisms for rule discovery in classifier systems. In this chapter, we first concentrate on the foundations of genetic algorithms, their applications, advantages and research issues. We, then, take up a comparative study of a number of reproductive strategies, in the context of two instrument design optimization problems: corrugated diaphragms and LVDTs. Finally, we investigate a number of techniques for the purpose of multimodal function optimization using genetic algorithms. These studies are carried out for the purpose of finding alternative optimal designs, satisfying the same user specified design criteria.

In chapter 6, we first present classifier system advantages, applications and research issues. Our main goal, in this chapter, is to investigate the applications of classifier systems to the design of instruments. To this end, we detail a number of proposals. In particular, we will investigate the feasibility of simulating parametric sensitivity analysis and dimensionless analysis, for the purpose of design heuristic extraction, as done by designers during the initial stages of the design process. For this purpose, a functional lumped parameter mathematical model is interfaced to an implemented classifier rule-based system. Simulation results and future work will be elaborated.

Chapter 7 is concerned with the conclusions and achievements of this study. Suggestions are made for fruitful future work.

CHAPTER 2

Artificial Intelligence in Engineering-Design

2.1 Introduction

In this chapter, we will investigate the application of Artificial Intelligence Techniques for the engineering design problems. The use of A.I. techniques combined with research in design methodology provides an opportunity to exploit the results of both to produce an understanding of design problem solving. This issue is investigated in section 2.2.

It is argued that complex engineering design problems are best tackled by using Distributed Artificial Intelligence techniques. In sections 2.3, 2.3.1 and 2.3.2, an overview of DAI, advantages and related research issues are investigated. A critical review of current DAI systems applied to engineering design problems is given in section 2.3.3.

Conclusions and directions for future research are stated in section 2.4.

2.2 Artificial Intelligence in Engineering Design

In recent years, there has been a growing interest in exploiting A.I. techniques to solve a wide range of engineering problems (Gero, 1988; Gero, 1990; Sriram,1986a; Sriram, 1986b). The conventional use of computers for engineering design problems has been mostly directed at the later stages of the design process. They include tasks which can be described in terms of procedures and calculations. These computer based techniques are mostly used for analysis of candidate designs after they have been generated by designers. They include finite difference methods, finite element methods, and boundary element method. The analysis techniques for engineering systems are well established (Mirza, 1992). However, the early stages of the design process can not be supported by conventional computing approaches.

The early stages of the design process requires creativity and innovation and involves conceptualization and synthesis. It is characterized by incompleteness, uncertainty, qualitative arguments, application of expertise and knowledge and the accumulation of experience.

A.I. techniques provide powerful means for the study and automation of the early stages of the design process.

In order to apply A.I. techniques effectively and efficiently to engineering design problems, there is a basic need for a systematic model of the design process. To this end, an extensive survey (Finkelstein & Finkelstein, 1983; Pahl & Beitz, 1984; Burton, 1990) has been made of the design literature, leading to the formulation of a general model of the design process. This model is based upon the broad agreement that exist in the literature on the elements of such a process. In this model, the design process consists of a sequence of stages, starting from the perception of a need and terminating in a final definition of a particular design configuration to satisfy that need. The process begins with information gathering and organization, delineating the design problem and collecting in an organized manner the basic information required for its solution.

The principal input to this stage is the requirement specification from the previous stage (or the initial primitive need statement).

The information gathering and organization is followed by the formulation of the value criteria which arise from the requirement and by which candidate designs may be evaluated.

The formulation of the value criteria is followed by the generation of a set of proposed or candidate solutions. This is the central activity of design, and currently the least supported by current A.I. systems.

The candidate designs are then analysed to determine those attributes relevant to the specification of requirements. This is achieved by calculation, simulation, building of models, etc.

Using the results of these analysis methods, the candidate designs are then evaluated in accordance with the value criteria adopted.

The process terminates with a decision step. A particular candidate may be judged satisfactory or optimal and may be accepted as the basis for the next design stage (i.e., for realization or implementation). Alternatively, it may be that none of the candidate designs are acceptable and further candidates need to be generated. Ultimately, if the design criteria cannot be satisfied, the criteria may need to be changed. This may involve changing decisions at the previous stage.

Models of the design process, and the methods by which these methods should be implemented in A.I. systems, has also been investigated by other design and A.I. researchers. In what follows, we give a brief account of these investigations:

Simon (1969) presents design as a problem solving process, and even more specifically, as a search process. The implication of search as a model for the design process is that design knowledge can be expressed as goals and operators.

As a general approach to modelling design, search provides a formalism for expressing design knowledge, However, it does not directly address some of the complexities of design problems. This is due to the fact that, design problem solving has additional characteristics that can be exploited by more explicit models.

In the pioneering work, carried out by Freeman & Newell (1971), a basic model of the design process has been proposed. This model is based on search processes of a simple design space containing primitive structures and their functionalities. The design requirement is given in terms of a sequence of top-level functional requirements for the overall artefact to be designed. Primitive structures provide primitive functionalities and they have to be connected to other primitive structures to provide more complex structures. This means that each primitive structure requires sets of functionalities to provide its own function. The matched required functionalities are consumed inside the composed structure, and the overall composed structure provides sets of functionalities provided by its component structures which are not consumed within the connections. Two heuristic search methods were indicated for structural recomposition: bottom-up (using a forward chaining search mechanism) and top-down (using a backward chaining search mechanism).

It is interesting to consider the three phases of design as a search process within a design space:

Design specification involves identifying the goal(s) of the design problem. Candidate design generation involves the search for one or more design solutions through the selection and application of appropriate operators. Design evaluation involves assessing whether the goal(s) have been identified.

This type of method is of particular importance in knowledge-based systems (Newell, 1979). Air-Cyl (Brown & Chandrasekaran, 1983) is an example of a design system that explore search spaces. This system can be understood as a searching in a space of parameters for the components of an air cylinder by using design plans that propose and modify parameter values.

The design process according to Maher (1990) is comprised of three phases:

1- Formulation
2- Design synthesis (Equivalent to candidate design generation)
3- Design evaluation and analysis

The models for design synthesis, according to Maher, includes:

1- Decomposition: This model is based on the idea dividing large complex problems into smaller, less complex sub-problems. The decomposition model provides a clear position about the type of design knowledge needed for a knowledge-based approach. This design synthesis method assumes that solutions to sub-problems will combine to form a good design solution and that design knowledge in the application domain has been specified as decomposition and recomposition knowledge.

2- Case-based reasoning: This is a model of design synthesis that directly uses design experience. The model uses analogical reasoning to select and transform specific solutions to previous design problems which are appropriate as solutions for a new design problem (Carbonell, 1986).

In this design synthesis method, there must be means for the identification of the necessary information about a previous design episode to reason about its suitability in the current design context. There also must be ways to modify the previous design to solve the current design problem. Although designers are good at using this type of design synthesis method, it has proved difficult to automate using conventional rule-based systems.

A general model of the design process, according to Gero (1990), involves the following activities: formulation, synthesis, analysis, evaluation, reformulation, and production of design description.

He proposes the use of conceptual schemas, in the form of design prototypes to represent design knowledge. The design prototype represents the design experience in such a way that allows representation at the concept level in the form of a class from which instances may be instantiated to meet the unique situation of a specific design problem.

The use of design prototype schemas is based on the fact that designers form their individual design experiences into generalized concepts or group of concepts at many different levels of abstraction. Therefore, they use schemas to represent their knowledge. Such schemas consist of knowledge generalized from aset of similar design cases and form a class from which individual cases can be accessed.

He argues that producing a design can be modelled by a process of prototype refinement. The prototype refinement requires a design space consisting of multiple prototypes. These prototypes serve as a basis for reasoning about the design specification, description, and requirements.

The selection of a prototype to be refined can occur at any level of abstraction. For example, during the early stages of design, a more abstract prototype is selected while during detailed design, a lower level prototype is selected.

Chandrasekaran (1990) argues that the most common top-level family of design methods are characterized as propose-critique-modify (PCM) methods. These methods have the subtasks of proposing partial or complete design solutions, verifying proposed solutions, critiquing the proposals by identifying cases of failure if any, and modifying proposals to satisfy design goals. He emphasises the sub-task of design proposal (i.e., generation of candidate solutions) as being the major part of the design activity, because most of the design knowledge is used in this sub-task.

Chandrasekaran identifies three groups of methods for design proposal:

(1) Problem decomposition-solution composition
(2) Retrieval of cases from memory
(3) Constraint satisfaction

Chandrasekaran proposes a task structure for automation of the design process, consisting of an organized hierarchy of tasks, methods and sub-tasks.

In his task structure analysis of the design process, the top-level task is the design itself which uses propose, critique and modify sub-tasks to achieve its goal. Each sub-task uses different methods to achieve its goals. For example, the sub-task of proposing alternative designs could use a decomposition method which in turn might generate sub-tasks for specification generation of sub-problems.

Different types of methods can be used for different sub-tasks. For example, a design system can use a knowledge-based problem solving method for the sub-task of creating a design but a quantitative method, such as a finite-element method, for the sub-task of evaluating the design. Therefore, the application of a method for a particular sub-task will generally provide further sub-tasks for which appropriate methods are necessary.

Chandrasekaran argues that there is no one ideal method for design, and good design problem solving is a result of recursively selecting methods based on a number of criteria including context and knowledge availability. Therefore, the structure of a design task is characterized by the way tasks, methods and sub-tasks and domain knowledge are related.

There is a general agreement that, the decomposition methods are the core precesses for design synthesis. Other design synthesis methods are used successfully only when initial design problem decompositions have been generated and design plans instantiated. Another important consideration in practical engineering design problems is the existence of several conceptual hierarchies. Each hierarchy represents a different aspect of the design problem.

Based on the above observations, Chandrasekaran & Brown (1989) has classified the design problem solving into three major categories:

1- Routine design: This is a form of design problem solving in which the way to decompose a design problem is already known, and compiled "design plans" are available for each major stage in design.

2- Innovative design: This class of design problem solving is characterized by the existence of powerful problem decompositions. However, new design plans might be needed for component problems or previous plans might need substantial modification. For example, design of a new automobile, does not involve new discoveries about decomposition, because the structure of the automobile has been fixed for quite a long time. On the other hand, several of the components might undergo technological changes, and routine design methods for some of these components are not suitable.

3- Creative design: This is open-ended "creative design". In this type of design effort, goals are ill-structured and effective decompositions are not known, and there is no design plans for sub-problems.

From the above review it becomes evident that there is general agreement among researchers as to what constitutes the overall stages of the design process. Design and A.I. researchers have proposed a number of different methods for each stage of the design process. These methods require different types of problem solving and knowledge. The design of engineering systems, almost always, consists of different stages comprising of task definition, solution generation, analysis, evaluation and decision. In a particular design problem, there are a multitude of inter-dependent design methods for different stages of the design problem solving and there are no predefined sequence of method invocations. I.e., a number of design stages can be carried out in parallel. Therefore, complex design problem solving can not be efficiently represented by a single knowledge source, and they are better described as a collection of cooperating knowledge sources, coordinating their knowledge, skill and plans to solve the overall design problem.

A number of complex issues arise using a distributed approach to design problem solving. These issues are best tackled by using Distributed Artificial Intelligence Techniques (DAI).

In the following sections, an overview of DAI as a subfield of A.I., its advantages and research issues is given. Next a critical review of current DAI systems applied to engineering design problems is presented.

2.3 Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI), is concerned with the collaborative solution of global problems by a distributed group of entities. The entities may range from simple processing elements to complex entities exhibiting rational behaviour. The problem solving is collaborative in the sense that mutual sharing of information is necessary to allow the group as a whole to produce a solution, or to successfully accomplish a global task. The group of entities is distributed in that both control and data are logically, and often geographically, distributed.

From a methodological perspective, virtually all research in DAI has focused on how a collection of agents can interact to solve a single common "global" problem, such as designing a very large scale integrated circuit (VLSI) chip, deriving a globally consistent interpretation of geographically distributed sensor data, or constructing a globally coherent plan for several agents.

In Distributed Artificial Intelligence (DAI), problems are solved by applying both artificial intelligence techniques and multiple problem solvers.

Two common approaches used for distributing control and data in DAI systems are:

1- Fine grained (Connectionist & Classifier Systems): This research is concerned with explaining higher mental functions and higher reasoning processes by reference to highly parallel collections of processes made up of very simple computing elements (McClelland & Rumelhart, 1987; Holland, 1986; Shaw & Whinston, 1989).

2-Coarse grained (Distributed Problem Solving): Research in Distributed Problem Solving (DPS), considers how the work of solving a particular problem can be divided among a team of coarse grained cooperating agents (Representing a distributed community of expert systems), that cooperate at the level of dividing and sharing knowledge about the problem and about the developing solution (Lesser & Corkill, 1987; Smith & Davis, 1981).

2.3.1 Advantages of Distributed Artificial Intelligence:

There are many reasons for distributing intelligence. These include:

1- Cost: A distributed system may contain a large number of simple processing modules (or computer units). This makes a DAI system substantially more cost-effective as compared to a centralized intelligent system.

2- Reliability: Distributed problem solvers, constructed from many general purpose machines, have the ability to adapt to failures at one or many nodes. In this way distributed problem solving provides ways to handle events like sensor failures and other causes of uncertain and incomplete information.

3- Efficiency: Concurrency almost always increases the speed of computation.

4- Resource sharing: Large problems can not be processed by simple computing modules due to limited knowledge and resources. This fact necessitates solving such problems by using a distributed community of co-operative programming modules.

5- Extendibility: Each agent entity within a DAI system encapsulates knowledge and procedures for solving a particular context dependent sub-problem allocated to it. This highly modular approach offers conceptual clarity and simplicity of design. This feature is particularly useful from software development point of view, because it facilitates software enhancements, modifications and extensions.

6- Naturalness: Almost all complex design problems are better described as collections of seperate processing modules. Complex design problems contain a huge amount of information and they have to be broken down into cooperating systems to be feasible. Multiple intelligent cooperating systems provide an opportunity to study how one system can reason about itself or other systems cooperating with it.

7- Contributions to Artificial Intelligence: DAI is highly important from a cognitive science viewpoint. Some researchers believe that all real intelligent systems are distributed (Hayes-Roth, 1980). Therefore DAI research is crucial to our understanding of real intelligence.

2.3.2 Basic Research Issues in Distributed Artificial Intelligence:

The basic research issues that are addressed in DAI are as follows:

1- Formulation, decomposition and allocation of problems:

The distribution of an overall design task among agents requires that the overall design problem be formulated and described in a way that it can be decomposed into sub-tasks and distributed. In practice formulation of problems requires some representation for the problem, as well as decisions on the boundries of the problem and on what is known and unknown (Gasser, 1988). In conventional DPS systems many of these activities are carried out by designers. The languages and concepts used for task description and formulation will affect how taks can be decomposed into parts, and the interdependencies among them. A design problem described from different perspectives may require different decompositions and different skills. Difficult problems of decomposition results, because of dependencies among sub-problems, decisions and actions of separate agents.

In real settings, for a particular problem, there are a multitude of problem descriptions and/or decompositions which will require different representations for the problem as well as decisions on the boundaries and what is known and unknown.

Problem decomposition necessitates sub-tasks to be allocated to particular agents, in other words techniques are needed for matching problem solvers to tasks (smith, 1980).

There has been little research in above areas and this has influenced the current approaches to the design using DAI techniques.

2- Methods for achieving distributed control:

The efficiency of distributing control in DAI systems is directly related to the overall coherence of the system. It is not clear how global coordination can be guaranteed from aggragation of actions based on local views with incomplete information. Coherence can be evaluated by considering several characterisations of a system's behaviour:

1- Solution Quality: This represents the system's ability to reach satisfactory solutions, and quality of solutions.

2- Efficiency: This represents the system's overall efficiency in achieving solutions.

3- Clarity: This is the conceptual clarity of the system's behaviour, and the usefulness of its behaviour. The system must be well-structured and describable so that the outside observer can understand it.

Most DAI researchers have analysed the coherence of a system by measuring its efficiency. From this perspective, incoherence can result from conflict over resources, from one agent undoing the results of another, and from duplicate actions carried out redundantly.

The amount of cooperation between agents in a DAI system can range from fully cooperative to antagonistic. In this discussion cooperation is considered as a special case of coordination among non-antagonist agents (Rosenschein & Genesereth, 1985).

Agents within fully cooperative systems will have to communicate extensively in order to change their goals and needs to suit the needs of other Agents or for the overall goal of the distributed problem solver system. In the middle of this range (i.e. half cooperative Agents) lie traditional systems which have no specific set of goals to achieve. Antagonistic systems often have no communication costs and may not communicate at all (Genesereth & Ginsberg, 1986).

The primary difficulty in establishing coherence and coordination is the uncertainty in agent invocation due to unavailability of centralized control or viewpoints (Corkill & Lesser, 1983).

A group of problem-solving agents achieve global coherence, if the actions of the agents makes sense with respect to the common goals of the entire group. Almost always DAI systems develop some type of organisation to guide the communication of tasks (goals) and results (data). The organisation that is developed says something about how the group achieves coherency.

An organisation provides a set of constraints and expectations about the behaviour of agents (e.g. a set of "roles") that defines the decision making and actions of particular agents (Gasser & Braganza, 1987).

The organisation implemented within different problem solving systems form a spectrum from totally free groups of Agents to master/slave relationships.

Totally free groups of Agents do not exhibit a control hierarchy and are naturally data driven. A problem in this case is solved when any Agent within the system comes up with an answer. Many production rule systems fall into this category.

Further along the organizational spectrum there is a gradual increased emphasis on the hierarchical relationships between problem solving Agents; where some Agents are given the task of an immediate solution of a Sub-Problem or play the role of a manager Agent giving a global direction to the Agents below them in the hierarchy. In these systems nodes with more global information guide nodes with less global information as decision-making data flow "upward" in progressively more abstract forms, and control information flows "downward".

The simplest hierarchy is two levels, in which one agent at the top level has complete information and all authority to control problem-solving agents at the second level. In such a "centralized" organisation, coherence is achieved by limiting decision making and by centralizing all decisions in one agent. This approach might degrade the overall solution quality in a dynamic environment (Chandrasekaran, 1981).

In Contract-Nets (as suggested by Smith & Davis (1981)) when an Agent play the role of a manager Agent to the Agents below it in the control hierarchy, its requests are not considered as orders, but they are executed by Contractor Agents through Neg~~a~~tiation. Thus, a lower-level Agent does not have to act upon receipt of a request from a manager Agent, but can wait for further developements and can refuse upon receiving an order if it has already received a higher priority task to do.

In the hierarchical organisations, the lowest level Agents are data driven as they will immediately achieve the low level Sub-Tasks allocated to them. The high level Agents are goal driven and decompose the Top-Level problems into Sub-Problems and distribute the Sub-Problems they can not handle within the Net, also they must be able to merge the solved Sub-Problems to obtain an overall solution for the Top-Level allocated problem.

The intermediate Agents within the hierarchy may range between the above two extremes.

In traditional systems no Negatiation takes place. In these systems each Agent will have to act upon receipt of a request from its superior Agents. These systems are totally goal driven. For an example of this strict organisation, we can consider a procedure call in a traditional programming language.

There is general agreement among DAI researchers that, no organisation structure is appropriate in all situations. Fox (1981) developed a taxonomy of how different types of organisations evolve as an organisation grows, becomes more complex, and encompasses more diverse activities. His view is based on the principle of bounded rationality. This principle, as expressed by Simon (1957) for human systems, states that the human mind's capacity for problem solving is limited. There is a limit to the amount of detailed data and control that one person can effectively manage. This implies the need for complex organisations to solve complex problems. Fox gives a detailed account of why human organisation metaphor is valid and how the principles of human organisational design (e.g. the functional or product division of a company), complexity reduction (Contracting and subcontracting, for example), and uncertainty reduction (such as contingency theory) can be applied to DAI systems.

3- The communication process:

In order to support coordination and coherence in DAI systems, agents need to communicate meaningfully. There is a need to design a common language to affect interaction, communication and organisation among cooperating agents.

This means that we need to know how to represent knowledge for communicating, and how to represent it in an interaction language. For example agents may need to communicate their mutual knowledge of each other, their current goals, etc. As in distributed problem solving, agents will have essentially different knowledge, the designed interaction language must have to allow for differences in knowledge, in order for communication and cooperation to be succesful.

The environment in which communication takes place in DAI systems is either shared global memory, message passing or some combination of both.

The most common environment used in DAI systems is the blackboard model (Hayes-Roth, 1985). In this type of environment, the black board represents the shared global memory on which messages are written by teams of agents, partial results are posted and necessary information is found. If only one Blackboard is used, the resulting system becomes unpractical for real DAI systems, because they produce bottlenecks due to limited bandwidth available (Hammon, 1984). Otherwise, if several blackboards are used (Lesser & Erman, 1980), the DAI system implemented is highly equivalent to a message passing system.

Message passing environments offer a more abstract means of communication than simple shared memory, and their semantics are well understood. These environments are mostly supported by using object oriented languages (Booch, 1991). In such languages, the high level encapsulation units for communication are abstract data-types called objects. Objects incorporate local domain knowledge, plans, variables and procedures, and interact by "sending messages" to one another. Message passing offers a more abstract means of communication than simple shared memory. Message passing is easier to program, modify and expand, because it is more abstract, and more efficient if we exploit the locality of data.

Communication costs must be minimal. This is mostly achieved by using appropriate protocols. There are many possible policies and protocols for communication in distributed problem solvers. The traditional distributed processing literature is a rich source of information about protocol details. In

DAI systems, we are concerned with high level protocols, used for problem solving applications. Cammarata (1983) describes policies in selection of recipient Agents and error handling. Selection falls into two categories:

1- Selective communication: happens when messages are targeted to a specific receiver Agent (either sent directly or indirectly throgh a black-board). On-demand communication Happens when messages are sent at the request of one Agent to another.

2- Broadcast-communication: Happens when messages are sent and they can be read by any one Agent within the system. Communication errors are handled by acknowledgements, but this is costly in terms of the available bandwidth.

One of the most studied protocols for distributed problem solving communication, is the Contract-Net-Protocol (Smith, 1980) (considered in the next chapter to be appropriate for conceptual design problem solving of instruments), which is based on the Cooperating Experts Metaphor. The individual message types used in the Contract Net combined with their expected responses yield a general two way interaction among agents. A contract is an agreement relating to any task to be allocated, so this protocol has quite a general applicability. The interaction supported by this protocol elegantly provides a two-way transfer of information, potential for complex information transferd in both directions, context dependent local evaluation by individual agents, and symmetric mutual selection.

4- Organized activitv via modellinq other aqents:

Meaningful interaction between two agents requires that they have at least implicit knowledge of each other, such as the knowledge encoded in a communication protocol or language. Therefore, an agent has to have some model of other agents inside itself. This yields several complex problems. One of the problems is how an agent predicts the next actions of other agents based on the information it has aquired. Another problem is how an agent plans its actions by taking the actions of others into account.

Communication is minimized when each Agent within the problem solving system has a model of other Agents in terms of their addresses, states, skills and beliefs, and therefore it only needs to communicate when its model of the environment is incomplete or incorrectly reflects its environment.

As was discussed before, in DAI systems, an agent must coordinate with other agents. Coordination is essential for avoiding harmful interactions and because local decisions have global effects. In early DAI research, Wesson & Hayes-Roth (1981) reported that one of the most important principles he devised involved the use of models to simulte and predict other nodes. He used this approach by means of Process Assembly Network (PAN) in which models evolved in a manner specified by the proposing Agent.

The advantage of this approach is that once all Agents have been informed of a model (Hypothesis), no further routine communication needs to take place, only when Agent models are incomplete or something unexpected happens, Agents need to communicate to others in order to update their models.

Genesereth & Ginsberg (1986) also write that systems can cooperate without communicating if they have good models of one another. This works best when the goals of the Agents are not conflicting.

When communication takes place between different Agents, different models of rational behaviour lead to different plans. More importantly, building up models about rationality of other Agents can lead to cooperation between Agents that do not communicate and might have antagonistic goals.

Dufree et. al. (1985), also, hypothesise that "...nodes can reduce the amount of meta-level communication by making intelligent predictions about the activities of other nodes based on information they already have, instead of exchanging more information....". His experiments have shown that the exchange of information consisting of plans of other Agents, is especially useful in cases where the domains of the nodes overlap considerably because it helps them to avoid redundant work .

Work done by Gasser et. al. (1987), on the developement of the MACE system, suggests a modelling approach for coordination.

In this developed system, which is a instrumented testbed for building a wide range of experimental DAI systems, every Agent has its own model of the outside world . This model includes knowledge of the names, addresses, roles, skills and plans of other Agents. In this system the key idea is that each Agent observes its outside world and models other Agents.

Sueyoshi & Tokoro (1990), following the same line of thought, add the following functions to an Agent (in a distributed cooperating problem solving system) in order to make the problem solving process by autonomous Agents more adaptive:

1- Observing actions of other Agents
2- Dynamic modelling of tasks (action rules) of other Agents
3- Predicting actions of other Agents by using the model

The key idea behind the above improvements is that, it enables each Agent to model other Agents within the system in its constructed My-World attribute. My-World constructed by each Agent is different from others, since each has different relations to other Agents which it needs in order to solve its problem. An Agent can then plan its action by simulating its tasks within its constructed My-World attribute and predict the actions of other Agents. In this way, its actual task changes reflectively by observing the action of others.

2.3.3 Applications of DAI in Engineering Design:

There has been a great deal of research on DAI in recent years. Much of the research, on Distributed Problem Solving, has been done by Artificial Intelligence researchers for solving particular problems. Testbeds for this research has been air traffic control studies (McArthur et. al., 1982), vehicle monitoring (Dufree et. al., 1987; Corkill & Lesser, 1983) and medical diagnosis (Gomez & Chandrasekaran, 1981). For the most part, this research is being done in domains where information comes from spatially distributed sensors and problems are solved by cooperative exchange of information among the Agents.

DAI techniques have been used to solve a range of engineering design problems. Yang et.al. (1985) has implemented an architecture for a DAI system, focusing on control and communication aspects. He has tested his architecture for the design of digital logic ~~design~~ circuits. Problem solving in this system occurs by an iterative refinement of several mechanisms, including problem decomposition, kernel subproblem solving and result synthesis.

Problem decomposition in this architecture occurs according to the mechanism stored in the metaknowledge data-base of the message receiving node as suggested by the application developer.

Lander & Lesser (1991) describe a cooperating experts framework (CEF) developed to support cooperative problem solving among sets of knowledge-based systems with limited knowledge about each other's local states. The framework has been used for the design of steam condensers. CEF is implemented in a blackboard environment. Its global blackboards are used to facilitate communications among agents. Any information placed on these blackboards must be represented in a common language shared by all agents. Problem decomposition in this system is hard-wired by the programmer for a specific application. The developer uses application knowledge contained in the domain specific objects and black board spaces: In the steam condenser design the object Agents include pump, heat exchanger and motor objects. In this framework cooperation is achieved by collection of non-local constraints and goals by each agent and conflict resolution strategies during the problem solving process.

The emphasis in the CEF has been on the methods by which the Agents solve sub-problems relevant to their specific expertise and integrate their efforts using conflict resolution strategies that are appropriate to the problem solving context.

R.J. Verilli et. al. (1988) have constructed a computational model called "Iterative Respecification Management" for solving mechanical design problems. This system consists of a static cooperative hierarchy that communicates via functionally accurate messages (specifications, comments and request for specification, and designs) and a protocol that only allows agents to communicate with their managers and subordinates and not with each other. In this model, managers have exclusive control over subordinate designer Agents and no direct communication is allowed among subdesigners except via hierarchy.

In their test-bed design problems are decomposed by manager nodes into separate sub-design problems and allocated to sub-designers. Problem decomposition in this system is predefined by the developer in the manager nodes. Conflict resolution is done via hierarchy and manager nodes have global views for the parts of design problems allocated to them.

In this framework, cooperation is achieved using a predefined problem decomposition and task allocation strategy, guided by the static hierarchical architecture of the framework.

Chandrasekaran & Brown (1989) have contrasted the "classical" view of expert systems with an alternative view in which a number of knowledge sources, each specializing in a generic type of knowledge-based reasoning, cooperate to produce a solution.

These knowledge sources are identified as:

1-Decomposition KS.
2-Design plan instantiation KS.
3- Design expansion KS.
4-Design modification KS.
5-Constraint satisfaction KS.
6-Goal/Constraint generator for subproblems KS.
7-Design verification KS.
8-Design criticism KS.

Based on this view, they have designed a framework for routine type of design problem solving.

This framework has been tested for design of Air-Cylinders. Again problem description and decomposition have been defined statically using a conceptual design hierarchy for the Air-Cylinder. This conceptual design hierarchy remains fixed during the problem solving process.

From the above review, It becomes evident that in the design domain almost always the general form of the artifact being designed is known by the DPS system. In these systems, the focus of research has been on conflict resolution strategies and the evaluation of variable parameters of the artifact, but problem description and decomposition has been provided by the developers prior to initiation of the problem solving process. In these frameworks, the DAI system is developed by assuming the existance of a static conceptual hierarchy of a design problem and the way to decompose a design problem is already known. The Contract Net (Smith, 1980) and its successors all have addressed flexible opportunistic allocation of tasks but decomposition of tasks were provided by the developer. Similarly, Actor systems (Kornfeld et. al., 1981) have considered dynamic task allocation decisions, but have not treated description or decomposition problems.

2.4 Conclusions

In this chapter, we have been concerned with the application of A.I. techniques for engineering design problems.

There is general agreement among researchers as to what constitutes the overall phases of the design process.

Conventional CAD tools are primarily aimed at providing assistance towards the later stages in design, and are well established. The early stages of the design process is characterized by synthesis processes, and A.I. techniques provide viable means for its automation.

Methods suggested by design and A.I. researchers, for the design synthesis, include: decomposition-recomposition, cased-based reasoning and constraint satisfaction. However, the decomposition methods are the core processes for design synthesis. Other design synthesis methods assume a pre-existing problem decomposition. Based on this observation, three types of design problems emerge:

1- Routine design
2- Innovative design
3- Creative design

Design problems, by nature, are parallel, distributed processes and are best implemented using DAI techniques. DAI systems have a multitude of advantages as compared to conventional rule-based systems. These include: naturalness, reliability, efficiency, resource sharing, extensibility and cost-effectiveness. However, complex research issues must be resolved when a distributed approach is used.

These issues are classified into four categories:

1- Formulation, decomposition and allocation of problems
2- The methods for achieving distributed control
3- Communication processes for coordination
4- Modelling other agents

Specifically, there has been relatively little research in problem formulation, decomposition and task allocation issues. This has influenced current approaches to the design of DAI systems.

In these systems, almost always, there exists a predefined static conceptual hierarchy of a design problem and the way to decompose a design problem is provided by the developer. These systems support a routine type of problem solving.

Non-routine type of design requires the generation of an extensive number of possible decompositions into subfunctions providing alternative synthesis plans for the overall design process.

In the next chapter, we will investigate the above issues further. The purpose of these investigations is to implement a Distributed Problem Solver that supports a non-routine type of design problem solving for measurement instruments.

CHAPTER 3

A Distributed Problem Solver for Conceptual Design of Instruments

3.1 Introduction

In recent years, development of computer aids based on artificial intelligence techniques, for conceptual design of engineering systems, has become an active area of research (Adeli, 1988; Topping, 1989; Gero, 1990).

In the initial stages of the design process (Non-routine type of design problem solving), the general forms of the artefact, its functions and their corresponding decomposition into sub-functions are not known apriori. There might be an extensive number of possible decompositions providing alternative synthesis plans for the overall design process.

The analysis of requirements leads to the establishment of the overall functions to be performed by an instrument system. At conceptual stage of design, alternative solutions are generated. The concepts are expressed in the form of general description of components defined by common principles of operations. The design details, such as dimensions and material properties of the component, are not considered at this stage of the design process.

The objectives of the work reported in this chapter has been to develop a flexible intelligent system for the conceptual design of instruments using Distributed Artificial Intelligence (DAI) techniques. DAI is concerned with solving a problem by applying both artificial intelligence techniques and multiple problem solvers. Multiple intelligent problem solvers (Agents) co-ordinate their knowledge, skills and plans to act or solve problems. They work towards a single goal or separate individual goals that interact. The key obstacle at conceptual design stage is the decomposition of the functional requirements for the design into sub-functions. Having achieved suitable decompositions, solutions for sub-functions are determined. Problem decomposition is not a well understood process. It is easy to recognise when it is done well or badly, but there are relatively few principles that can be used prospectively to produce good decompositions.

There has been relatively little research on automatic problem decompositions using AI techniques. In almost all the AI based systems problem decomposition is provided by the developers prior to the initiation of the problem solving process. In Distributed Design Problem Solving systems almost always the sub-problem solvers are hardwired in the system by the developers. This was confirmed during the literature survey of the previous chapter.

The implemented DPS system, for the conceptual design of instruments, consists of a community of Agents - an agent provides expertise on a particular aspect of the problem or solution of a sub-problem. An agent may be a complete expert system in its own right. The DPS system, for conceptual design of interments, consists of two types of Agents:

(a) Functional agents
(b) Instrument subsystem agents

The subsystem agents deal with the solution of sub-problems, while functional agents are mainly concerned with the task of management and co-ordination. A solution for a problem is achieved through co-operative negotiations among the agents. Co-operation between agents is based on the Contract Net (CNET) approach [3]. CNET models task sharing or opportunistic allocation of tasks among the agents in a system. The design problem is communicated to the entire community of the agents in the system. If a subsystem agent can assist in the solution (partially or completely), it becomes part of a dynamic organization which gradually builds an overall solution of the design problem. An agent that can only partly contribute towards the solution will approach other agents in the system to complete its task, which in turn may require further assistance. The process continues until the problem is decomposed, distributed, and solutions for sub-problems achieved. Finally, partial solutions are combined together to generate new design concepts.

Excessive negotiations, between agents, are controlled through implementation of concepts introduced by MACE (Gasser et. al., 1987), whereby a functional agent has knowledge of capabilities of suitable agents (acquaintances) in the system. During inter-agent negotiations, acquaintances are approached first, if the acquaintances are not able to contribute towards the solution of the problem, then other agents in the system are interrogated. The details of an agent able to assist in the solution of the problem are added in the acquaintance-list if it does not already exist.

This system proves the suitability of a task-sharing type of co-operation in conjunction with a result-sharing type of co-operation (Smith & Davis, 1981) for design concept generation of instruments. The implemented system exploits the complementary nature of these two forms of co-operation.

3.2 Conceptual Design of Instruments

The conceptual design of a device involves development of ideas to solve the design problem. Abstraction, at this stage, is essential so that many alternative solutions may be generated. Feasibility of these solutions is considered at the next stage of the design where unsuitable solutions may be ruled out.

A systematic methodology for generating design concepts involves (Mirza et. al., 1990b):

1-Decomposition of the overall functions into sub-functions.
2-Mapping of sub-functions to physically realizable sub-systems.
3-Suitable combinations of sub-systems to generate new concepts.

The kernel of the design task is then the generation of sub-systems and/or concepts for functions that cannot be further decomposed. Different approaches to concept generation for sub-functions may be grouped into:

(a) Convergent methods.
(b) Divergent methods. (Finkelstein & Finkelstein, 1983)

The initial steps of systematic concept generation methodology are those of problem decomposition and abstraction. The system to be designed is decomposed into components, each of which is considered in terms of its function. Abstraction is essential in the process of identification of the component sub-functions into which the overall functional behaviour can be resolved. At the core of the process is the need to generate concepts for the elementary component functions. Alternative methods suggested for the purposes are: exploration of existing design concepts, the use of analogies, the transformation of existing concepts, and the convergent generation from systematic listing and examination of physical laws (Finkelstein & Finkelstein, 1983). Variant concepts for the total system are obtained from the combination of sub-function concepts (Mirza et. al., 1990a). Evaluation of the generated candidate solutions leads to a design configuration that will be the object of more detailed design.

The work reported in this chapter is based on overall knowledge of functionality of instrument subsystems as explained in the following sections. Alternative techniques for the solution of subproblems are discused elsewhere (Mirza & Finkelstein, 1992). The overall framework of the Distributed Problem Solver is general and has been developed with a view to future expansion.

For the instrument design our approach has been to classify subfunctions according to the basic operating principles of the component. Associated with these are functional knowledge concerning the inputs and outputs characteristics of the corresponding sub-system component. They can be either measurands, signal carrying variables, or parameters which may be used in the transduction chain when component combination is performed.

Using above methodology, a DPS (Distributed Problem Solving) system was implemented that uses dynamic problem formulation and decomposition for the purpose of conceptual design of instruments which will be explained in the following sections.

3.3 Problem formulation and decomposition in DAI

The main difficulty in the conceptual design process is the decomposition of the overall function into subfunctions. It is not possible to decompose a problem without prior knowledge of sub-function solutions, at the same time a sub-problem can not be solved unless it has been defined first. This exemplifies the recursive nature of the problem that imposes difficulties in the development of CAD tools to for the initial stages in engineering design.

Decomposition choices are critically dependent on how a problem is described. This is due to the fact that, the description of a problem prescribes a view point to tackle it.

Problem description is the source from which the collection of problem solver agents and their attributes are identified, providing a framework for expressing inter-agent dependencies.

Examination of the distributed AI literature (Bond & Gasser, 1988; Gasser & Huhns, 1989; Huhns, 1987) reveals that, a significant proportion of existing co-operative systems are built to solve particular problems. These systems use techniques which are only appropriate for their application domains (e.g., air traffic, vehicle monitoring & speech recognition).

The Contract Net (Davis & Smith, 1983) and its successors all have addressed flexible opportunistic allocation of tasks; decomposition was provided by the developer.

Similarly in the DVMT (Distributed Vehicle Monitoring system) (Durfee et. al., 1988), the descriptions, partitioned knowledge for data interpretation and regional responsibilities have been generated by designers, the system itself makes semiautonomous, opportunistic allocation decisions about which nodes perform which particular aggregation of tasks.

Actor systems, also, have considered the dynamic task allocation decisions, but have not treated description or decomposition problems (see Kornfeld & Hewitt, 1981).

In general, there must be a choice among alternative task decompositions, depending on the ability of the Agents performing the tasks. Key research questions include how we construct or select a set of operators for the task set to be produced, and how we construct and decompose problems so as to minimize the costs of computing, management and development of knowledge distribution and resource allocation.

Difficulties in decomposition arise because of dependencies among sub-problems and among the decisions and actions of separate Agents. In real settings, for a particular problem, there are a multitude of problem descriptions/formulations which will require different representations for the problem as well as decision on the boundaries of the problem and what is known and unknown. A problem representation must be selected that is well defined and covers all aspects of distribution and allocation across the multi-agent system.

In summary, intelligent approaches to task decomposition must consider:

1-An effective representation of sub-problems/tasks to be allocated to the resources and capabilities of different Agents. This means that the developer has to make decisions about the alternative types of problem decomposition. A problem decomposition must be selected that takes into account an efficient knowledge distribution among resources and a control strategy that produces effective solutions.

2-The decomposition method must take into account the dependencies among tasks/sub-problems and among the decisions and actions of separate Agents.

There has been little research in above areas and this has influenced the current approaches to the design using DAI techniques.

3.4 The DPS system for Conceptual Design of Instruments

At the conceptual design stage, problem decompositions and sub-problem descriptions are not known in advance. Therefore, in order to support automated problem description and decomposition, dynamic control is needed to support dynamic organisational structuring. In the conceptual level of design problem solving little can be said, a priori, about any individual problem, and agents need to negotiate over appropriate responsibilities for the description, decomposition and allocation decisions. In this way, the expert agents can configure conceptual design organisations supporting conceptual hierarchies of alternative artefacts for the same specification of requirements. This process will support the initial stages of design problem solving, including dynamic problem description and conceptual problem decomposition (not supported in current design frameworks).

Smith, working with Davis introduced concept called "Contract Net Protocol". The activity being modelled by Contract Nets is task-sharing (Davis & Smith, 1983). Task-sharing is a form of co-operation in which individual nodes assist each other by sharing the computational load for the execution of subtasks of the overall problem. Control in systems that use task-sharing is typically goal-directed; that is, the processing done by individual nodes is towards the achievement of sub-goal problems whose solutions can be integrated to solve the overall problem. Contract Nets give us the best opportunity for dynamically decomposing problems because they were designed to support task allocation. In other work, Smith and Davis has discussed in a general sense about co-operative frameworks (task-sharing and result-sharing) in DPS systems (Smith & Davis, 1981), with no attempt to relate them to the area of design problem solving.

In this section, an implemented DPS system is described which can be considered as an extension of Smith and Davis's work. This system proves the suitability of a task-sharing framework in conjunction with a result-sharing framework for the design concept generation of instruments.

On the basis of the methodology introduced in section 3.2, a DPS system was developed which simulates dynamic design problem formulations and decompositions, supporting the design of instruments at the conceptual level of abstraction. In order to develop the software, first a classification of instrument sub-systems was developed. The classification of instrument sub-systems was the source from which a co-operative community of organisations of problem solver agents was identified and constructed. On the basis of the developed architecture, a DPS framework was implemented which supports both forms of co-operation (task-sharing and result-sharing).

The stages of the design and implementation of the DPS system are explained in the following sub-sections.

3.4.1 Classification of Instrument Sub-Systems

The present work has been concerned, in the first instance, with the design of sensor elements and sub-systems, and discussion here is concerned with classification applicable to them. Classification of sensor elements and sub-systems is generally based on their physical principles of operation, (e.g. resistive, inductive, elastic elements). Alternatively, sensing elements are classified according to the type of measurand. Both classification systems have been used in instrumentation literature for the description of instrument systems (Neubert, 1975; Doeblin, 1983). A more fundamental classification system, based on power-flow models of instrument sub-systems, has been developed by Finkelstein and Watts (1978). This classification is based on mathematical similarities between instrument sub-systems of different energy domains, and leads to a consistent method for developing mathematical models of complex instrument systems.

In the DPS developed, instrument subsystems are grouped together according to their energy domains. This classification is based on Finkelstein's (Finkelstein & Watts, 1978) classification system but, mathematical similarities are not considered at this level of abstraction. The resulting classification is shown schematically in Figure 3.1 below:

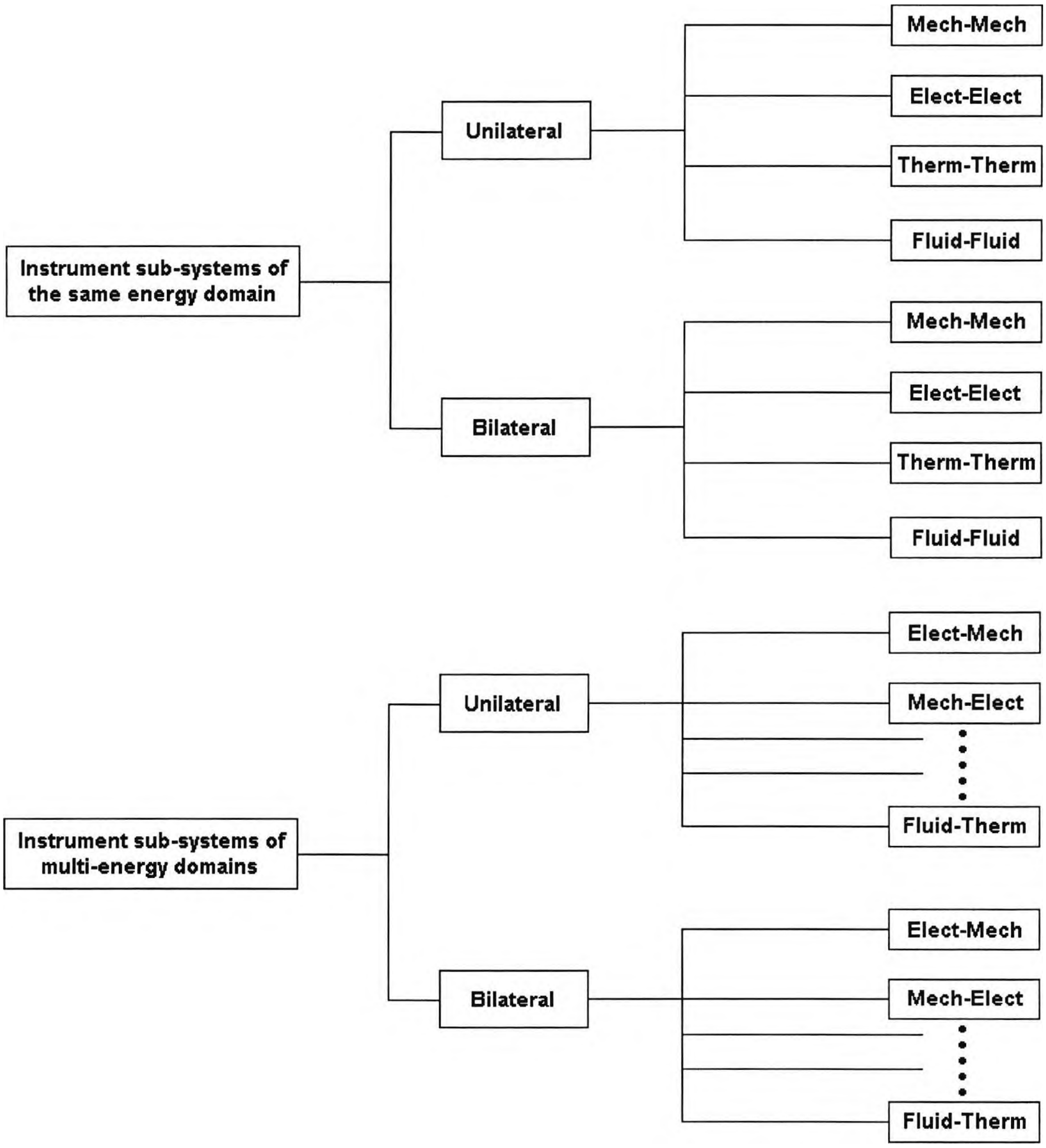


Figure 3.1 The Classification of Instrument Sub-systems

In this classification, instrument sub-systems are grouped according to their common functional characteristics.

Instrument sub-systems that belong to the same energy domain are grouped into:

1-Unilateral: These sub-system instruments convert input functional variables into their output functional variables only in one direction, i.e., from input to output.

2-Bilateral: These sub-system instruments convert input functional variables into their output functional variables from either of their ports. They include Transformer and Gyrator elements.

The instrument sub-systems, contained in an energy domain organization, deal with input-output functional variables belonging to the same domain.

Instrument sub-systems that belong to different energy domains are similarly grouped into Bilateral and Unilateral elements. These instrument sub-systems deal with input-output functional variables belonging to different energy domains.

3.4.2 The DPS system Architecture

In order to support automated problem formulation and dynamic problem decomposition for conceptual design of instruments, the derived classification of instrument sub-systems was used to produce a co-operative community of organisations of Agents. This was achieved by:

1-Identifying each energy domain as a functional-agent which has appropriate design heuristics and knowledge for coordination of sub-system elements that belong to its local model of the overall design environment.

2-Identifying each sub-system element (i.e., a primitive non decomposable instrument sub-system), as a member of an organisation of sub-system elements that belong to a particular energy domain. Each sub-system agent represents a knowledge-based expert system with problem solving methods for detailed self-design that can include anything from numerical optimization to inferencing capabilities using appropriate sets of design heuristics. The sub-system elements are sub-ordinate to their functional-agents.

The overall architecture of the DPS system is shown in Figure 3.2 below:

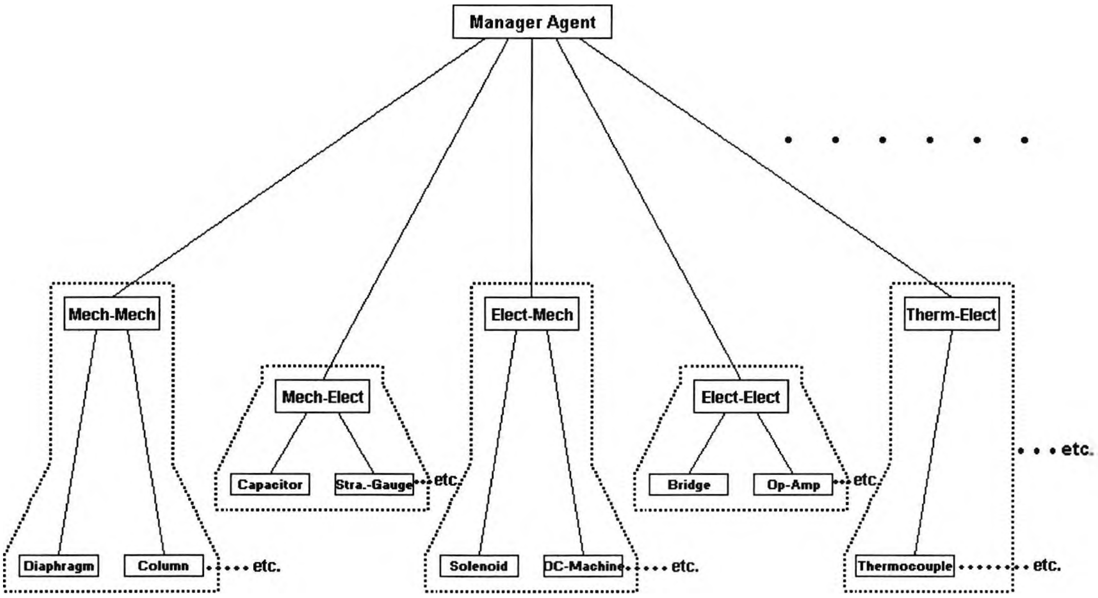


Figure 3.2 The Control Structure of the DPS system

In order to support co-ordination among problem solver Agents, the ideas introduced by the MACE (for Multi-Agent Computing Environment) system, concerning the organisations of Agents, acquaintances models and self-model qualifiers and their role in the problem solving process, have been implemented in the software framework.

MACE is a language, programming environment and a test-bed for DAI systems(Gasser et. al., 1987). In the MACE system, every Agent has its own model of the outside world (env_model). This model includes knowledge of the names, skills, goals and plans of other Agents.

According to the ideas introduced within the MACE framework, Each energy domain functional_agent was designed to contain the following attributes:

1-Self-Model: The Self-model attribute, for each agent, represents domain knowledge and capabilities.

2-Env-Model: The env-model attribute is a dynamic data-structure that is created and completed gradually as a functional-agent receives information-providing messages from its own sub-system Agents or other energy domain organisations.

The env-model attribute is a list of information units (representing models of suitable candidate subsystem Agents) to be completed and updated whenever an information providing message is received by a functional-agent.

Each information unit, contained within the env-model of a functional-agent, at this stage, consists of:

1-The identity of the candidate sub-system agent (being modelled) that belongs to a functional-agent organisation.

2-The relevant skills of the sub-system agent (i.e., its input-output characteristics).

3-A list of the sub-system agent's compatible sub-system agents that can be connected to its input to form parts of overall design concepts. These compatible sub-system agents will either belong to different energy domain organisations or the same energy domain.

4-A list of the sub-system agent's compatible sub-system agents that can be connected to its output to form parts of overall design concepts. These compatible sub-system agents will either belong to different energy domain organisations or the same energy domain.

The env-model is created and expanded according to the number of sub-system elements participating in the design activity within an organization. Each functional-agent, when receiving an information requiring message(i.e. the <s_need> Message type, see section 3.4.4) from other organisations, will first interogate its env-model to see if complete models of suitable sub-system agents already exist within its model of its organisation. If it finds compatible sub-system agents within its model, it will send information providing messages back to the message sending organisation. Otherwise, it will invoke negotiations down the organizational hierarchy to find suitable sub-system agents. This feature increases the efficiency of the net as each functional Agent, while completing its model of its organisational members, will learn about their capabilities and will not have to negotiate again when similar requirements are needed.

The organisational feature,also, helps to have a more efficient distribution of design goals among sub-system element agents that belong to a particular energy domain. This feature also simplifies the program expansion. The framework is simply expanded by adding more subsystem agents to their appropriate energy domain organisations.

3.4.3 Functioning of the Implemented System

It is useful to construct a model for the phases of a distributed problem solver. The problem solving process occurs in four phases (Smith & Davis, 1981):

1-Problem decomposition
2-Sub-problem distribution
3-sub-problem solution
4-Answer synthesis

The DPS system is developed based on a CNET (Davis & Smith, 1983) type of negotiation, during which the overall top-level design goal is co-operatively decomposed and allocated among a network of problem solver agents, which represent different knowledge based expert systems (i.e, a community of heterogeneous agents).

This phase of the problem-solving process is analogous to task-sharing in a conventional CNET framework, with the difference that the problem description and decomposition is implicitly and co-operatively achieved, because sub-problems are highly dependent on each other (as opposed to a conventional CNET framework in which task decomposition is the responsibility of a manager node and it is assumed that sub-problems are relatively independent).

During the sub-problem design solution phase, the agents with allocated sub-problems will have to share their partial results to come up with a complete design solution. A result-sharing approach (Smith & Davis, 1981) is used which, also, include answer-synthesis as a natural continuation of the sub-problem solution phase. During answer-synthesis, the partial results (local solutions contained within each problem-solver agent) are synthesized to achieve a solution or sets of alternative solutions to the overall design problem.

The informal strategy, as derived in previous section, is used to construct an specification of the DPS system. The class diagram of the DPS system is shown below:

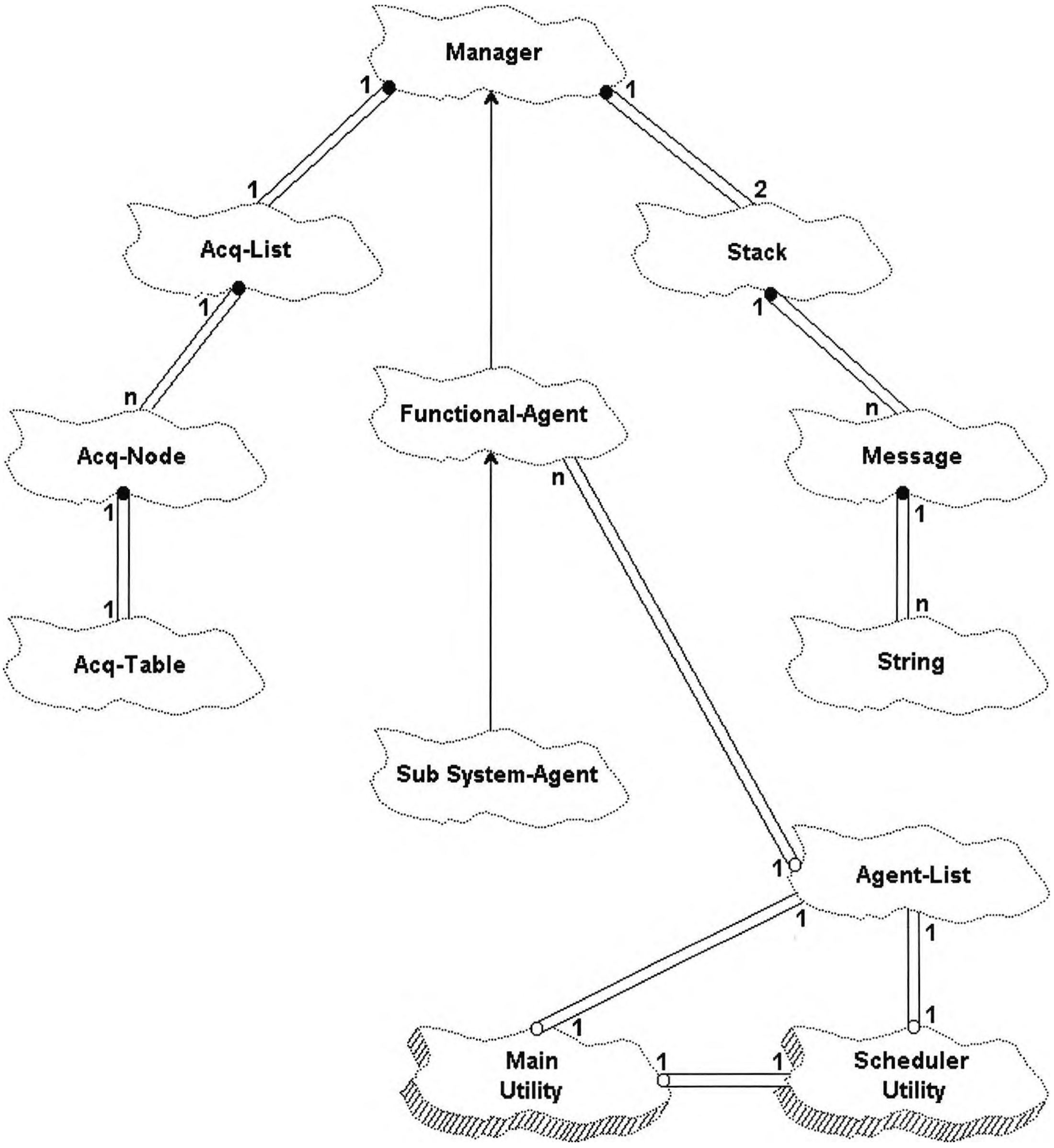


Figure 3.3 Class Diagram of the DPS system

C++ programming language has been used in the development of the software system. The class diagram of the DPS system has been developed, using Booch's (Booch, 1991) Object Oriented Design Methodology (figure 3.3).

The class components of the system, representing the key abstractions of the problem domain, are described below.

1-The Manager-Agent is the interfacing agent to the main utility. It encapsulates appropriate methods for distribution of the user specified design goals across the network of energy domain functional agents. It is invoked by the main utility.

2-An energy domain functional agent class represents an energy domain organisation. It can encapsulate appropriate heuristic knowledge and inferencing capabilities related to the overall design problems and sub-system elements that belong to its dynamic local model of the overall design goals.
Each functional-agent object instance has appropriate methods for communication processing. These methods support inter-organisational and intra-organisational communications and are overloaded by sub-system agents. The communication processing methods are overloaded by sub-system agents to support polymorphism.

During the task-sharing processes, each functional agent will accept or refuse the allocated design problems. If committed, it will enter into negotiations with its sub-system organisation members and other energy domain organisations. Functional agents will update their env-model attribute instances according to the environmental messages received.

3-A sub-system agent class represents a primitive non-decomposable instrument. All instantiated sub-system agents must include expert knowledge for self-design problem solving that can include anything from numerical optimization to inferencing capabilities using relevant sets of design heuristics.

An instantiated sub-system agent belongs to a particular energy domain organisation as represented by a functional agent.

As shown in Figure 3.3, there is an inheritance relationship among the manager agent, functional-agents and sub-system agents. This is a much more efficient implementation as compared to a flat structure of agents. Useful attributes and methods, if needed, are inherited down the inheritance hierarchy. This feature prevents the need for the creation of repetitive code. It also promotes modularity and ease of future expansions of the problem solving network.

4-The Acquaintances-Model class is instantiated for each energy domain functional-agent, and is a private attribute. It supports the overall structure and methods of the env_model qualifier for each functional-agent instance and is inherited to sub-system agents. The env-model attribute allows agents to incorporate detailed models of the behaviour and capabilities of their acquaintances and reason about their actions.

5-The Message-Queue class represents incoming and outgoing message queues of agents. The incoming message queue contains messages addressed to an agent. Messages are generally queued in the order in which they arrive.

Before receiving at least one message, an agent is dormant. When a message arrives for an agent, the agent is activated by the scheduler utility. It is up to the agent to interpret and respond to the received messages using its own particular message processing methods. The out-going queue contains messages to be transmitted to other agents within the problem solving network.

6-The Agent-List is the container class of energy domain functional-agents. It has appropriate access methods used by the scheduler utility. It represents the overall problem solving network of energy domain organizations.

7- Both communication and parallelism are simulated in the system by means of the scheduler utility. It controls message exchange by accessing the Agent-List container object instance. It allocates processing time for agents to process their incoming queue of messages and transmission of their outgoing messages.

8-The Main Utility represents the interface of the problem solving net. It asks the user for the required design problems and transmits the design problems via messages to the manager-agent. It, also, initiates the scheduler utility. If further details and limitations for a particular design concept, suggested by the system, is requested by the user, it will invoke the appropriate agent for requested information.

The overall functioning of the implemented DPS system can be described by

the following algorithm:

1- Set up all of the Agent organisations within the framework.

2- Ask the user for the required functional description (input-output characteristics) of the design concepts to be generated.

3- Invoke the manager agent to distribute the allocated top-level design goal among suitable energy domain functional-agent candidates.

4- Invoke the energy domain functional-agents to start negotiations with their sub-system agents. The purposes of these negotiations are:

- To find suitable sub-system elements that can completely or partially satisfy the overall design goal.
- To build up initial acquaintances-models of suitable candidate subsystem elements, in terms of their characteristics.
- To build output-queue information-requiring messages to be sent to other organisations.

5- While there are functional-agents with output-queue messages to be processed, do:

- Schedule a functional-agent with output-queue message(s) to be processed.
- Send the output-queue message(s) to its destination functional-agent(s).
- Invoke the destination functional-agent(s) to process the received message(s). This process involves either of the following:

a) The destination functional-agent has received an information requiring message. If complete acquaintance models exist, the destination functional-agent uses its model of its organisation to send back information providing messages. Otherwise, it will invoke negotiations down the hierarchy to complete its model of suitable candidate sub-systems before sending back a message.

b) The destination functional-agent has received an information-providing message. It uses the received message to update its env-model qualifier.

6) At the end of step 5, all energy domain functional-agents will have complete local models in their env-model qualifier, in terms of their own suitable sub-system agents, their characteristics and their immediate compatible sub-system elements. Invoke a recursive answer synthesis mechanism, within the network of functional-agents, to arrive at the overall conceptual designs. This is done using the partial solutions contained within each functional-agent's env-mod qualifier. During this process, functional-agents will share their partial solutions to arrive at complete conceptual designs.

7) Ask the user, if extra detail, concerning the characteristics and feasibility for a chosen concept, is needed.

-If the user needs such information, invoke the appropriate agents to display details of the chosen conceptual design.

8) Ask the user if he/she wishes to start a new design problem.

-If the answer is positive loop back to step 2.

-Otherwise exit the program

Above functional algorithm is serially processed. This means that, although several functional_agents within the net might have several output_queue messages to be processed, each time, the scheduler function will invoke only one Agent for processing its output_queue messages. A parallel processing hardware capability will, substantially, increase the processing time of the DPS system. However, if a concurrent system is to be used, more sophisticated message passing mechanisms must be employed to preserve the semantics of the objects within the framework in the presence of multiple threads of control.

The task-sharing framework (Smith & Davis, 1981) is used based on negotiations supporting the problem formulation, dynamic co-operative problem decomposition and sub-problem distribution phases of the problem solving process as described below:

The overall Top-Level design goal, in the form of input and output requirements, is communicated via messages to the Manager Agent. The Manager Agent (see Figure 3.2) is responsible for allocating Top-Level design goals to the overall problem-solving net. In order to do so, it will invoke negotiations with the energy domain functional-agents by sending messages containing abstract descriptions of the overall design goal. These descriptions are in terms of the input-output power, effort and flow relationships.

Functional agents have abstract knowledge about the capabilities of sub-system agents. This information is encoded in the self_model qualifier of the functional agents. On the basis of this knowledge, the design goal is either accepted or refused by the functional agents.

If the design goal is accepted, the functional agent negotiates with its sub-system agents to determine which elements can partially or completely satisfy the design requirements. If a sub-system agent can contribute towards the solution of the problem, its identity and capabilities are stored in the env_model of the functional agent.

In cases where a sub-system agent can only partially satisfy the design goal, it sends messages to other functional agents requesting assistance to complete the design goal along with the addresses of preferred functional agents. The sub-system agents can inherit these addresses from the env_model qualifier of their energy domain functional agents or build their own environment models independently. The functional agent will invoke negotiations with the selected functional agents in the system. If the identities of compatible functional agents are not known, then the entire community of functional agents is approached. A functional agent, upon receiving request for assistance from other agents, will start to negotiate with its own organizatinal member sub-system agents.

The process is repeated until negotiations are exhausted terminating the task-sharing phase. Task-sharing leads to the establishment of a number of models within each functional agent's env_model qualifier; consisting of the description of instrument sub-systems, their input/output characteristics, and addresses of compatible sub-systems from other energy domains. This aspect supports a dynamic problem decomposition, whereby Agents within the net cooperate to find all possible combinations of primitive sub-system elements which when synthesized will satisfy the overall design goal.

Sub-system organisation members, that belong to a particular energy domain, represent a particular primitive, non-decomposable instrument transducer. For example, sub-system elements that belong to the organisation represented by the Mech-Mech functional-agent (see figure 3.2) consist of springs, gears, levers, diaphragms, columns, beams, etc.

The above idea is based on the insight that an instrument system is composed of a number of sub-system instrument transducers. Each sub-system primitive instrument is only connected to a few neighbouring sub-system elements and does not have to know about the overall functionality of the instrument system (within which it is connected) to perform its own function.

As sub-problems allocated to each sub-system agent will be highly dependent, during the sub-problem solution phase, the functional-agents will have to share their organizations' partial solutions to synthesize complete conceptual designs. In this phase of the problem-solving, a result-sharing (Smith & Davis, 1981) approach is used which must also include answer synthesis. In the implemented DPS system, in order to find different design concepts, information from the env_model of the functional agents, involved in the negotiations, needs to be extracted and synthesized. A search is initiated, starting with the functional agents which satisfy the input requirements. Extraction of the first instrument subsystem leads to the identification of the next functional agent to be searched. The process is recursively invoked until all the combinations of instrument subsystems that satisfy the design requirements are found. Evaluation of the generated candidate solutions leads to design configurations which will be the object of more detailed design.

3.4.4 The Communication protocol

A common language is required to allow processing agents to communicate their intentions and share information with each other for effective coordination. A protocol is also required for correct communications. The protocol is a set of rules that specifies how to synthesize meaningful and correct messages.

Elements of the common language serve as words do in a natural language such as English. The protocol has the same function as grammar in a natural language.

The message types designed are very simple and their purpose is to distribute the design requirements of the manager Agent across the problem solving net as explained below: A backus-Naur (BNF) specification of this protocol is given below:

```
<Message>     ⇒  <Addressee> <Originator> <Text>
<Addressee>   ⇒  {Address}
<Originator>  ⇒  {Address}
   <Text>     ⇒  <Have?>|<Have>|<I-have>|<S-have>|
                 <S-need>
```

As shown above, the <text> non-terminal of a message is decomposed into several message categories. Each message category corresponds to a particular message-type that is generated during the negotiation process. These message categories (contained within the text of a message non-terminal) are further decomposed into:

```
<Have?>  ⇒ HAVE?
           {Message_Body}
<Have>      ⇒ HAVE
           {Message_Body}
<I_have> ⇒ I_HAVE
           {Message_Body}
<S_have>⇒ S_HAVE
           {Message_Body}
<S_need>    ⇒ S_NEED
           {Message_Body}
```

Non-terminal symbols are enclosed by "< >" (a non terminal symbol can always be substituted by either a set of non-terminals, terminals or a combination of both, depending on the rewrite rules specified), terminal symbols are written without delimiters. Slots that are to be filled with information encoded in the Common Internode Language are enclosed in "{ }". The terminals, shown above, are part of the common internode language filling the Message_Type attribute for each message.

The language needed in which to represent the information in the slots of a message is specified in a single relatively high level language in which all such information is expressed. This high level language is called the Common Internode Language (Davis & Smith, 1982). This language forms a common basis for communication among all the nodes and supports the task-sharing phase of the DPS system.

In the implemented software, the Message_Type and Message_Body non-terminals, as described in previous section, are to be filled with such a high level language understood by all the Agents within the system.

In the developed framework, message categories are divided into two groups:

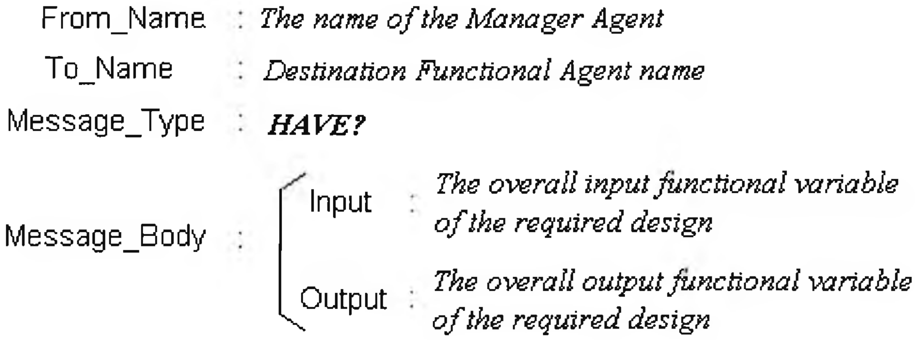
1-Information Requiring Messages:

These messages are sent when:

1-1) The manager node (The interfacing Agent) distributes the overall top-level design goal among its sub-ordinate energy domain functional-agents, by sending the following message type:

<Have?>

This occurs during the first cycle of the problem solving process. This message type has the format shown in Figure 3.4.



From_Name : *The name of the Manager Agent*

To_Name : *Destination Functional Agent name*

Message_Type : ***HAVE?***

Message_Body :

- Input : *The overall input functional variable of the required design*
- Output : *The overall output functional variable of the required design*

Figure 3.4

During the negotiations within an energy domain organisation, functional-agents will send similar message types to their sub-ordinate sub-system agents to enquire about their suitability to co-operate towards a particular allocated top-level design goal. This message type is analogous to a Task-Announcement message in a contract net framework.

1-2) Information is required by a subsystem agent which can only partially satisfy the design goal allocated to it. When this happens, the sub-system Agent, in order to find its compatible sub-system elements existing within other energy domain organisations, will send the following message type to its organisational functional-agent:

<s_need>

The receiving functional-agent, when scheduled by the scheduler utility, will inform the destination energy domain organisation about the requirement of its organization member. The format of this message category is shown in Figure 3.5.

From_Name	:	Org_Address	: *The name of the sender Functional Agent*
		Sub_Org_Address	: *The name of the requesting sub_system Agent*
To_Name	:	Org_Address	: *Preferred destination Functional Agent name*
Message_Type	:	***S_NEED***	
Message_Body	:	Skill	: *The sub_system element skills*
		Input	: *Required input variables*
		Output	: *Required output variables*

Figure 3.5

This message type illustrates the possibility of horizontal communications across the hierarchy (i.e., communications among two contractor sub-system agents that belong to different energy domain organizations) .

2-Information Providing Messages:

These messages are sent when:

2-1) A particular sub-system element can fully satisfy the design requirement allocated to it. In this case, it will inform its functional-agent by sending the following message type:

<Have>

The functional-agent, when receiving the above Message_Type, will update its env-model qualifier according to the message content of this message. This message type is analogous to a Bid message in a contract net framework. The format of this message category is shown in figure 3.6.

From_Name : *The name of the sub_system element*

To_Name : *The functional Agent of the sub_system element*

Message_Type : ***HAVE***

Message_Body :
- Input : *The input characteristic of the sending sub_system Agent*
- Output : *The output characteristic of the sending sub_system Agent*

Figure 3.6

2-2) A particular sub-system Agent can only partially satisfy the design goal allocated to it. In this case, it will inform its functional-agent about this by sending the following Message_Type:

<I_Have>

This message category has the format shown in figure 3.7.

From_Name : *The name of the sub_system element*

To_Name : *The functional Agent of the sub_system element*

Message_Type : ***I_HAVE***

Message_Body :
- Input : *The input characteristic of the sending sub_system Agent*
- Output : *The output characteristic of the sending sub_system Agent*

Figure 3.7

It also will inform its functional-agent about its preferred energy domain organisations within which compatible sub-system elements belonging to other energy domain organisations can be found. This it does by sending an <s_need> message category (as illustrated in Figure-5) to its functional-agent as described previously.

2-3) A particular functional-agent might receive an **<s_need>** message type from another energy domain organization. In this case, it will invoke negotiations with its sub-ordinate sub-system agents to find suitable candidate elements (these negotiations only occur if a functional-agent's env-model of its organization is incomplete). A sub-ordinate sub-system agent, satisfying the partial requirement (needed by the organisation that initiated the <s-need> Message-Type), will inform its functional-agent of its suitability by sending the following Message-Type:

<s-have>

The format of this Message-Type is shown in Figure 3.8.

From_Name :	Org_Address :	*The name of the sender Functional Agent*
	Sub_Org_Address :	*The name of the sub_system Agent satisfying the requirement*
To_Name :	Org_Address :	*The name of the receiver Functional Agent*
	Sub_Org_Address :	*The name of the destination sub_system element that initiated the <S_NEED> message*
Message_Type :	***S_HAVE***	
Message_Body :	Skill :	*The input characteristic of the destination sub_system element*
	Input :	*The input characteristic of the sender sub_system element*
	Output :	*The output characteristic of the sender sub_system element*

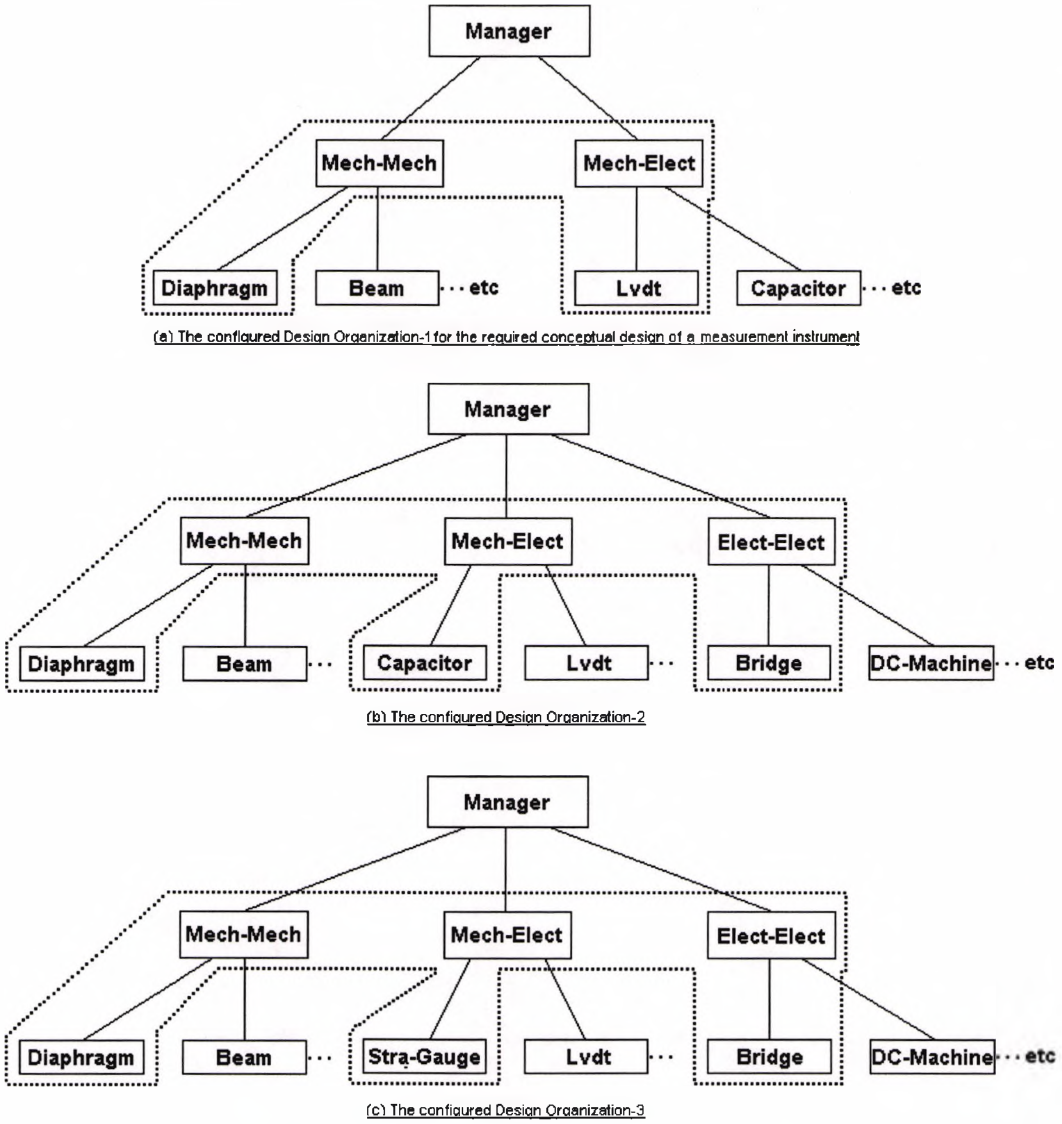
Figure 3.8

It must be noted that, different design organisations are configured, for particular user specified design requirements, during similar negotiations.

For the software documentation of the implemented DPS system please refer to APPENDIX-I.

3.4.5 An Example

For the purpose of illustration, we consider a simple problem. The DPS system is required to generate design concepts for the measurement of pressure with voltage as the output. Using a DPS system, consisting of four energy domain functional-agent organisations (i.e., Mech-Mech, Mech-Elect, Elect-mech and Elect-Elect), 25 possible design organisations were dynamically configured. Some of these organisations are shown in figure 3.9 below:



(a) The configured Design Organization-1 for the required conceptual design of a measurement instrument

(b) The configured Design Organization-2

(c) The configured Design Organization-3

Figure 3.9 Some of the configured design organizations

Above configured design organisations support the following conceptual plans:

1-Pressure --> Diaphragm --> Displacement
Displacement --> Lvdt --> Voltage

2-Pressure --> Diaphragm --> Displacement
Displacement --> Capacitor --> Capacitance
Capacitance --> Bridge --> Voltage

3-Pressure --> Diaphragm --> Displacement
Displacement --> Strain_Gauge_Bonded --> Resistance
Resistance --> Bridge --> Voltage

As illustrated in figure 3.9, These design organisations are dynamic organisations which span through sub-sets of organisation members belonging to different energy domain organisations. This aspect illustrates the possibility of horizontal communications across the hierarchy (i.e., communications among two contractor sub-system agents belonging to different energy domain organizations), as well as via more traditional (vertical) communications between manager functional-agents and their organization member sub-system agents. Sub-system agents, that belong to a particular configured design organization, can identify their best compatible elements by using their env-model attribute.

For each of these design-organisations, abstract models were produced within each functional-agent's env_model qualifier. These are models representing their own suitable sub-system element agents and their compatible neighbouring sub-system elements. For example, an abstract model for the bridge sub-system element (existing within the env-model qualifier of the Elect-Elect functional agent) co-operating with other sub-system agents, as shown in the configured org-2 (Figure 3.9), is shown in figure 3.10 below:

Name : *Bridge*

Characteristics : Input : *Capacitance*
Output : *Voltage*

Inconnects : *Capacitor*

Outconnects : *Output*

Figure 3.10 An information unit within the Env-Model of the Elect-Elect Functional Agent

Above model is a partial-view of the overall design concept generated by the configured org-2 that exists in the Elect_Elect functional-agent's env_model qualifier. This model represents the bridge element as a candidate that can be connected to the capacitor element from its input and produces voltage as its output.

The capacitor element, itself belongs to the organisation of the Mech_Elect functional-agent which from its input (in the model existing within its functional-agent) is connected to the diaphragm. Diaphragm in turn belongs to the Mech_Mech organisation which (again in the existing env_model qualifier of its functional-agent) from its output is connected to the capacitor.

It is important to note that, the Elect_Elect functional-agent does not know about the capacitor's in_connect attribute within its model (i.e., it does not know about the capacitor's compatible input sub-system elements). Therefore, the Elect_Elect functional-agent has only a local view (model) consisting of the ways its bridge sub-system element can be connected to other sub-systems to produce at least a part of a complete overall design concept.

This means that each functional-agent will only have a local model of the overall design process and will have to share its local partial models with other functional-agents to improve its partial view. These answer synthesis processes are invoked after organisations are configured.

3.5- Discussion and Conclusions

Most AI systems are based on predefined knowledge of decomposition of a problem into sub-problems. At conceptual stage of engineering design, decomposition of the functional requirements of a design problem can not be assumed prior to the initiation of the solution procedures. Therefore, it is not possible to develop an AI based system with hard-wired interdependencies between sub-problem solvers. This chapter presented an account of a Distributed Problem Solving system developed for conceptual design of instruments in which the functional requirements are dynamically decomposed and allocated to suitable sub-problem solvers. Discovery of alternative decompositions is based on a task-sharing type of co-operation among the contracting agents.

The system has been able to determine sets of alternative conceptual design solutions from the specification of the input/output requirements of a measurement instrument. The solution principles consist of feasible combination of physical sub-systems or elements commonly used in instrument design.

At present, the DPS system does not have a comprehensive list of physical sub-systems. However, the framework is easily expandable. Sub-system agents and new energy domain functional-agent organisations can be simply added to the problem solving net. These new agents and organisations will simply listen to the Common Internode Language, and if they find design problems for which they are suitable, they will join in to form dynamic design organisations in which they can co-operate to form design concepts.

The problem decomposition and sub-problem distribution phases of the DPS system are analogous to a task-sharing type of co-operation. In most implemented task-sharing frameworks (see Davis & Smith, 1983) task decomposition is the responsibility of a manager node and it is assumed that sub-problems are relatively independent. In the implemented DPS system the problem decomposition is implicitly and co-operatively achieved (during inter-agent negotiations), because sub-problems are highly dependent on each other.

Although in the framework no explicit problem decomposition is done by contracting agents, it is believed that the assumption that sub-tasks might be highly dependent does not imply non-compatibility for a task-sharing approach (but we need a modified negotiation mechanism).

The argument is that in instrument design problems alternative decompositions will map into sub-problems which are highly dependent on each other. As during the conceptual design stage of the problem solving no apriori design plan is assumed, the sub-problem interdependencies can not be hard-wired into contracting Agents for the purposes of decompositions and sub-problem allocations. In fact, the sub-problem interdependencies are a key to the process of creativity in the initial stages of the conceptual design process and must be discovered during inter-agent negotiations. The outcomes of these negotiations are the emergence of interdependencies among design agents. These interdependencies are embodied within configured design-organisations, each supporting an alternative design synthesis plan.

In the instrument design domain, the number of alternative conceptual decompositions is high and there are no easy , formalizable heuristics to choose among them. As described before, a functional-agent must be able to select the best candidate sub-system Agents, using its expert knowledge and local model of the design environment (i.e., award tasks to the most suitable contractor sub-system agents within its organization). This choice depends on conflicting aspects of the design environment such as cost, weight, functionality, robustness, appearance, topology, ergonomic characteristics, performance and so on.

The DPS system supports a non-routine type of design problem solving, during which it is effective in proposing alternative conceptual designs and the user in evaluating them and making a selection. It is highly important to develop appropriate design inferencing capabilities for energy-domain functional-agents, enabling them to choose set(s) of the most appropriate candidate sub-system agents. These choices depend on a particular instrument specification. Also vitally related to this future development is the implementation of suitable conflict resolution strategies (Lander et. al., 1991). The conflict resolution strategies are needed, because the sub-system solutions for detailed self-designs are highly dependent.

Each sub-system agent represents a complete knowledge-based expert system with problem solving methods for detailed self-design that can include anything from numerical optimization to sophisticated inferencing abilities. Extensive research effort in this direction has been carried out (Mirza & Finkelstein, 1992).

The DPS system exploits the complementary nature of both forms of co-operation (i.e. task-sharing and result-sharing) as opposed to a conventional CNET framework (Smith, 1980) which only supports task-sharing or a Hearsay-II (Lesser et. al., 1980) type of framework in which only result-sharing is supported.

CHAPTER 4

Towards an Intelligent and Adaptive Distributed Problem Solver for Design of Instruments

4.1 Introduction

The importance of integrating multiple sources of knowledge in the design process has resulted in a major research effort towards the implementation of optimized DAI systems (sriram, 1987). A multi-agent system can only achieve on-line optimized performance by exploiting the complementary role of group learning during its problem solving activity. In this chapter our efforts are directed towards the investigation of suitable machine-learning techniques and their possible extensions into our multi-agent system.

The learning processes in a DAI system are more complex than those in a single agent. To apply learning, the agents in the DAI system need to adjust, adapt and learn from working with other agents in problem solving. More importantly, in a DAI system, the overall system is capable of achieving more tasks than the sum of tasks which can be individually achieved by agents. This phenomenon is referred to as "emerging intelligence" (Forrest, 1990). It is this emerging intelligence of DAI systems, that the group of agents collectively can offer something not available in the individuals, that makes DAI a powerful problem solving tool. The learning processes that go on among the agents are the key factor resulting in emergent intelligence.

As stated in chapter 3, most problem solving processes, handled in DAI systems, consist of four phases:

1- The decomposition of the problem into sub-problem tasks.
2- The allocation of the sub-problem tasks among the agents.
3- The solving of the sub-problem tasks by the assigned agents.
4- The integration of the solutions, obtained in phase 3, to obtain the global solution.

On the basis of this "task-sharing" form of cooperation, in chapter 3, we took up the implementation of a coarse-grained Distributed Problem Solver (DPS). The implemented DPS system uses a contract-Net type of negotiation for dynamic problem decomposition and sub-problem distribution phases of the problem solving. The bidding mechanisms, as supported by Contract-Net negotiations, have been shown to be an effective coordination mechanism for adaptive multi-agent problem solving (Davis & Smith, 1993).

The foundation of this approach rests upon the idea of contracts and sub-contracts in a market-like organization. The bidding processes, such as the ones used in Contract-Net protocols, introduces an element of competition and for recording an agent's performance. On the basis of this framework, each agent in the loosely coupled system bids for announced tasks and the best bidder is selected to be the contractor for task sharing.

A contract is an agreement relating to any task to be allocated, so this protocol has quite a general applicability. The interactions, supported by this protocol, elegantly provides a two way transfer of information, potential for complex information transferred in both directions, context dependent local evaluation by individual agents, and symmetric mutual selection.

We recall, from the survey of chapter 2, section 2.3.2 that, Fox (1981) has studied the relationship between organization theory and distributed systems. By viewing distributed systems as analogous to human organizations, Fox shows that concepts and theories germane to the management science field of organization theory can be directly applied to the design of DAI systems. Task complexity, uncertainty, coupled with resource constraints are shown to be important factors in deciding how a system is to be distributed.

Fox (1981) states that price-like systems evolve naturally as the problem solving organization grows, and becomes more complex. The price system eliminates all forms of control between units. All communication is contained in a contract to purchase some product or service.

Contracting assumes that a set of independent processes exists. Once processes enter into contracts, an organization is instantiated. Hence, contracting can be viewed as the dynamic creation of a system architecture. The next step in successive reduction of control and information flow is the introduction of competition. Competing approaches to goal achievement are allowed (in the marketplace) with many organizations available to achieve any goal. Hence, each organization persues its own goals which correspond to another organization's needs. This is the general market situation. Services are contracted for in the market place for short or long periods of time.

The market-type problem solving organizations are highly adaptive systems, capable of adaptive self-organization in the face of highly complex, uncertain, dynamic and ill-structured problem environments.

We recall that our implemented DPS framework (please refer to chapter 3) consists of a community of agents. An agent provides expertise on a particular aspect of the overall design or solution of a sub-problem. In this setting each agent might be a complete expert system in its own right. For example, an LVDT sub-system agent must have appropriate design heuristics, deep knowledge and analytical procedures for detailed design of an LVDT sub-component.

Our objective in this chapter is to investigate A.I. techniques which are promising for the design and implementation of adaptive DAI systems which are capable of improving their performance. At a coarse-grained level (i.e. at the level of inter-agent interactions), this is achieved by using task-sharing and result-sharing forms of cooperation. Task-sharing is implemented using the Contract-Net approach (please refer to chapter 3). At a fine-grained level (i.e. at a single agent setting), the overall performance of the DPS system hinges on the capabilities of each single knowledge-based agent. At this level, we are concerned with developing techniques enabling each agent to be adaptive. This means that, we must develop ways enabling each agent to improve its knowledge and skills, so that not only can the individual agents be better at their tasks, but the whole DPS system can also keep improving its performance as a result. This means that, we must extend machine learning techniques used for single agent systems, such as explanation-based learning, case-based reasoning, or inductive learning, to our multi-agent system, where one agent can learn by observing its problem solving environmnt. Our main goal in this direction is to simulate group learning processes often used in human decision-making situations, such as group induction and brain storming. This issue directly implies the investigation of learning capabilities within a single agent setting and its possible extension to the overall DPS system.

A critical review of machine learning methods developed for single knowledge-based agents is given in the next section.

4.2 Machine learning for Knowledge-Based Expert Systems

Learning is a many-faceted phenomenon. Learning processes include the acquisition of new knowledge, the development of motor and cognitive skills through instruction and practice, the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation.

In all learning processes, inductive inference is the central element. In contrast to deduction, the starting premise of induction are specific facts, rather than axioms. The goal of inductive inference is to formulate plausible general assertions that explain the given facts and are able to predict new facts. In other words, inductive inference attempts to derive a complete and correct description of a given phenomenon from specific observations of that phenomenon or part of it.

Virtually, all inductive inferences may be regarded in one sense as either generalizations or specializations. For example, category formation is a relatively complex form of generalization. It typically involves both the generation of novel rules and the clustering and strength revision of existing ones. Analogy, as it arises in the most sophisticated types of human reasoning, is perhaps the most complex type of knowledge modification procedure. At this level, it usually involves a substantial amount of generalization and specialization and depends, also, on the application or formulation of several categories.

A syntactic approach to inductive inference has been proposed by Nillson (1986). Nillson has suggested that constructing an inductive assertion from observational statements can be conceptually characterized as a heuristic state-space search, where:

- States are symbolic descriptions; the initial state is the set of observational statements.

- Operators are inference rules, specifically, generalization, specialization and reformulation rules.

- The goal state is an inductive assertion that implies the observational statements, satisfies the problem background knowledge and maximizes the given preference criteria.

In this work, specialization and reformulation rules are the conventional truth-preserving inference rules used in deductive logic. In contrast to them, the generalization rules are not truth-preserving but falsity preserving. This means that if an event falsifies some description, then it also falsifies a more general description. This is immediately seen by observing that $H \Rightarrow F$ is equivalent to $\sim F \Rightarrow \sim H$ (the law of contraposition).

Therefore, if $H \Rightarrow F$ is valid, and H is true, then by the law of modus ponens F must be true. Deriving F from H (deductive inference), is therefore, truth preserving. In contrast, deriving H from F (inductive inference) is not truth preserving, but falsity preserving. In other words, if some facts falsify F, then they also must falsify H.

Based on this theory, Michalski (1983) has developed an annotated predicate calculus (APC). The APC adds to predicate calculus additional forms and new concepts that increase its expressive power and facilitate inductive inferences used for classification of single and multiple objects. The APC has been implemented in the INDUCE program and applied to a problem from the area of conceptual data analysis.

This problem is concerned with inducing discriminant descriptions and characteristic descriptions of "cancerous" and "normal" cells. The program is given examples of "cancerous" and "normal" cells for this purpose.

The solution to the problem posed is obtained by a successive repetition of the "*focus attention* → *hypothesize* → *test*" cycle.

The "*focus attention*" phase is concerned with defining the scope of the problem under consideration. This includes selecting descriptions appearing to be relevant, specifying underlying assumptions, formulating the relevant problem knowledge and the type of description sought and the hypothesis preference criterion. This phase is supplied to the system by the user. It involves his/her technical knowledge and informal intuitions. The "*Test*" phase, examines the hypothesis and tests them on new data. This phase requires collecting new samples, performing laboratory experiments, and/or critically analysing the hypothesis. This phase, again, involves knowledge and abilities required from the user.

It is the second, the "*hypothesize*" phase, in which the INDUCE program may play a useful role: The role of an assistant for conducting a search for the most plausible and/or most interesting hypothesis. The search process is basically a heuristic search through a space of symbolic descriptions, generated by the application of inference rules to the initial observational statements (i.e. teacher generated examples of some concepts).

The major drawback of this approach is the substantial background knowledge that has to be provided to the machine before plausible inferences can be made. Background knowledge is neccessary to provide the constraints and a preference criterion for reducing the infinite choice to one hypothesis or a few most preferable ones constituting the goal description. The scope of the INDUCE program is limited and considers, only, the identification of discriminant and characteristic descriptions of single or multiple objects.

More important topics of inductive learning, such as learning from incomplete or uncertain information, learning from descriptions containing errors and discovering new concepts has not been addressed.

One of the most impressive research efforts on induction, performed by Lenat (Davis & Lenat, 1981), has yielded programs for generating concepts and heuristics for mathematics and other domains. His work is concerned with establishing new concepts or theories characterizing given facts. This type of inductive learning includes such topics as automated theory formation and discovery of relationships in data.

Lenat constructed a program, called AM, for this purpose. AM is a program which models one aspect of elementary mathematics research, developing new concepts under the guidance of a large body of heuristic rules.

The local heuristics communicate via an agenda mechanism, which is a global list of tasks for the system to perform and reasons why each task is plausible. A single task can direct AM to define a new concept, to explore some facet (property, slot, attribute) of an existing concept, to examine some empirical data for regularities, and so on.

Repeatedly, the program selects from the agenda the task having the best supporting reasons, and then executes it. Each concept is represented internally as a data structure with a couple dozen slots or facets, such as "Definition", "Examples" and "Worth".

Initially, most facets of most concepts are blank. There are 115 of these structured modules provided initially, each one corresponding to an elementary set-theoretic concept (for example, union). This provide an immense "space" which AM begins to explore, guided by a corpus of 250 heuristic rules. AM extends its knowledge-base, ultimately rediscovering hundreds of common concepts (such as, numbers) and theorems (such as, unique factorization). Some heuristics are used to select which specific facets of which specific concept to explore next, while others are used to actually find some appropriate information about the chosen facet. Other heuristic rules prompt AM to notice simple relationships between known concepts, to define promising new concepts to investigate, and to estimate how interesting each concept is.

The foundation of Lenat's work is based on the assumption of reducing the definition of the processes of creativity and discovery into the application of "known heuristics" away from known concepts. In other words, the discoverer is moving upwards in the search tree. He/She is not rationalizing how a given discovery might have been made; rather, he/she is moving outward into the unknown for some new concept which seems to be useful or interesting by applying "known heuristics" to "known concepts".

In the same way, to cope with the large size of the potential "search space" involved, AM uses its heuristics as judgmental criteria to guide development in the most promising direction (The process of inventing worthwhile, new (to AM) concepts is guided using a collection of few hundred relevant known heuristics).

Based on these assumptions, Lenat has proposed the investigations into a "theory of known heuristics" by averaging all the world's heuristics in a generalization/specialization hierarchy, catalogued according to some conjectural power curves for heuristics in terms of their utilities. The power curves of a heuristic represents the utility of that heuristic as a function of tasks being worked on.

Lenat consequently has represented a metaphorical view of the process of evolution in nature based on his assumptions (Lenat, 1983). He hypothesises that the processes of evolution in nature are also guided by heuristics encoded in the DNA structure.

From past efforts in the development of knowledge-based expert systems it is evident that some heuristics greatly reduce search effort but do not guarantee finding minimal cost paths. In these problems, we want to minimize some combination of the cost of the search path and the cost of the search required to obtain that path. Minimizing the resulting combined cost should be averaged over all the problems likely to be encountered.

However, in practice, for real world problems, the average combination cost is not used, because it is highly difficult to decide on ways to combine path cost and search effort cost. Also, it is impractical to define a probability distribution over the set of problems likely to be encountered. Therefore, almost always heuristic power is left to informed intuition gained from actual experience with the methods by human experts. This is an important flaw in Lenat's position. Even his intuitive metaphorical statements likening his AM program to evolutionary biological processes is based on "freedom from targets and/or goals" as he puts it.

This viewpoint causes his programs encounter the problem of "mud", which is Lenat's informal designation for uninteresting definitions and tasks. Mud sooner or later accumulates to the point that a system becomes totally involved in a round of tasks that contribute nothing to the expansion of concepts and heuristics. This is because, expansion in Lenat's system is driven only by internal criteria of how "interesting" the structures are to the system, not by any external and/or environmental constraints having to do with their pragmatic effectiveness in solving problems.

As mentioned previously, analogy is perhaps one of the most complex inductive processes. Because of the close relationship between everyday notions of analogy and similarity, several models of analogical reasoning in A.I. have been developed which use partial pattern matching. Algorithms like those developed by Winston (1980) were based on the assumption that a best partial match could be found by accumulating evidence for each of a number of possible object-to-object mappings between representations of two situations and then choosing the one that scored highest. In these systems, evidence for a match consisted essentially of the number of relational connections preserved between corresponding objects for a given alignment of objects. The object alignment that placed the largest number of relations and attributes in correspondence was considered the best match and thus the "correct" analogical interpretation.

Winston (1984) consequently has implemented a program called ANALOGY for learning physical descriptions from functional definitions, examples and precedents. ANALOGY works on the basis of functional descriptions to identify objects, and learn their physical descriptions.

His identification and learning system is based on the following synthesis steps (Implemented in the ANALOGY program):

1- Description of the object to be recognized in functional terms.

2- Show a physical example. This is done using a semantic net representation.

3- Show that the functional requirements are met by the physical description using precedents. Several precedents are usually necessary to show that all of the functional requirements are met. During this process, the matcher determines part correspondence using the links that populate the precedent and the problem. The matcher pays particular attention to links that are enmeshed in the causal relations of the precedent. The analogizer transfers causal constraints from the precedent to the problem. The problem is solved if links in the problem match the links carried along with the transfered causal relation.

4- Create a physical model of the functionally defined concepts. Generalization is achieved by learning from examples and near misses.

The ANALOGY program has been used in simple object recognition tasks.

This approach has several major drawbacks as a model for analogical learning. First, it presupposes that well-defined, bounded representational models of the situations in both the "base" (i.e., familiar) domain and the "target" domain are available as inputs. In a learning situation, however, the required prior representations of objects and relations in the target domain may be wrong or inconsistent with the analogy. If the domain is totally unfamiliar, there may not even be any fragments of a useful representation available.

The second problem, with theories based principally on the matching of descriptions, is that conceptual representations for real situations may contain many causal relations that don't take part in analogy. Winston's ANALOGY program suggests that attention to important relations such as those involving causal links, can reduce the number of links and the computational complexity of the matching process; but it does not mention which causal relations are important for a particular analogy and why they are important.

Gentner (1983) has outlined a syntactic cognitive model of learning from scientific or "explanatory" analogies involving some of the problems discussed above. The analogies considered by Gentner included such statements as:

- The hydrogen atom is like the solar system
- Electricity flows through a wire like water through a pipe

The model Gentner proposed for learning from such analogies, unlike those founded on pattern matching, did not require a full description of the target domain beforehand.

In her model, Gentner distinguishes between "attributes", which are one-place predicates, "first-order relations", which are multi-place predicates with objects as arguments, and "higher-order relations", which are multi-place predicates with propositions as arguments.

The syntactic claim is that in using an analogy, people are most likely to map higher-order relations, next most likely to map first-order relations, and least likely to map attributes. For example, in the analogy between atomic structure and a solar system, the target and the source share the higher-order relations indicated by causal relations between ATTRACTS and REVOLVES-AROUND predicates. However, attributes of the mapped objects, such as their absolute size, do not transfer.

The above mapping process, characterized as a preference for higher-order causal relations, is called by Gentner as the "systematicity condition". This principle states that a highly interconnected predicate structure - one in which higher-order relations enforce connections among lower-order predicates - is most likely to be mapped.

The systematicity-based matching, together with simpler partial matching processes, has been consequently used by Navinchandra (1988) for the purpose of case retrieval from the long term memory, in an implemented design problem solver (called CYCLOPS). In Navinchandra's system, analogical reasoning is controlled through a process of asking relevant causal questions in a given design problem and searching the KBS for cases that have related causal relations as a direct solution for the encountered problem. If no analoguos cases are matched, the causal questions are redefined by finding and addressing their causes and effects (This process is an attempt to simulate self-interrogation, as studied by researchers interested in the psychology of creativity (Osborn, 1953). The process is continued recursively until an analogous case is found (using partial matching or Gentner's systematicity based matching), in which case the causal structure of the source case is transferred to the target design problem (This process is similar to that used in Winston's (Winston, 1984) ANALOGY program). The transfered causal structure contains a relevant design action which must solve the target problem. The target design problem is deemed solved if all disjunctive causal design questions are matched (via partial matching and/or systematicity based matching). Navinchandra has applied CYCLOPS to a simple landscape design problem.

Although, systems such as the CYCLOPS design problem solver are highly useful for practical engineering applications, there remains some fundamental problems related to systematicity based matching which has to be resolved. For example, the failure of transfer of predicates such as HOTTER-THAN in the atomic structure analogy. Predicates such as HOTTER-THAN, YELLOWER-THAN, and so on are themselves causally related to an indefinitely large number of other propositions. These interconnected propositions, obviously, have little or nothing to do with our understanding of the analogy with the atomic structure, but the systematicity principle does not show why they are irrelevant.

Furthermore, Gentner's analysis seems to imply that the mappable propositions can be determined by a syntactic analysis of the source analog alone (since, the higher order causal relations are defined independently of their participation in an analogy). This results in the same information being mapped in all analogies involving a given source. This is a major problem, because the basis of an analogy is intimately related not only to the source but also to the target and the context in which the analogy is used.

An alternative view of learning by analogy has been proposed by Carbonell (1983). His approach is a direct extension of "Means-Ends Analysis" (MEA) problem solving method. His central hypothesis (influenced by Schank's (Schank, 1980) theory of human memory organization) is that, when encountering a new problem situation, a person is reminded of past situations that bear strong similarity to the present problem (at different levels of abstraction). This type of reminding experience serves to retrieve behaviours that were appropriate in earlier problem solving episodes, based on which past behaviour is adapted to meet the demands of the current situation.

The traditional means-ends analysis (MEA) problem space consists of (Newell & Simon, 1972):

- A set of possible problem states
- One state designated as the "initial state"
- A set of operators with known preconditions that transform one state into another state in
the space.
- A "difference function" that computes differences between two states (typically applied to compute the difference between the current state and the goal state).
- A method for indexing operators as a function of the difference(s) they reduce (such as the table of differences in General Problem Solver (Newell, 1963).
- A set of global path constraints that must be satisfied in order for a solution to be acceptable. Problem-solving in this space consists of standard MEA:

1) Compare the current state to the goal state and identify differences
2) Select an operator relevant to reducing the difference

3) Apply the operator if possible. If it can not be applied, establish a subgoal of transforming the current state into one in which the operator "can" be applied (Means-ends analysis is then invoked recursively to achieve the subgoal).
4) Iterate the procedure until all differences have been eliminated (that is, the goal state has been reached) or until some failure criterion is exceeded.

Carbonell argues that, during a reminding process an "augmented difference function" be used. The "augmented difference function" defines a similarity metric to retrieve the solution of a previously-solved problem closely resembling the present problem. The "augmented difference function" is similar to the "difference function" of a conventional MEA which apart from comparison of initial and final states of source and target, it also takes into account the operator-sequence differences and path-constraint differences during the analogical mapping process.

In this framework, reminding is only the first phase in analogical problem-solving, during which a previous similar solution path is identified (using the "augmented difference function") and becomes a state in the "analogy transform problem space". The differences that the problem solver attempts to reduce in the "analogy transform problem space" are precisely those computed by the similarity metric in the reminding process.

Carbonell introduces a number of operators such as General Insertion, General Deletion, Operator Reordering, and Solution Sequence Truncation to be used in the "analogy transform problem space" for reducing the difference between the reminded problem and the new problem.

The differences that the problem-solver attempts to reduce are indicated by the similarity metric (as computed by the "augmented difference function") and the overall process is an standard MEA problem-solving in the "analogy transform problem space".

Some serious problems with Knowledge-Based Expert Systems, developed based on MEA problem-solving method is the fact that such systems demand very specific information about the problem at hand before they can make any attempt to solve it. Carbonell's arguments imply that the source and target problems must be well defined: initial states, goal states, and allowable operators (associated with the differences to which they are relevant) must be fully specified. Without a clear description of the goal state, for example, it is impossible to compute the difference between it and the current state. Also, these problem solving methods function by the assumption that the domain of problem-solving is decomposable to the degree that each subgoal can be solved without knowledge of the other subgoals in the system.

Unfortunately, non-routine and creative design problem-solving are characterized as fuzzy and ill-defined sorts of problems that fall short of the well-defined ideal. In fact, in these problems any of the basic components (i.e. the initial state, the goal state, the allowable operators, and the applicable constraints) may be only partially known when a solution is attempted.

In contrast to syntactic approaches in the study of induction, a number of fine-grained distributed inductive learning systems have been proposed which are studied in the next section.

4.2.1 Fine grained DAI systems for the simulation of induction

From chapter 2, section 2.3, we remember that, fine-grained DAI approaches are concerned with describing higher mental functions and higher reasoning processes by reference to highly parallel collection of processes made up of very simple computing elements.

Up to now, two classes of fine-grained, inductive and distributed learning systems have been proposed:

The first approach is the well known connectionist paradigm. In a connectionist system, symbolic objects and concepts correspond to a sub-network of nodes with weighted links.

Connectionist systems are implemented in a network of neuron-like units and are known to exhibit emergent behoviours. Most workers in the field of neural networks regard the subject as an attempt to uderstand cognition as a property emerging from the interaction of connected units in a network. Artificial neural networks (ANNs), also called parallel distributed processes (PDPs) or connectionist models, are an attempt to simulate, at least partially, the structure and functions of brains and nervous systems of living creatures.

Generally speaking, an artificial neural network is an information processing system composed of a large number of simple processing elements, called artificial neurons or simply nodes, which are interconnected by direct links called connections and which cooperate to perform parallel distributed processing (PDP) in order to solve a desired computational task.

One of the attractive features of ANNs is their capability to adapt themselves to special environmental conditions by changing their connection strengths or weights. Learning or building the knowledge structure in PDP systems involves modifying the weights for a given structure of interconnectivity. Currently, the most popular learning algorithm, used in neural networks, is the back-propogation algorithm (McClelland et. al., 1987). The back-propogation network has been shown to be capable of implementing 6 approximations to a variety of mappings from R^n to R^m. The approximation achieved has been shown to be optimal in a certain least mean squared error sense. The back-propagation is important for multilayered networks with one or more hidden layers. The back-propagation algorithm is effectively minimizing the error, gradient descending in the weight space.

Unfortunately, research in neural networks has not identified an appropriate technique for the determination of the optimal network architecture for a given task and there are no comprehensive analytic solutions available.

The space being searched, that is, the error surface on which a minimum is to be found, depends on network configuration and the application. A particular network configuration assumes an specific connectivity pattern. For a fixed application, some network configurations do not guarantee a satisfactory solution, while others might lead to slower or faster convergence. The training process also depends on the initial weights; as with all gradient procedures, the process suffers from local minima problem.

Apart from the above disadvantages, the crucial weakness of connectionist systems is the near impossibility of simulating one-shot learning as characterized in simple skill acquisition processes. For example, if we need to add a new relation such as "owner of" to associate the entities say "John" and "Fido" in a semantic network or a rule-based representation of knowledge, nothing could be easier. We add an "owner_of" labelled arc between the nodes labelled "John" and "Fido" in the semantic network, or we add a fact (OWNER_OF FIDO JOHN) to our knowledge base. Unfortunately, we can not do this using a connectionist system, because there is no direct way of introducing this new relation and relate the pattern of activity for "John" with the pattern of activity representing "Fido". A trivial problem for the conventional syntactic knowledge representation approaches looks to be close to impossible (almost meaningless) for the connectionist approach.

Holland's (Holland, 1986a) Classifier Systems, apart from fruitful suggestions for solution of the above mentioned problems related to a connectionist approach, span both syntactic and connectionist approaches. Holland, Holyoak, Nisbett and Thagard (Holland et. al., 1986b) have introduced a general theoretical framework for the study of induction. They argue that, the central problem of induction is to specify processing constraints that will ensure that the inferences drawn by a cognitive system will tend to be plausible and relevant to the system's goals. Plausible inductions can only be determined with reference to the current knowledge of the system and the problem context.

In their view, the study of induction, is the study of how knowledge is modified through its use. Because of their theoretical emphasis on the role of the system's goals and the context in which induction takes place, they characterize their proposed theory as "pragmatic". In contrast most studies of induction reviewed above treat induction from a purely syntactic viewpoint, considering only the formal structure of the knowledge to be expanded.

These studies mostly leave out the pragmatic aspects. I.e., those concerned with goals and problem solving contexts. In the next section, we will review the most important aspects of this theoretical development.

4.3 A Pragmatic Theory of Induction

Holland, Holyoak, Nisbett and Thagard's (Holland et. al., 1986b) theoretical framework can be considered as a synthesis of previous research concerned with syntactic and pragmatic aspects of induction.

A summary of their theoretical framework concerned with fundamental characteristics of inductive systems is given below:

In their framework, general knowledge is represented by condition-action rules. These rules can vary enormously in the complexity of their conditions and actions, representing features that can range from elementary perceptual ones to highly abstract categories. The immediate actions of rules consist of the posting of "messages" internal to the system.

Rules can represent both "diachronic relations" (for example between current and expected future states) and "synchronic relations" (associations and recategorizations of categories). These two types of rules constitute the empirical rules of the cognitive system and they act together to generate inferences and solutions to problems. Problem-solving, in this framework, involves both diachronic search and synchronic categorizations of elements.

Rules can also represent inferences. Whereas, the function of empirical rules is to model the world, the primary function of inferential rules is to produce better empirical rules.

A controversial claim made by this framework involves the nature of inferential rules. This is a response to the body of evidence indicating that people are not able to make effective use of deductive rules characterized by the logic of the conditional, when reasoning about abstract symbols (Evans, 1982). However, peaple do not normally violate the logic of the conditional when reasoning about concrete events. This means that, deductive real world problems are, in fact, solved by means of highly general but not purely syntactic rule systems.

This position can be further clarified by considering that, in real world problems, goal expressions contain a significant amount of control information. For example, the clausal expression: $(A \vee B \vee C)$ is logically equivalent to any of the implications $(\sim A \wedge \sim B) \Rightarrow C$, $(\sim A \wedge \sim C) \Rightarrow B$, $(\sim D \wedge \sim C) \Rightarrow A$, $\sim A \Rightarrow (B \vee C)$. But, each of these implications carries its own, rather different, extra-logical control information, not carried at all by the clausal form. Real world problems are mostly solved by using these extra-logical control information as opposed to techniques used in conventional Theorem Proving Knowledge-Based Expert Systems which have to convert the goal expressions into clausal forms before any progress can be made.

In this framework, higher-order knowledge structures, such as categories, correspond to clusters of rules with similar conditions. Larger structures are, therefore, composed of more elementary building blocks.

Superordinate relations among categories and rules results in an emergent default hierarchy. Exceptional information about specific examples will tend to override default rules, with the consequence that imperfect general rules will be protected from mistakes by rules concerned with exceptions.

A set of synchronic and diachronic rules, organized in a default hierarchy, gives rise to an emergent mental model. The mental model guides behaviour and is used to generate predictions that serve as the basis for inductive change.

The process of model construction is viewed as the progressive refinements of a quasi-morphism as contrasted to isomorphisms in which each unique state of the world maps onto a unique state in the model. An isomorphic mental model is unreasonable due to the limitations of realistic cognitive systems and the complexity of realistic environments.

In general, the cognitive system will attempt to construct various simplified quasi-morphic mental models for achieving certain goals. In a quasi-morphism, a higher layer in the model, with its broader categories, provides default expectations (such as, fast moving objects slow down) that will be used to make predictions unless some exceptional category is signaled. An exception invokes a lower level of the model, at which a different model transition function is specified to capture the exception (such as, small, striped, fast-moving, airborne objects -e.g., a "wasp"- remain fast-moving).

The concept of a quasi-morphism captures several basic aspects of a pragmatic account of the performance of cognitive systems. First, its hierarchical structure allows the system to make approximate predictions on the basis of incomplete knowledge of the environment. Second, as the model is refined, rules that represent useful probabilistic regularities can be retained as defaults. Holland (1986b) has proved that a hierarchy of default rules with exceptions can represent knowledge more compactly (i.e. with much fewer total rules) than a system restricted to "exceptionless" rules.

In this theoretical framework, rules act in accord with a principle of multiple sources of knowledge. Those rules with their conditions satisfied by current messages compete to represent the current state of affairs and to guide thinking and action. But in addition to competing with each other, multiple rules will often act simultaneously to complement and support each other. Through summation of converging evidence, the system can use multiple sources of weak support to arrive at a confident conclusion.

Furthemore, induction is claimed to have two basic classes of mechanisms at its disposal:

1- Mechanisms for revising parameters such as the strength of existing rules.
2- Mechanisms for generating plausibly useful new rules.

A realistic inductive system must have these mechanisms for constructing higher order knowledge structures (representing mental models of the environment). In other words, the system must obey the principle of inductive adequecy. This means that, the system should contain no structures that could not have been produced by the inductive mechanisms of the system.

Mechanisms for generating new rules are constrained by triggering conditions that ensures new rules are likely to be useful to the system. Most particularly, inductions are guided by background knowledge about the variability of classes of objects and events and are triggered in response to the consequences of the use of current knowledge, such as failed or successful predictions.

Analogy, in this framework, involves "second-order" modelling. A mental model of the target problem is constructed by "modelling the model" used in a source problem. Analogy is therefore a devicefor performing categorizations in the absense of immediately applicable rules. In terms of the theoretical framework, analogy occurs when there are no diachronic rules immediately available to construct a path from the initial problem state to a goal satisfying state.

Among the rules executed early in the problem-solving attempt will be synchronic rules (both associative and categorical) that send activation to concepts that share components of the representation of the target problem. Therefore, multiple related knowledge sources become active during this process. A single shared property might result only in a small increment in activation for an associated concept; but several shared properties, will raise the activation level of an associated concept enough to allow it to direct further processing. This process ensures that source analogs that share multiple properties with the target will be activated.

However, the process is further constrained by retrieval of properties that are goal-related. In other words, a potential analogy is found when synchronic rules connect the initial target state to an initial state in a source domain, diachronic rules in the source domain connect its initial state to a subsequent state, and the latter state in the source domain is in turn connected by synchronic rules to the target goal.

The analogically derived model will be immediately subject to the inductive mechanisms such as strength revision and specialization.

It is important to note that this is a pragmatic approach to analogy. The usefulness of an analogy, like the usefulness of any mental model, is determined by pragmatic factors and the analogy is an attempt to construct a set of diachronic rules for the target problem that embodies a transition function adequate to achieve the goal.

Based on the above pragmatic characterization of inductive systems, Holland (1986b) has proposed a computational implementation of inductive systems embodied in classifier systems.

Classifier systems are general-purpose programming systems that use condition-action rules (classifiers) for the representation of their procedural knowledge. They are characterized as fine-grained DAI systems (Holland, 1986b; Shaw & Whinston, 1989).

A classifier system consists of (1) a finite population of fixed length condition-action rules called classifiers, (2) a message list, (3) an input interface, receiving messages from the environment, and (4) an output interface for affecting the environment.

A large number of rules in the classifier system can be active simultaneously. All rules are described in the form of Condition(s) → Action. The LHS is generally a conjunction of a number of conditions, each of which specifies the set of messages satisfying it. The action, is the message sent when the condition part is satisfied. Each condition is a string of length l on the alphabet {1, 0, # }. The LHS is satisfied, if and only if every condition matches some messages currently on the message list. An individual condition matches a message, if and only if for every 1 or 0 in the condition the same value occurs at the corresponding position in the message; "#" functions as a "don't care" symbol in a condition and matches unconditionally.

In classifier systems, two algorithms have been constructed to support inductive mechanisms:

The first is the Bucket-Brigade algorithm (Holland, 1986b) which enforces rule competition in classifier systems. This competition is in terms of a bidding process, that determines which of the rules with matched conditions will be activated on a given processing cycle. The larger the bid made by a rule, the greater its chances of becoming active. Specific parameters jointly determine the size of a rule's bid: past usefulness is represented by a numerical parameter called "strength". Relevance is a function of the "specificity" of the condition of the matched rule: the more detailed the rule's condition, the greater its specificity. By forming more specific rules, the bidding process dynamically constructs a default hierarchy in which specific exception rules tend to override more general default rules.

In effect the algorithm treats each rule as a middleman in a complex economy, its survival being dependent upon "making a profit" in its local interactions. In the long run, a rule makes a profit only if the rule is tied into chains of interactions leading to successful actions. In this way, the algorithm supports the accumulation of experience.

The process of generating plausibly useful new rules is supported by genetic algorithms which use genetic operators such as crossover and mutation to recombine past useful rules for construction of possibly better rules. Thus genetic algorithms exploit the accumulated experience of the system for the generation of new and better rules.

It is important to note that classifier systems by using the competitive bidding mechanisms (supported by the Bucket Brigade Algorithm) exploit a "task-sharing" strategy for distributed problem solving which is characterized by a competitive market system. This "task-sharing" form of cooperation takes place at a fine-grained level (i.e. at the level of rule interactions). Therefore, analysis techniques appropriate for further development of our DPS system can also be exploited for the analysis of classifier systems. more generally, mathematical techniques appropriate for the study of classifier systems holds many elements in common with the analytical techniques appropriate for the study of an adaptive DPS system supporting a market-like form of cooperation.

Specifically analytical techniques used in mathematical economy become directly relevant for both (Holland, 1986b). Price systems, in common with a cognitive system, exhibit (1) hierarchical organization, (2) retained earnings (strength) as a measure of past performance, (3) competition based on retained earnings, (4) distribution of earnings on the basis of local interactions of consumers and suppliers (the bucket brigade), (5) taxation as a control on efficiency, and (6) division of effort between production and research (exploitation versus exploration).

Mathematical economics deals with many of these processes and with suitable modifications, much of this mathematics is relevant to the study of classifier systems.

4.4 Conclusions

In this chapter, our main aim has been to investigate A.I. techniques for the simulation of adaptation and inductive processes at the single agent setting and incorporate them into our general DPS framework.

Researchers active in Resource Management and DAI (Davis & Smith, 1983, Fox, 1981) has pointed out that, market systems, representing a "task-sharing" type of cooperation, are especially suitable for task or resource allocation, because of the combination of their efficient use of communication activities and being able to atain good global performance by distributed control.

Fox demonstrates that, the price systems are evolved in response to increases in task complexity encountered by DAI systems.

During the survey of section 4.2, it was shown that most research concerned with induction consider only the formal syntactic aspects of the knowledge structures and ignore the pragmatic aspects concerned with goals and problem-solving contexts.

In contrast, Holland et. al. (1986b) have introduced a general theoretical framework for the study of induction, encompasing almost all aspects of inductive processes. This theoretical development is pragmatic in the sense that, not only it considers the syntactic aspects of induction, but also it emphasises the role of the system's goals and the context in which induction takes place.

Based on this pragmatic characterization of inductive systems, Holland (1986a) has proposed the classifier-systems as a computational framework for the study of induction.

Holland's classifier-systems (a fine-grained DAI system), spans both syntactic and connectionist approaches. This computational framework, (representing a cognitive system) incorporates the pragmatic aspects of induction by being environment-oriented in its problem-solving.

More specifically, the classifier system can be thought of as receiving information (through its input interface) about the current state of its environment, which is representing problems in terms of goals to be attained. In this context, the system "closes the loop" through the environment by receiving information from the environment and acting upon the environment to bring about goal related transitions.

The major inductive mechanism in classifier systems for accumulation of experience is the Bucket Brigade algorithm. This algorithm has the following characteristics:

(1) It treats the inductive system as a market, where each classifier rule is analogous to a middle-man.

(2) The classifier rules use a strength parameter as the hypothetical capital for granting contracts or charging services.

(3) It uses a bidding mechanism to determine task assignments and at the same time, updates the strength of the classifier rules involved.

(4) The bidding mechanism introduces an element of competition, since the strength would affect a rule's ability to bid.

Furthermore, classifier systems use genetic operators to recombine the genetic characteristics of well performing rules to produce plausible better rules.

Classifier systems, by using the competitive bidding mechanism (supported by the Bucket Brigade Algorithm) exploit a "task-sharing" strategy for distributed problem solving which is characterized as a competitive market system. This process takes place at a fine-grained level (i.e. at the level of rule interactions).

Therefore, by considering each agent in our DPS framework as a classifier-system module (i.e. a classifier system module represents an inductive knowledge-based expert system), a number of significant advantages ensue:

Firstly, we would take advantage of a computational inductive system which apart from its comprehensive coverage of most aspects of induction, is supported by a substantial amount of research into the nature of cognition and its pragmatics (Holland et. al., 1986b).

Secondly, by considering the agents in our DPS framework as classifier systems, instead of using a predetermined, procedural approach for the implementation of Self-Model and Env-Model qualifiers as suggested in the MACE programming testbed (Gasser et. al., 1987), we would use the message-list of the classifier systems to represent both. In this way, all external communication and internal processing is carried out by sending messages to the message-list. Therefore, for the distributed version of classifier systems, to be incorporated in our DPS framework, in addition to the messages from the environment and/or from the actions of executed rules, there is another source of incoming messages: the messages sent by other agents (i.e., other classifier system modules) through the network.

Thirdly, in this proposal we can, also, investigate the application of genetic operators, such as crossover and mutation, at the coarse-grained level of processing. It is relatively easier to apply genetic operators to a community of classifier system modules as compared to a community of syntactic or connectionist knowledge-based agents because, classifier systems use a chromosome-like syntax for the representation of their knowledge.

Lastly, the proposed organization for our DPS system is much more coherent in its problem solving than a community of conventional heterogeneous knowledge-based expert systems. This is due to the fact that in this proposal, the system, both at the coarse-grained level and at the fine-grained level of processing, uses a "task-sharing" strategy for cooperation, characterized by a competitive market system. Hence, as elaborated in section 4.3, there are possibilities for technology transfer from one level to the other or vice versa.

This coherence in concept significantly reduces the effort in the implementation and further expansions of this framework on parallel hardware.

We emphasize that, the proposed adaptive organization, to be incorporated into our DPS system, at this stage has only a theoretical value. In order to investigate the capabilities of this framework, we must first concentrate on the practical capabilities of Classifier Systems and Genetic Algorithms as applied to the design of instruments at the sub-component level.

In the next two chapters, we will examine these capabilities with some concrete instrument design problems.

CHAPTER 5

Genetic Algorithms for Design Optimization of Instruments

5.1 Introduction

In chapter 4, section 4.2.1, it was stated that genetic algorithms have been proposed to support inductive mechanisms for rule discovery in classifier systems.

Research on genetic algorithms has paralleled work in mainstream A.I. in the sense that simpler studies of search and optimization have preceded the more complex investigations of machine learning. This is due to the fact that search and optimization applications, with their rather well-defined problems, objective functions, constraints and decision variables provide a more tractable environment where alternative techniques may be compared easily. By contrast, machine learning problems, with their ill-defined goal statements, subjective evaluation criteria and a great number of decision options, constitute a highly complex environment not easily open to comparison or analysis. This motivates us to first study the genetic algorithms in the context of a number of concrete instrument design optimization problems.

In this chapter, we first concentrate on the foundations of genetic algorithms (section 5.2), their applications, advantages (section 5.3) and research issues (section 5.4). We, then, take up a comparative study of a number of reproductive strategies, in the context of two instrument design optimization problems: corrugated diaphragms (sections 5.5 and 5.6) and LVDTs (section 5.7).

Finally, we investigate a number of techniques for the purpose of multimodal function optimization using genetic algorithms (section 5.8). These studies are carried out for the purpose of finding alternative optimal designs, satisfying the same user specified design criteria.

5.2 Genetic Algorithms

Genetic Algorithms are search algorithms based on the natural evolution metaphor. Genetic algorithms have been developed by John Holland (1975).

These algorithms maintain a set of heuristics based based on chromosomal string structures (Problem Solving Heuristics) and evaluate the structures for selection. They use a structured yet randomized information exchange to create a new set of artificial structures, using bits and pieces of the fittest of the old (cross-over) and an occasional new part is tried for good measure (mutation). Genetic algorithms exploit the accumulated information within the generated population members to speculate on new search points with expected improved performance.

By working from a population, genetic algorithms maintain a rich data-base of well-adapted structures (Chromosome strings) from which new members may be created. By maintaining this diversity, These algorithms can reach different regions of a search space in parallel. Overall this algorithm is much more globally oriented than any conventional optimization method that searches from a single point (e.g. gradient-based optimization methods).

Genetic algorithms are reproduction plans composed of two types of transition rules:

1- Reproductive processes
2- Genetic operators

Reproductive processes determine the number of copies (Offspring) of an string to produce during a reproductive cycle (iteration). Genetic operators determine the modifications and combinations of these strings which will form the strings of the next generation.

Briefly, a reproductive plan operates as follows: Each population member in the search space is represented uniquely by a string generated from some alphabet. There is evidence that the alphabet {0,1}, i.e., a binary representation is optimal (Holland, 1975). The strings symbolise the "genetic material" with specific positions on the string (genes) taking on a variety of values (alleles). Each string has a performance index defining the concept of fitness for the members of the search space.

As a matter of notation, we consider a sequence of populations $A(t)$, where A represents a vector of strings and the index t refers to the time step. At any particular iteration cycle, a genetic algorithm maintains a population $A(t)$ of strings which represent the current set of search points being evaluated for fitness. A new generation *A(t+1)* of strings is generated by simulating the dynamics of natural evolution, as shown in Figure 5.1 below:

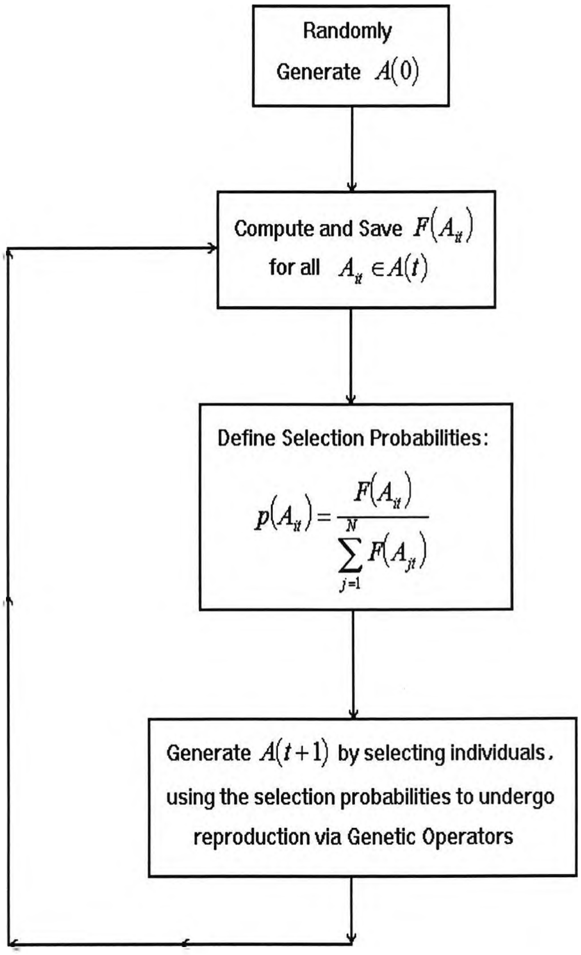

Randomly Generate $A(0)$
Compute and Save $F(A_{it})$ for all $A_{it} \in A(t)$
Define Selection Probabilities:
$p(A_{it}) = \frac{F(A_{it})}{\sum_{j=1}^{N} F(A_{jt})}$
Generate $A(t+1)$ by selecting individuals, using the selection probabilities to undergo reproduction via Genetic Operators

Figure 5.1 The Reproductive Plan

The process begins by randomly generating the initial population *A(0)*. Each individual in the current population is evaluated, saving its associated fitness value. A selection probability distribution is then defined over the current population *A(t)*. This probability distribution is denoted by $P(A_{it})$, $i = 1,2,.......N$. Finally the next generation *A(t+1)* is produced by selecting individuals via the selection probabilities to undergo "reproduction" via genetic operators. The selection probabilities are defined such that the expected number of offspring produced by a search-point is proportional to its associated fitness value. This can be viewed as the process of selecting individuals for reproduction as *N* samples from *A(t)* using the selection probabilities.

The expected number of offspring for individual A_{it} is given by :

$$E(A_{it}) = N \times P(A_{it})$$

$$= \frac{N \times F(A_{it})}{\sum_{j=1}^{j=N} F(A_{it})}$$

$$= \frac{F(A_{it})}{(1/N) \times F(A(t))} = \frac{F(A_{it})}{\overline{F}(A(t))} \qquad (5.2.1)$$

Where:

N	:	Number of strings in the population (population size)
$F(A_{it})$	:	Fitness of the ith. member at iteration t
$F(A(t))$	:	Sum of individual fitnesses
$\overline{F}(A(t))$	:	Average population fitness

Equation (5.2.1) indicates that individuals will produce offspring according to their fitness relative to the average fitness of the overall population. With no other mechanisms for adaptation, reproduction proportional to performance results in a sequence of generations $A(t)$, in which the best individual will eventually take over a larger and larger proportion of the population. However, in nature, offsprings are almost never exact copies of a parent. The genetic operators are introduced to exploit the selection process by producing new individuals which have high performance expectations. The choice of operators is motivated by the mechanisms of nature: Crossover, Mutation, inversion and so on (Frantz, 1973; DeJong, 1975).

We only use cross-over and mutation, used in conjunction with three operator genetic algorithms, which are shown to give good practical results.

Simple crossover consists of two steps. First, members of the newly produced generation are mated at random. Next, each pair of strings undergoes cross-over as follows: an integer position k along the string is selected uniformly at random on the interval $1 \leq k \leq l - 1$. Where l is the length of the string. Two new strings are formed by exchanging all alleles between positions 1 and k inclusively.

For example, considering two strings A an B of length 7 being mated at random:

$$A = a_1a_2a_3a_4a_5a_6a_7$$
$$B = b_1b_2b_3b_4b_5b_6b_7$$

The uniform random variable k can take any integer value between $1 \leq k \leq 6$. Suppose it takes the value of 4 (This process is analogous to the rolling of a die). The resulting crossover produces two new strings:

$$A' = b_1b_2b_3b_4a_5a_6a_7$$
$$B' = a_1a_2a_3a_4b_5b_6b_7$$

Therefore, cross-over is a randomized, but structured information exchange. A more intuitive argument can express what the crossover operator is actually achieving. A string can be considered as a complete idea or prescription of how to do a particular task. Substrings contain notions of what is important or relevant to the task. Therefore, the population can be considered as a Knowledge-Based System containing a multitude of different ideas and notions and rankings of these notions for a particular task. The cross-over process, combined with reproduction combines various notions of high performance strings to form new ideas. This is similar to the process of innovation. Most often innovation is a combination of things that have worked well in the past. Reproduction and cross-over are effective tools in search and recombination of useful genetic material, but during these processes potentially useful alleles might be lost. Mutation operator protects against these irreversible losses by using a random alteration of a string position. In a binary code, this simply means changing a "1" to a "0" and vice versa. Therefore the mutation operator is used to ensure against premature loss of important notions.

The above intuitive arguments are supported by more rigorous analysis of the working of the three operator genetic algorithms. The analysis is based on the similarity templates or schemata contained within each population member (Holland, 1975). A schema is a similarity template describing a subset of strings with similarities (i.e., similar notions) over certain string positions.

A similarity template is represented by appending the symbol # or wild card to the normal alphabet of the string. As an example, considering the strings of length 5, the schema #0000 describes the subset of strings: { 10000 , 00000 }. The schema #111# describes a subset with 4 members: { 01110 , 01111 , 11110 , 11111 }. In this way, schemata provide a straight forward means of describing all the similarity subsets possible within the strings of a given length. We note that in previous example, with a string length of 5 there are 3^5 = 243 different similarity templates, because each of the 5 positions may be a 0, 1 or #. In general for alphabets of cardinality k, there are $(k + 1)^l$ schemata (l being the string length). As a result a population of size N contains somewhre between 2^l and N×2^l schemata depending on the diversity of the population. According to Bethke's (Bethke, 1981) work three properties of a schema are important to consider. These are:

1- Schema order: This property defines the number of ones and zeros present in the schema (denoted by $O(h)$, h representing the schema). Therefore the higher the schema order the more specific it becomes.

2- The Schema's defining length: The defining length of a schema h, denoted by $L(h)$, is the distance between the first and last specific string positions and represents the schema's span.

3- The set of Defining Positions: This is the set of indices of ones and zeros in h, denoted by $\Delta(h)$. As an example, consider the following schema:

$$h_a = \#10\#\#\#1$$

The order of h_a (i.e., O(h_a)) is calculated by noting that the first defining position is at location 2. The last defining position is at location 7. The defining length is the difference, 7-2 = 5. The set of defining positions for h_a (i.e., $\Delta(h)$) is { 2, 3, 7 }.

Two distinct schemata, h and h', are said to be competing schemata, if they have the same set of defining positions. For example, let h = ##1#, h' = #0#1 and h'' = ##0#. Then $\Delta(h) = \Delta(h'')$ = { 3 } and $\Delta(h')$ = { 2 , 4 }. Schemata h and h'' have the same set of defining positions, and so they are competing schemata. The three schemata, which compete with h', are #0#0, #1#0 and #1 #1. Each complete set of competing schemata forms a partition of the search space.

From equation (5.2.1), under the genetic algorithm without genetic operators, the expected number of copies of a string S in the population at time t + 1 is directly proportional to the number of copies at time t and to its fitness value. Therefore, letting $N(S, t)$ represent the number of copies of string S at time t, then we have:

$$E\left[N(S,t+1)\right]=\frac{F(S)}{\overline{F}(A(t))}\times N(S,t) \qquad (5.2.2)$$

Letting $\hat{F}(h,t)$ represent the average fitness of all instances of h in the population at time t and let $N(h,t)$ be the number of instances of h in the population at time t. We can deduce that the same relation, as represented by equation (5.2.2), holds for all the schemata with instances in the population at time t :

$$E\left[N(h,t+1)\right]=\sum_{S\in h}\frac{F(S)}{\overline{F}(A(t))}\times N(S,t)$$

$$=\left(\frac{1}{N(h,t)}\times\sum_{S\in h}F(S)\cdot N(S,t)\right)\times\frac{1}{\overline{F}(A(t))}\cdot N(h,t)$$

$$=\frac{\hat{F}(h,t)}{F(A(t))}\cdot N(h,t) \qquad (5.2.3)$$

Where:

$\overline{F}(A(t))$: average fitness of the overall population
$\hat{F}(h,t)$: observed average fitness of h at time t

Therefore, from above equation, we deduce that under the action of reproduction alone, the number of schemata will grow or decline depending upon the ratio :

$$\frac{\hat{F}(h,t)}{\overline{F}(A(t))}$$

This growth ratio is directly related to whether a schema is above or below the current sampling average.

But, as we are using a three operator genetic algorithm, we must take into account the effects of cross-over and mutation in the above analysis. As mentioned before, simple crossover is executed by the random selection of a crossover site and the exchange of material across the site with the chosen mate. The probability of disruption of a schema due to cross-over is given by:

$$p_d = p_c \times \frac{l(h)}{l-1} \qquad (5.2.4)$$

where:

p_c : probability of executing cross-over
$l(h)$: Defining length of the schema
l : Length of the string

In other words a schema is destroyed if the cross-over site falls within its defining length. From equation (5.2.4) we observe that the higher definition schemata have a higher probability of getting corrupted.

Using equation (5.2.4) the survival probability of a schema due to cross-over becomes :

$$p_{cs} \geq 1 - p_c \frac{l(h)}{l-1} \qquad (5.2.5)$$

Note that this estimate defines a lower bound on the actual probability of survival due to cross-over because it does not include the probability of swapping identical defining positions between two strings.

Therefore using the cross-over operator is likely to badly disrupt the allocation of trials among the long definition schemata, but it is unlikely to distroy the short-definition schemata. However cross-over generates new instances of the schemata present in the current population as well as generating instances of schemata not already existing in the population. This is exactly the purpose of using cross-over. i.e., it combines high performance low order schemata to produce new schemata with expected above average performance.

The last operator to consider is the mutation operator. The mutation operator generates a new string by changing one or more bits of a string. The probability of mutating a given bit, denoted by p_m, is a parameter of the genetic algorithm which is mostly kept fixed during the course of GA runs.

The probability of survival due to mutation is related to the number of defining positions in a schemata (i.e., its order, O(h)). It is given by :

$$p_{ms} = \left(1 - p_m\right)^{O(h)} \qquad (5.2.6)$$

For small values of p_m (p_m << 1), the schema survival probability is approximated by the expression $1 - O(h) \cdot p_m$.

Equation (5.2.6) represents that the mutation operator is more likely to significantly distrupt the allocation of trials to high order schemata. Therefore, the cross-over and mutation operators will not distrupt appreciably the nearly optimal allocation of trials to the short definition schemata.

We can now deduce the combined effect of all three operators, reproduction, cross-over and mutation. The expected number of schemata h to survive into the next generation is the product of the expected number from reproduction alone and the survival probability of cross-over and mutation. This is given by:

$$E\left[N(h,t+1)\right] \geq N(h,t) \times \frac{\hat{F}(h,t)}{\overline{F}(A(t))} \times \left(1 - p_c \frac{l(h)}{l-1}\right) \times \left(1 - O(h) \cdot p_m\right) \qquad (5.2.7)$$

Equation (5.2.7) represents the fundamental theorem of genetic algorithms which defines a lower bound on the expected number of trials given to the existing schemata in a population. Therefore, short, low-order, and above average schemata receive exponentially increasing number of trials in future generations.

Holland and DeJong (Holland, 1975; DeJong, 1975), using the fundamental theorem of genetic algorithms and the k-armed bandit analogy, has shown that genetic algorithm is a near optimal procedure for searching among alternative solutions.

Holland (1986) also has observed that approximately N^3 schemata where N is the population size are usefully sampled in parallel during each generation of the genetic algorithm. This computation represents a large amount of data processing for populations of even moderate size (50-100). This process continues in parallel with the action of genetic operators applied to strings only. No explicit computation is necessary to correlate or trace the schemata development. This property of genetic algorithms has been called **implicit parallelism,** by Holland, because large number of schemata are handled simultaneously without explicit manipulation and centralized book keeping.

Holland (1987) has stated that genetic algorithms are the only known example of systems that exhibit implicit parallelism.

By processing similarities in this manner, a genetic algorithm reduces the complexity of arbitrary problems. Therefore highly fit, short, low-order schemata become the partial solutions to a problem, representing partial notions or building blocks for the solution of the task. The genetic algorithm discovers new solutions by speculating on many combinations of the best partial solutions contained within the current population.

5.3 Applications and advantages of Genetic Algorithms

The application of genetic algorithms in search and optimization has both tested and improved genetic algorithms, and has encouraged their successful application to search problems that have not given way to more traditional procedures.

The first application of a genetic algorithm, came in Bagley's (Bagley, 1967) pioneering dissertation. At that time there was much interest in game playing computer programs. Bagley devised a contrallable testbed of game tasks modelled after the game hexapawn. Bagley's genetic algorithm operated successfully on "diploid chromosomes" (paired strings) which were decoded to construct parameter sets for a game board evaluation function. The genetic algorithm contained the three basic operators - reproduction, cross-over, and mutation- along with dominance and inversion. At about the same time, Rosenberg (1967) was studying the simulated growth and genetic interactions of a population of single-celled organisms. His organisms were characterized by a simple biochemistry, a permeable membrane, and a classical, one-gene/one-enzyme structure. He introduced an interesting adaptive cross-over scheme that associated linkage factors with each gene, thereby permitting different linkages between adjacent genes. Rosenberg's work is sometimes overlooked by genetic algorithm researchers because of its emphasis on biological simulation, but its nearness to root finding and function optimization make it an important contribution to the research domain.

In 1971 Cavicchio (1970) investigated the application of genetic algorithms to a subroutine selection task and pattern recognition task. He used a genetic algorithm to search for good sets of detectors (subsets of pixels). His genetic algorithm found good sets of detectors more quickly than a competing "hill-climbing" algorithm. Cavicchio was one of the first to implement a scheme for maintaining population diversity.

Hollstien's (1971) work was the first to apply genetic algorithms to well-posed problems in mathematical optimization. The work is interesting in its use of allele dominance and schemes of mating preference adopted from traditional breeding practices. Hollstien's genetic algorithm located optima for his functions much more rapidly than traditional algorithms, but it was difficult to draw general conclusions because he used very small populations (pop-size = 16).

Frantz (1973) studied the effect of positional nonlinearities (epistasis) in genetic algorithm optimization. He constructed combined linear-nonlinear functions over binary haploid chromosomes and studied the positional effect (linkage) of several functions where the chromosome ordering was changed to affect the length of particular building blocks. He tested the hypothesis that an inversion (string permutation) operator might improve the efficiency of a genetic algorithm for such functions. Because the standard genetic algorithm found near-optimal results quickly in all cases, the inversion operator had little effect. However, for substantially more difficult problems, such as the travelling salesman problem, job shop scheduling and bin packing, Frantz's hypothesis remains a fruitful avenue of research (Davis, 1985; Grefenstette, 1985a).

Bethke (1981) added rigor to the study of functions that are hard for genetic algorithms. Using walsh functions and a clever transformation of schemata, he has devised an efficient, analytical method for determining schema average fitness values using walsh coefficients. This in turn might permit us to identify whether, given a particular function and coding, building blocks combine to form optima or near optima.

DeJong's (1975) dissertation was particularly important to the subsequent applications of genetic algorithms. He recognized the importance of carefully controlled experimentation. Varying population size, mutation and cross-over probabilities, and other operator parameters, he examined genetic algorithm performance in a problem domain consisting of five test functions ranging from a smooth, unimodal function of two variables to functions characterized by high dimensionality (30 variables), great multimodality, discontinuity and noise. To quantify genetic algorithm performance he defined online and offline performance measures, emphasizing interim performance and convergence, respectively. He also defined a measure of robustness of performance over a range of environments and demonstrated by experiment the robustness of genetic algorithms over the test set.

Having been established as a valid approach to problems requiring efficient and effective search, genetic algorithms are now finding more wide spread applications in business, scientific and engineering circles (Grefenstette, 1985b; Holland, 1987; Schaffer, 1989; Belew & Booker 1991).

For example Goldberg (Goldberg, 1983; Goldberg, 1986), has used genetic algorithms successfully in the optimization of pipeline systems, rule-learning in dynamic control applications and structural design.

The reasons behind the growing number of applications of genetic algorithms as compared to conventional search methods are:

- Genetic algorithm techniques use only pay-off information to direct the search, making them independent of a particular application domain. Conventional search techniques use vastly different forms of auxiliary information and application dependent metrics.

- Genetic algorithms use randomized operators such as reproduction, cross-over and mutation as a tool to guide a search towards regions of the search space with likely improvements. As Holland (1986) suggests, most often intuitive ideas are a juxtaposition of things that have worked well in the past. In much the same way, reproduction, cross-over and mutation help to arrive at potentially innovative new ideas.

Recently Goldberg (1992) has drawn a connection between the discriminative and recombinative processes of conceptual design and genetic algorithms. Goldberg states that inductive designers, during conceptual design, most often recombine bits and pieces of previous designs to form new, possibly better proposals. Similarly, Genetic algorithms recombine bits and pieces of artificial chromosomes to search for globally optimal solutions. He further asserts that, these similarities make genetic algorithms a powerful tool for the analysis of innovation in conceptual design. This observation is supported by recalling the inductive nature of genetic algorithms. In a sense, genetic algorithms support a more human-like search during which they are able to generalize from specific instances. As a result, genetic algorithms represent a vital component in the process of conceptual design. This viewpoint is further supported by qualitative arguments viewing design as evolution (Thompson, 1961; French, 1988).

Apart from above theoretical utility, intelligent systems based on genetic algorithms benefit from the following advantages :

1- Genetic algorithms are highly adaptive: search spaces with discontinuities, vastly multimodal and noisy can be searched effectively without auxiliary information requirement. Experience accumulated in a particular search space can be used to encounter new and novel search spaces.

2- No predefined sets of heuristics are given. Intelligent systems based on genetic algorithms start from a randomly generated population of possible heuristic structures. Genetic algorithm creates new and better heuristics using its innovative search mechanism.

3- Genetic algorithms support parallel competition and collaboration among heuristic structures.

4- Genetic algorithms have learning capability : heuristics are created and improved without user intervention.

5.4 Research issues

Genetic algorithms are stochastic processes which sometimes exhibit "premature convergence". The so called "premature convergence" is characterized by convergent behaviour without guarantee of perfect optimality. The major causes of this occasional behaviour are discussed in the following sections :

5.4.1 Genetic drift

Premature convergence to suboptimal results has been observed in empirical studies by both Cavicchio (1970) and DeJong (1975). DeJong linked the primary cause of this behaviour with a natural genetic phenomenon called "genetic drift". In small populations, the difference between the expected number of offspring and the actual realization can cause the population to drift away from the desired path. From section 5.2 we recall that during reproduction the number of copies allocated to a candidate chromosome is proportional to its fitness ratio $F/\overline{F}$. This rate of sampling has been identified as a near-optimal, realizable strategy. Unfortunately, in practice we must deal with the fractional nature of the quotient, $F/\overline{F}$, and allocate an integer number of offspring.

In genetic algorithm research a number of selection strategies has been proposed to reduce the difference between the theoretical near optimal sampling rate and that of the actual realization.

A large class of proposed selection strategies are classified under proportionate reproduction strategies. In proportionate reproduction strategies, individual chromosomes are chosen according to their objective function values f. In these schemes, the probability of selection p of an individual from the ith class of identical chromosomes, in the tth generation is calculated as :

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^{k} m_{j,t} f_j} \qquad (5.4.1.1)$$

Where m is the number of identical individuals in a particular class, k classes exist and the total number of individuals sums to the population size. Various methods have been suggested for sampling this probability distribution, including Monte Carlo or roulette wheel selection (DeJong, 1975), stochastic remainder with replacement and stochastic remainder without replacement (Booker, 1982). There are theoretical differences among stochastic properties of these strategies. Booker's (1982) investigations in a machine learning application has demonstrated the relative superiority of stochastic remainder selection without replacement over DeJong's Monte Carlo selection, However, his approach might still suffer from stochastic errors causing premature convergence. This is primary due to the, relatively, high variance process involved when the fractional parts of fitness ratio $(F/\overline{F})$ are rounded up with a process which is similar to tossing a biased coin, using the fractional parts as the bias.

An alternative selection strategy has been proposed by Baker (1985), called the Ranking Selection. His idea is to sort the population from best to worst and assign the number of copies that each individual should receive according to a non-increasing assignment function. Baker used ranking in an effort to stop premature convergence.

Baker seems largely concerned with slowing down searches that progress too fast because of "super individuals". This is in fact the case with normal selection rule $(p_{i,t}=\frac{f_i}{\sum f})$, the extraordinary individuals would take over a significant proportion of the finite population in a single generation, and this is undesirable. Also, at the later stages of a run, there may still be significant diversity within the population; however, the population average fitness may be close to the population best fitness. If this situation is left alone, average members and best members get nearly the same number of copies in future generations, and the proportionate selection strategy becomes a random walk with no hope of further improvement.

Proponents of proportionate selection have devised a number of fitness scaling procedures to avoid these problems (Goldberg, 1986a; Forrest, 1985; Gillies, 1985). However, these procedures are ad-hoc, problem specific, and complicate the genetic algorithm simulations by adding extra parameters for controlling selective pressure.

Baker's ranking selection, completely solves the scaling problem and provides a consistent means of controlling offspring allocation. In general, ranking methods provide an even, controllable pressure to push for the selection of better individuals.

Whitely (1989) has experimented with ranked-based selection strategies in the context of DeJong's standard test suite and a set of neural net optimization problems. Whitely, based on four standard performance metrics (i.e., De Jong's (1975) on-line, off-line, average and best) , shows the relative superiority of ranked-based reproduction strategies as compared to proportionate reproduction selections.

However, better theories of trial allocation are needed before a final judgment could be made.

5.4.2 Genetic algorithm hard problems

While stochastic errors due to reproduction are a primary ingredient in premature convergence, there is another important reason that might cause premature convergence : The problem might be "genetic algorithm hard". Bethke's (1980) dissertation addressed this issue by applying walsh functions to the study of schema processing in genetic algorithms.

In particular, Bethke developed the walsh-schema transform, in which discrete versions of walsh functions are used to calculate schema average fitnesses efficiently. He then used this transform to characterize functions as easy or hard for the genetic algorithm to optimize. In what follows, we give a brief account of his work.

Walsh functions are a complete orthogonal set of basis functions that induce transforms similar to fourier transforms. However, walsh functions differ from other bases (e.g., trigonometric functions or complex exponentials) in that they have only two values, +1 and -1.

The discrete walsh functions map bit strings x into $\{1,-1\}$. Each walsh function is associated with a particular partitioning of the search space. A partitioning of the search space is defined by a partition number j for those schemata that share the same fixed positions:

$$j(H)=\sum_{i=1}^{l}\alpha(h_i)\cdot 2^{i-1} \qquad (5.4.2.1)$$

Where i is an index over the string positions and the function α assumes a value of 0 when h_i=# and a value of 1 otherwise. In this way the partition number function j assigns a unique number to each of the 2^l partitions of the string space defined by the set of 2^l fixed positions. For example, the schema

### is assigned the partition number *j*(###) = 0. The schemata ##0 and ##1 share the partition number *j* = 1, and the schema 0#1 is assigned a partition number *j*(0#1) = 5. From section 5.2, we recall that schemata with similar partition numbers will have the same set of defining positions, and they become competing schemata.

The walsh function corresponding to the *j*th partiotion is defined as follows (Bethke, 1980):

$$\psi_j(x)=\begin{cases}1 & \textit{if } x \wedge j \textit{ has even parity}\\ -1 & \textit{otherwise}\end{cases}$$

Here, $\wedge$ stands for bitwise AND. It is important to note that $\psi_j(x)$ has the property that the only bits in x that contribute to its value are those that correspond to 1's in $j(H)$ (i.e., $j(H)$ represented by its equivalent binary representation). Since the walsh functions form a basis set, any function $f(x)$defined on $\{0,1\}^l$ can be written as a linear combination of walsh functions:

$$f(x)=\sum_{j=0}^{2^l-1}\omega_j\ \psi_j(x) \qquad (5.4.2.2)$$

The above expression is called the the walsh polynomial representing $f(x)$; Where x is a bit string, l is its length, and each ω_j is a real-valued coefficient called a walsh coefficient. Knowing that ψ_j basis is orthogonal, in general we may calculate the walsh coefficients as follows:

$$\omega_j=\frac{1}{2^l}\sum_{x=0}^{2^l-1}f(x)\psi_j(x) \qquad (5.4.2.3)$$

The above expression is called the walsh transform (Goldberg, 1989b). Once the ω_j's have been determined by equation (5.4.2.3), $f(x)$can be calculated by equation (5.4.2.2).

There is a close connection between the walsh transform and schemas. The walsh-schema transform formalizes this connection. Formal derivations of the walsh-schema transform are given by Bethke (1980) and Goldberg (1989b). The walsh schema transform provides the relationship between a schema's average fitness, $f(H)$, and the function's walsh coefficients, ω_j . It is given by:

$$f(H)=\sum_{j:j\ subsumes\ H} \omega_j\, \Psi_j(H)$$

Where a partition j is said to subsume a schema H if it contains some schema H' such that $H' \supseteq H$. For example, the three-bit schema 10# is subsumed by four partitions : dd#, d##, #d#, and ###, which correspond to the $j(H)$ values 110, 100, 010, and 000 respectively (please see equation 5.4.2.1). In other words, j subsumes H if and only if each defined bit in j (i.e., each 1) corresponds to a defined bit in H (i.e., a 0 or a 1, not a #).

More simply a schema's average fitness may be calculated as a partial, signed sum of walsh coefficients, where the only coefficients included in the sum are those associated with partitions that contain the schema. The sign of a particular coefficient is positive or negative if $\Psi_j(H)$ is positive or negative respectively. The sign of $\Psi_j(H)$ is given by (Here, dont care tokens (#) are considered as 0):

$$\Psi_j(H)=\begin{cases} 1 & if\ H \wedge j\ has\ even\ parity \\ -1 & if\ H \wedge j\ has\ odd\ parity \end{cases}$$

For example, in a single variable optimization problem, with the independent variable represented by a 3 bit string, the average fitness of the schema 101 is given by :

$$f(101)=\omega_{000}-\omega_{001}+\omega_{010}-\omega_{011}-\omega_{100}+\omega_{101}-\omega_{110}+\omega_{111}$$

Therefore, by using walsh-schema transform, the average fitness of a particular schema can be calculated from the average fitness of its underlying constituents. i.e., in above case, the average fitness of the schema 101, is calculated by using the average fitness of the schemata : ###, ##1, #0#, #01, 1##, 1#1, 10#, and 101.

It is instructive to note that as a schema becomes more specific, new walsh coefficients are added to its walsh transform representation. For example an order-1 approximation to the average fitness of the schema 101 would be :

$$\hat{f}^{(1)}(101) = \omega_{000} - \omega_{001} + \omega_{010} - \omega_{100}$$

But an order-2 approximation to its average fitness is given by :

$$\hat{f}^{(2)}(101) = \omega_{000} - \omega_{001} + \omega_{010} - \omega_{011} - \omega_{100} + \omega_{101} - \omega_{110}$$

Therefore, we may view the fitness of a higher order schema as estimated by the summation of its lower order constituent schemata. More rigorously, $\hat{f}^{(o)}(H)$ may be defined as the oth-order approximation to the fitness of the schema H; the magnitude of the difference between a schema average and its next lowest order approximation is simply the highest order walsh coefficient in the sum. In the above example this difference is given by ω_{111}.

This view of assembling functions from lower order schemata shows nicely the way genetic algorithms actually work : A population of strings in a genetic algorithm can be thought of as a number of samples of various schemas, and the genetic algorithm works by using the fitness of the strings in the population to estimate the fitness of schemas. It exploits fit schemas via reproduction by allocating more samples to them, and it explores new schemas via crossover by combining fit low-order schemas to sample higher-order schemas that will hopefully also be fit. In general there are many more instances of low-order schemas in a given population than high-order schemas (i.e., in a randomly generated population, about half the strings will be instances of 1### · · · #, but very few, if any will be instances of 1111· · · 1. Therefore, accurate fitness estimates will be obtained much earlier for low-order schemas than for high-order schemas. The genetic algorithm, initially bases its estimate on information about low-order schemas containing a given H, and gradually refines this estimate from information about higher and higher-order schemata containing H. In the same way, the terms in the sum of the above example represent increasing refinements to the estimate of how good the schema 101 is. The term ω_{000} gives the population average (corresponding to the average fitness of the schema ###) and the increasing higher-order ω_j's in the sum represent higher-order refinements of the estimate of 101's fitness, where the refinements are obtained by summing ω_j's corresponding to higher and higher-order partitions j containing 101.

To summarize, one way of describing the genetic algorithm's operation on a fitness function f is that it makes progressively more accurate estimates of the f's walsh coefficients, and biases the search towards partitions j with high-magnitude ω_j's, and to the partition elements (schemas) for which these correction terms are positive.

Bethke (1980) used walsh analysis to partially characterize functions that will be easy for the genetic algorithm to optimize. Bethke suggested that if the walsh coefficients of a function decrease rapidly with increasing order and defining length of the j's (i.e., the most important coefficients are associated with short, low-order partitions) then the function will be easy for the genetic algorithm to optimize. In such cases, the location of the global optimum can be determined from the estimated average fitness of low-order, low-defining-length schemata. Therefore, a function whose walsh decomposition involves high-order j's with significant coefficients should be harder for the genetic algorithm to optimize than a function with only lower order j's, since it will be harder for the genetic algorithm to construct good estimates of the higher-order schemata belonging to the higher-order partitions j.

Bethke's analysis was not intended as a practical tool for use in deciding whether a given problem will be hard or easy for the genetic algorithm. Unfortunately, the fitness functions used in many genetic algorithm applications are not of a form that can be easily expressed as a walsh polynomial. Moreover, even if a function f can be so expressed, a walsh transform of f requires evaluating f at every point in its argument space, and is therefore an infeasible operation for most applications of interest (this is also true for the "Fast Walsh Transform", Goldberg, (1989b)). However, walsh analysis can be used as a theoretical tool for understanding the types of properties that can make a problem hard for the genetic algorithm. For example, Bethke used the walsh-schema transform to construct functions that mislead the genetic algorithm, by directly assigning the values of walsh coefficients in such a way that the average values of low-order schemata give misleading information about the average values of their higher-order schemata sub-sets. Such functions were later called "deceptive" by Goldberg (1987a), who carried out a number of theoretical studies of such functions. Goldberg used a dynamic approach which in addition to the deceptive properties of a particular fitness function takes into account the combined consideration of genetic algorithm's operators and coding. This approach can only be used in small problems and was used for the dynamic analysis of the minimal, deceptive problem (MDP). He concluded that although his fitness function was deceptive, it was not genetic algorithm hard and the genetic algorithm consistently found the global optimum in his experiments.

Deception has since been a central focus of theoretical work on genetic algorithms (Tanese, 1989). Walsh analysis can be used to construct problems with different degrees and types of deception, and the genetic algorithm's performance on these problems can be studied empirically. The goal of such research is to to learn how deception affects genetic algorithm performance and why the genetic algorithm might fail in certain cases and to learn how to improve the genetic algorithm or the problem's representation in order to improve performance.

Having gone through the above research considerations, it is important to emphasize that in practice it is much more efficient to run the genetic algorithm on a given function and measure its performance directly than to decompose the function into walsh coefficients and then determine from these coefficients the chance of success. Practically speaking, deceptive functions are mostly characterized by having the best optimal points being surrounded by the worst which might happen only in exceptional real world problems. Moreover, finding a needle in a haystack is going to be difficult regardless of the search technique used.

5.5 Genetic algorithms for the design of corrugated - diaphragms

In this section we apply the three operator genetic algorithm to the problem of design and optimization of corrugated diaphragms using analytical models.

A diaphragm is characterized at the power-flow level of abstraction as an instrument which transforms input pressure or force into its output pressure, displacement or force.

A particular diaphragm design achieves any of the above functionalities by obeying specific geometric constraints.

Diaphragms can have linear or non-linear pressure or force characteristics depending on their geometry. Diaphragms of the simplest shape are the flat diaphragms whose characteristics are strongly progressive for large deflections. Corrugation of the diaphragms increases their deflections. The diaphragm characteristics can thus be varied by changing the shape and dimentions of the corrugations. Corrugated diaphragms are accordingly used more than flat diaphragms. The geometrical shape of the corrugated diaphragms is such that its rigidities in radial and peripheral directions are different : The resistance of an elemental strip cut from the corrugated diaphragm to bending and stretching will be much smaller in the radial direction than in the peripheral direction. Thus, the corrugated diaphragm is anisotropic, due to its particular geometry. Consequently, the idea of flat anisotropic diaphragms as being equivalent to corrugated diaphragms, has been used to simplify the derivation of mathematical models for the corrugated diaphragms (Andreeva, 1966). These mathematical models are described in the next section.

5.5.1 Analytical models

Two models for corrugated diaphragms are derived (Andreeva, 1966) :

1- Pressure-loading case
2-Force-loading case

The characteristic equation for the pressure-loading case is :

$$\frac{PR^4}{Eh^4} = a_p \cdot \frac{\omega_0}{h} + b_p \cdot \frac{\omega_0^3}{h^3} \qquad (5.5.1.1)$$

The coefficients a_p and b_p in this formula are :

$$a_p = \frac{2(3+\alpha)(1+\alpha)}{3 \cdot k_1\left(1-\frac{\mu^2}{\alpha^2}\right)} \qquad (5.5.1.2)$$

$$b_p = \frac{32 \cdot k_1}{\alpha^2 - 9} \cdot \left[\frac{1}{6} - \frac{3-\mu}{(\alpha-\mu)(\alpha-3)}\right] \qquad (5.5.1.3)$$

Where:

ω_o	:	The deflection of the diaphragm center (mm)
p	:	The pressure being sensed (kg/m2)
E	:	The modulus of elasticity (kg/m2)
μ	:	Poisson's ratio
h	:	The diaphragm thickness (mm)
R	:	The active radious (mm)

For the force-loading case the characteristic equation is :

$$\frac{Q \cdot R^2}{\pi \cdot E \cdot h^2} = a_Q \frac{\omega_0}{h} + b_Q \frac{\omega_0^3}{h^3} \qquad (5.5.1.4)$$

Where Q is the pointed force (kg). The coefficients a_Q and b_Q are :

$$a_Q = \frac{(1+\alpha)^2}{3 \cdot k_1\left(1-\left(\frac{\mu^2}{\alpha^2}\right)\right)} \quad , \quad b_Q = \frac{k_1}{(\alpha^2-1)}\left[\frac{1}{2} - \frac{1-\mu}{(\alpha-\mu)(\alpha+1)}\right]$$

According to Andreeva (1966), the parameter α in these equations is given by:

$$\alpha = \sqrt{k_1 \cdot k_2} \tag{5.5.1.5}$$

The coefficients k_1 and k_2 depend on the shape of the diaphragm's profile and in the general case of periodical corrugations these coefficients for saw-tooth type of profiles are:

$$k_1 = \frac{1}{\cos\theta_0} \quad , \quad k_2 = \left(\frac{H}{h}\right)^2 \times \frac{1}{\cos\theta_0} + \cos\theta_0$$

Where:

θ_0 : Profile-angle (Radians)

H/h : Relative corrugation depth

The diagram of a prototypical saw tooth corrugated diaphragm is shown below:

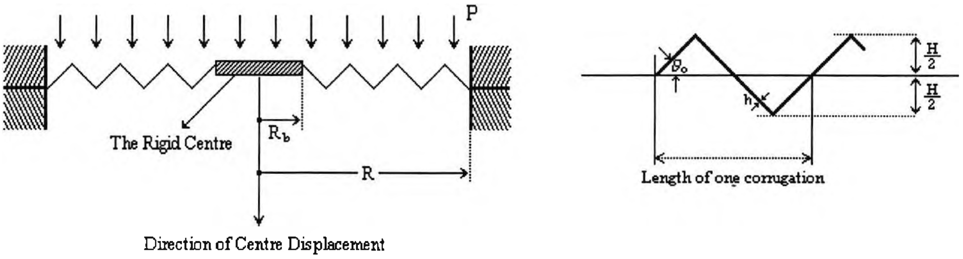


Figure 5.7a Schematic of a saw tooth corrugated diaphragm

5.5.2 An initial study using diaphragms with known characteristics

In this section, we formulate the diaphragm design problem in a simple single parameter optimization problem to investigate the functioning of the three operator genetic algorithm. The problem is formulated as follows :

Given a user specified pressure-displacement or force-displacement characteristics, the three operator genetic algorithm is required to find the appropriate diaphragm profile parameters (i.e., the relative corrugation depth of diaphragm, its thickness and height) such that the required characteristic is satisfied with minimum error.

From equation (5.5.1.1), if the general pressure-loading characteristics is shown as:

$$p = A \cdot \omega_0 + B \cdot \omega_0^3 \qquad (5.5.2.1)$$

Where:

$$A = \frac{E}{R^4} \cdot h^3 \cdot a_p \qquad , \qquad B = \frac{E}{R^4} \cdot h \cdot b_p$$

Elimination of thickness h from above two equations gives :

$$\frac{a_p}{b_p^3} = \frac{A}{B^3}\left(\frac{E}{R^4}\right)^2 \qquad (5.5.2.2)$$

Similarly using equation (5.5.1.4), if the general force loading characteristics is shown as :

$$Q = A' \cdot \omega_0 + B' \cdot \omega_0^3 \qquad (5.5.2.3)$$

Where:

$$A' = \frac{\pi \cdot E \cdot h^3}{R^4} \cdot a_Q \qquad , \qquad B' = \frac{\pi \cdot E \cdot h}{R^2} \cdot b_Q$$

Elimination of the thickness h gives :

$$\frac{a_Q}{b_Q^3} = \frac{A}{B^3}\left(\frac{\pi \cdot E}{R^2}\right)^2 \qquad (5.5.2.4)$$

For a particular *H/h*, the cost (i.e., error which defines a diversion measure from the user specified characteristic) is defined as:

$$COST_p = \left| \frac{a_p}{b_p^3} - \frac{A}{B^3}\left(\frac{E}{R^4}\right)^2 \right| \qquad (5.5.2.5)$$

$$COST_Q = \left| \frac{a_Q}{b_Q^3} - \frac{A}{B^3}\left(\frac{\pi \cdot E}{R^2}\right)^2 \right| \qquad (5.5.2.6)$$

Equation 5.5.2.5 defines cost for a pressure loading case. Equation (5.5.2.6) defines cost for a force-loading case.

5.5.3 Some initial simulations

As discussed in section 5.2 genetic algorithms have a number of parameters which must be selected : population size N, cross-over probability p_c and mutation probability p_m.

The effect of these parameters upon genetic algorithm performance has been investigated by DeJong (1975). He has performed parameter studies of the three operator genetic algorithm over a set of five problems in function optimization. He included functions with the following characteristics:

Continous / Discontinous
Convex / Nonconvex
Unimadal / Multimodal
Quadratic / Nonquadratic
Low-dimensional / High-dimensional
Determnistic / Stochastic

To quantify the effectiveness of different genetic algorithmic parameters, De Jong devised two measures, one to gauge performance and the other to gauge the on-going performance. He called these measures off-line (convergence) and on-line (on-going) performance respectively.

In his study, DeJong defined the on-line performance $X_e(s)$ of strategy s on environment e as follows :

$$X_e(s) = \frac{1}{T} \sum_{1}^{T} f_e(t) \qquad (5.5.3.1)$$

Where $f_e(t)$ is the objective function value for environment e on trial t. In words the on-line performance is an average of all function evaluations up to and including the current trial.

He also defined the performance measure $X_e^*(s)$, the off-line performance of strategy s on environment e as follows:

$$X_e^*(s) = \frac{1}{T}\sum_{1}^{T} f_e^*(t) \qquad (5.5.3.2)$$

Where $f_e^*(t)$ = best $\{f_e(1), f_e(2), \ldots\ldots, f_e(T)\}$. In words, the off-line performance is a running average of the best performance values to a particular time. With a test-bed of five trial functions, and two criteria of goodness, DeJong was able to offer the following conclusions in regard to the selection of appropriate genetic algorithm parameters:

" Increasing the population size was shown to reduce the stochastic effects of random sampling on a finite population and improve long-term performance at the expense of slower initial response. Increasing mutation rate was seen to improve off-line performance at the expense of on-line performance. Reducing the cross-over rate resulted in an overall improvement in performance, suggesting that producing a generation of completely new individuals was too high a sampling rate (DeJong, 1975). From these experiments there emerged a set of values for these parameters that was found to yield generally good behaviour for this suite of problems for both on-line and off-line performance. They are :

Population size = 50 - 100
Cross-over rate = 0.65
Mutation-rate = 0.001

DeJong also investigated the use of generalized (i.e., multi-point) crossover. After showing that theoretically these operators exhibit very different schema disruption behaviour, he concluded that no significant empirical difference could be seen on the suite of test-bed problems.

Following these considerations, at this stage we choose the same values for these parameters. But first, we must consider an appropriate cost-to-fitness transformation for the effective performance of the genetic algorithm. We recall that our objective is to find the optimal diaphragm profile parameters such that the required characteristics is satisfied with minimum error. Therefore, our objective function is more naturally stated as the minimization of the cost (i.e., error) function. But genetic algorithms function by processing non-negative fitness values. In general, the fitness function consists of the composition of two functions :

$$u(x) = g(f(x)) \qquad (5.5.3.3)$$

Where f is the objective function and g transforms the value of the objective function to a non-negative fitness value. The mapping performed by g is always necessary when the objective function is to be minimized (since lower objective function values must map to higher fitness values) or when the objective function can take on negative values (since genetic algorithms can only process positive fitness values).

There are a number of alternatives for this mapping. One way to do this is with the following simple mapping relationship :

$$u(x) = \begin{cases} C_{\max} - f(x) & f(x) < C_{\max} \\ 0 & f(x) < C_{\max} \end{cases}$$

Where:

$C_{\max}$: Nominal maximum cost

$u(x)$: fitness function

$f(x)$: cost function

There are a variety of ways to choose the coefficient $C_{\max}$. $C_{\max}$ may be taken as an input coefficient, as the largest $f(x)$ (i.e. cost) value observed so far in the current population or the largest of the last k generations.

Another alternative for this mapping function is the following rational function:

$$u(x) = \frac{c_1}{c_2 + f(x)} \qquad (5.5.3.4)$$

Clearly as cost goes to infinity, fitness goes to zero. c_1 and c_2 may be selected to scale $u(x)$ appropriately. This fitness mapping is particularly useful in our application because our objective function takes a wide range of values.

As discussed in section (5.3.1), during the selection phase, by using a proportionate selection strategy, the genetic algorithm determines an individual's expected number of offspring by saving its associated fitness value as compared to the average fitness of the population. This process defines a frequency distribution over the current population.

The next generation is produced by taking N samples from the current population using this frequency distribution. In the following simulations, we will use the stochastic remainder without replacement selection. In this selection algorithm samples are awarded deterministically based on the integer portions of the expected values. Next the fractional parts of the expected number values are treated as weighted success probabilities. For example, a string with an expected number of copies equal to 1·5 would receive a single copy deterministically and another with probability 0·5. This process continues until the next generation's population is full. This sampling algorithm provides minimum bias, and the greatest lower bound on the standard related to the number of offspring for a particular search point. The standard-deviation characterizes the possible spread of actual number of offspring an individual receives in a given generation.

For the simulations to be considered in this section, the three operator genetic algorithm was implemented such that each population member represents a particular relative corrugation depth *(H/h)*, according to the constraint :

$$1mm < \frac{H}{h} < 17mm \qquad (5.5.3.5)$$

30 bit per population member was chosen to give a resolution per bit increment of :

$$\pi = \frac{17-1}{2^{30}-1} \qquad (5.5.3.6)$$

In these studies unsigned integer fixed-point coding have been used. This means that a binary number with 30 bits is first translated into its equivalent decimal representation and next mapped into a particular *(H/h)* value according to:

$$\frac{H}{h} = \left(\pi \times \left(decoded\ string\ in\ unsigned\ \mathrm{int}\,eger\right)\right) + 1 \qquad (5.5.3.7)$$

Therefore, the binary parameter represents $2^{30} \cong 1{\cdot}07\times10^{9}$ number of alternative strings and the genetic algorithm is processing $3^{30} \cong 2{\cdot}06\times10^{14}$ number of different schemata per generation.

As we have formulated the problem as a minimization, we transform the objective function to a fitness function with two alternative transformations using equation (5.5.3.4) :

$$u_1 = \frac{1}{f(x)} \tag{5.5.3.8}$$

$$u_2(x) = \frac{1}{1+f(x)} \tag{5.5.3.9}$$

In equation (5.5.3.8), and recalling equation (5.5.3.4), we have c_1=1, c_2= 0 and, in equation (5.5.3.9), we have c_1 =1 and c_2 =1. It must be mentioned that fitness mapping as represented by equation (5.5.3.9) is superior as compared to equation (5.5.3.8). This is due to the fact that the fitness transformation, as represented by equation (5.5.3.8), is unbounded and for a zero cost value, we will obtain a fitness of $+\infty$.

The fitness transformation represented by equation (5.5.3.9) which is a normalized fitness function will always lie between 0·0 and 1·0 in our case, because the cost value is bounded by $[0,+\infty]$. This normalized fitness mapping is also advantagous for optimization using penalty methods. In these problems, we need to estimate the nominal cost of the objective function.

However, at this stage, we use both transformations to investigate the genetic algorithm robustness properties under different fitness mappings.

We are now in a position to run some simulations and investigate the performance of the genetic algorithm. Two design examples from Andreeva (1966) are used. These are:

1- Given the following user specified pressure-displacement characteristics (i.e., pressure- loading case):

$$p = 0\cdot 106\cdot \omega_0 + 0\cdot 1055\cdot \omega_0^3 \qquad (5.5.3.10)$$

p : pressure (kg/cm^2)
ω_0 : displacement (mm)

and the following diaphragm parameters :

E (young's modulus) = 1·35×10^6 (kg/cm^2)
R (effective radious) = 24 mm

The genetic algorithm is required to find :

1- The optimal profile's relative corrugation depth
2- The diaphragm thickness
3- The diaphragm height

These profile parameters must satisfy the user required characteristics with minimal error.

2- Given the following user specified force-displacement characteristics (i.e., force-loading case) :

$$Q = 1\cdot 012\cdot \omega_0 + 0\cdot 437\cdot \omega_0^3 \qquad (5.5.3.11)$$

Q - pointed force (kg)

and the following diaphragm parameters :

E (young's modulus) = 13500 (kg/cm^2)
R (active radious) = 24.75 mm

We are to find the optimal similar profile parameters satisfying the user required force-displacement characteristics. The mathematical model, objective functions and genetic algorithm parameters have been programmed as described. Different independent trials, using different seeds for the pseudo-random number generator, were run. For each trial, the genetic algorithm is run to generation 100.

Figure 5.2 represents a typical run for our first example design problem. Figure 5.3 represents a typical run for our second example design problem. In both of these simulations, equation (5.5.3.8) has been used for fitness mapping. Each figure shows the fitness of the best string of each generation and the average fitness of the population as the solution proceeds.

In both simulations, near optimal results are obtained by generation 40 (i.e. 40 ×100 = 4000 function evaluations). This may seem like a large number of function evaluations. But if we consider the size of the space being searched, we recall that the binary strings are of the length l = 30. This represents a total of $2^{30} \cong 1{\cdot}07\times10^{9}$ possible different alternatives in the search space. Therefore 4000 function evaluations is a very small fraction (0.00000037%) of the possible unique alternatives. The near optimal solutions are given below :

Design Problem-1 (Pressure-loading case)	**Design Problem-2** (Force-loading case)
H/h = 4·098511	H/h = 2·964136
Thickness = 0·100934 mm	Thickness = 0·135124 mm
Corrugation depth = 0·4136 mm	Corrugation depth = 0·4005mm

Table 5.1 Optimal designs for design problems 1 & 2

These results were checked against Andreeva's (1966) results. They were found to be accurate to 4 decimal places. Using the alternative fitness mapping, given by equation (5.5.3.9), similar results were obtained demonstrating the robustness of genetic algorithm and its relative insensitivity to the particular fitness transformation used. These results are more than acceptable for practical purposes.

Considering the population average performance, as shown in figure 5.2 and figure 5.3, at first, most of the population have very low fitness values. During the early and middle stages of the simulation, the creative schema exchange brings fast rapid improvement. After a while, the population enters a stagnation period. An examination of the individual strings at this point shows substantial convergence at most positions.

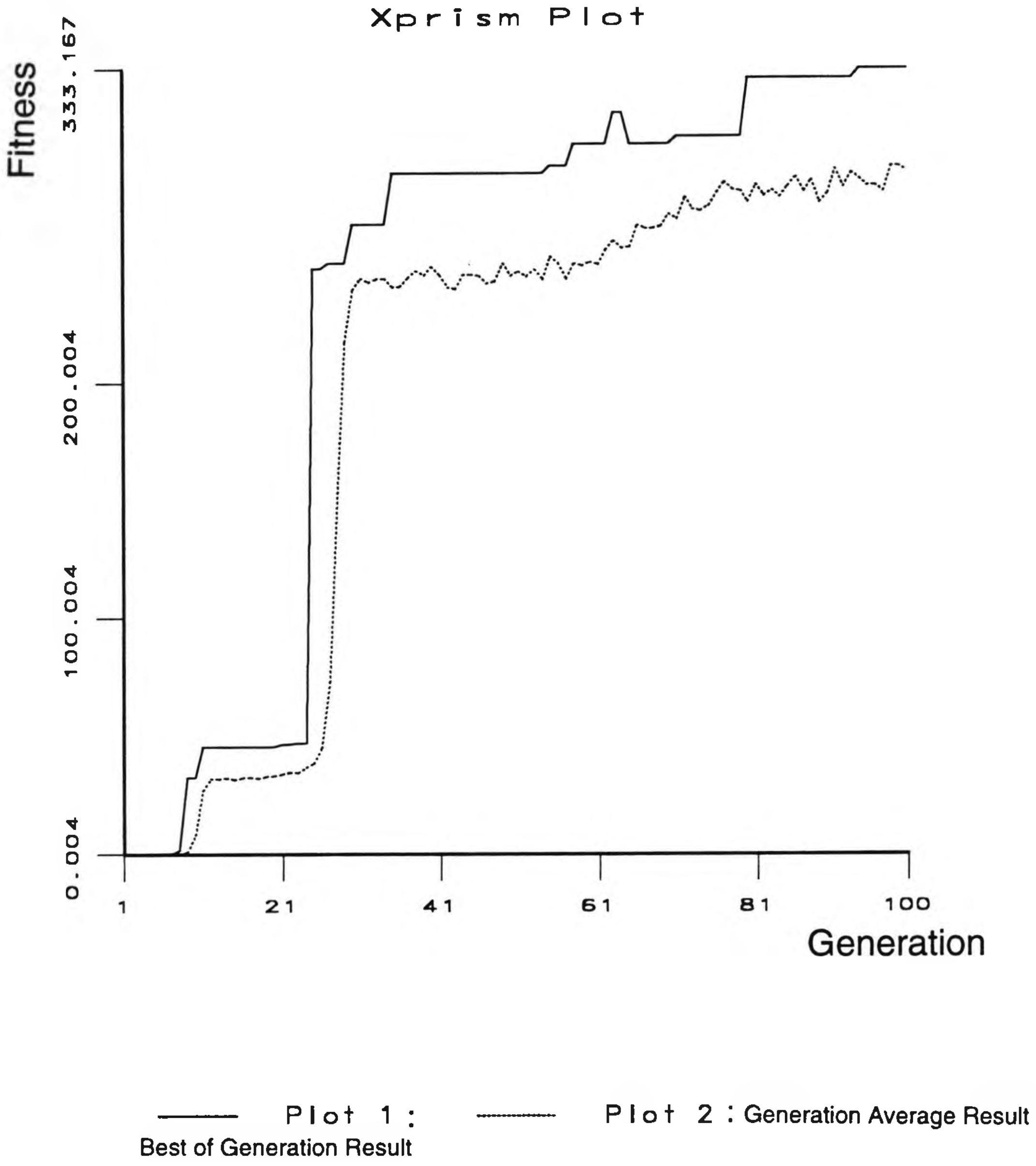


Figure 5.2 Pressure Loading Design Problem

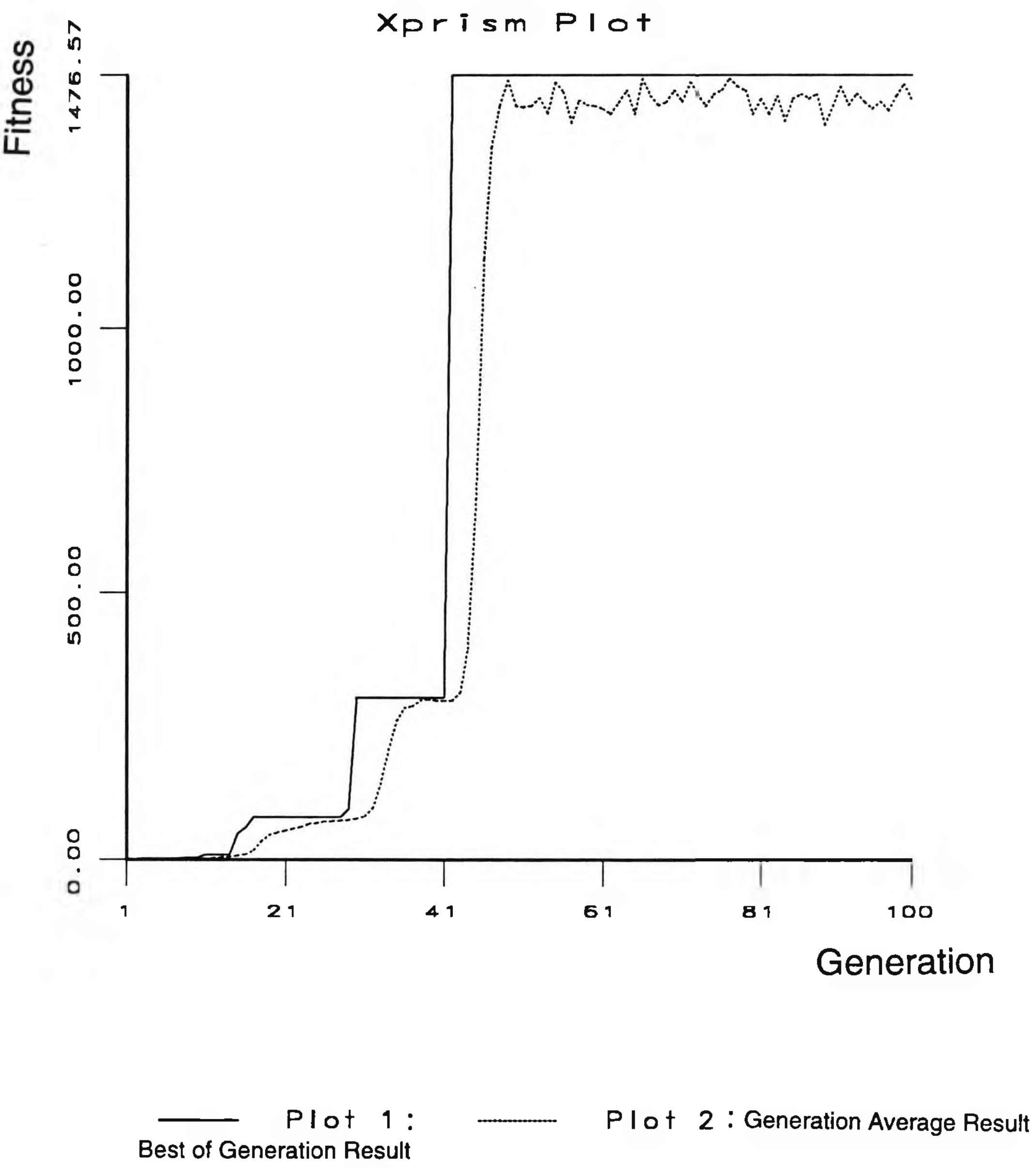


Figure 5.3 Force Loading Design Problem

This is represented, in these figures, by the closeness of the population average and maximum value during later generations. Recalling our fitness mapping (i.e., equation (5.5.3.8)) used in these simulations, there is still immense scope in increasing the fitness (i.e. in case of equation (5.5.3.8), the ideal zero error corresponds to $+\infty$ fitness). Obviously, we expect that with higher iterations the accuracy of the results must converege to the precision provided per bit increment of each population string member (as specified by equation (5.5.3.6)). The major reason for this behaviour is genetic drift (elaborated in section 5.3.1). Therefore, we must consider possible improvement to the reproduction operator.

However, a more robust approach for determination of optimum parameter settings has been suggested by Grefenstette (1986). He used a meta-GA to locate parameter sets which themselves were used for genetic algorithm searches on the DeJong's set of trial functions. This approach is advantagous as genetic algorithm's powerful implicit parallelism is used to explore the space of parameter combinations. Unfortunately, this method suggests good parameter sets without providing much insight as to how sensitive performance is to variations. Using this method, Grefenstette was able to locate an interesting parameter set that provided significantly better on-line performance than De Jong's settings, but was unable to improve upon the conventional DeJong's off-line performance settings. His recommended values were :

population size = 30
crossover rate = 0·95
mutation rate = 0·01

Note that, he is recommending a smaller population size and much higher rates of applying the genetic algorithm operators than DeJong does. Using these parameter values, another set of simulations were carried out. In these simulations, the normalized fitness mapping as represented by equation (5.5.3.9) was used for the same example design problems. In figure 5.4, plot-2 and figure 5.5, plot-1, typical best string produced after each generation is represented for the1st. and 2nd example problems.

As expected, the on-line performance has improved significantly but the off-line performance has degraded. Specifically, in figure 5.4, plot-2, the best-of-run result has converged to a highly sub-optimal solution at generation 41.

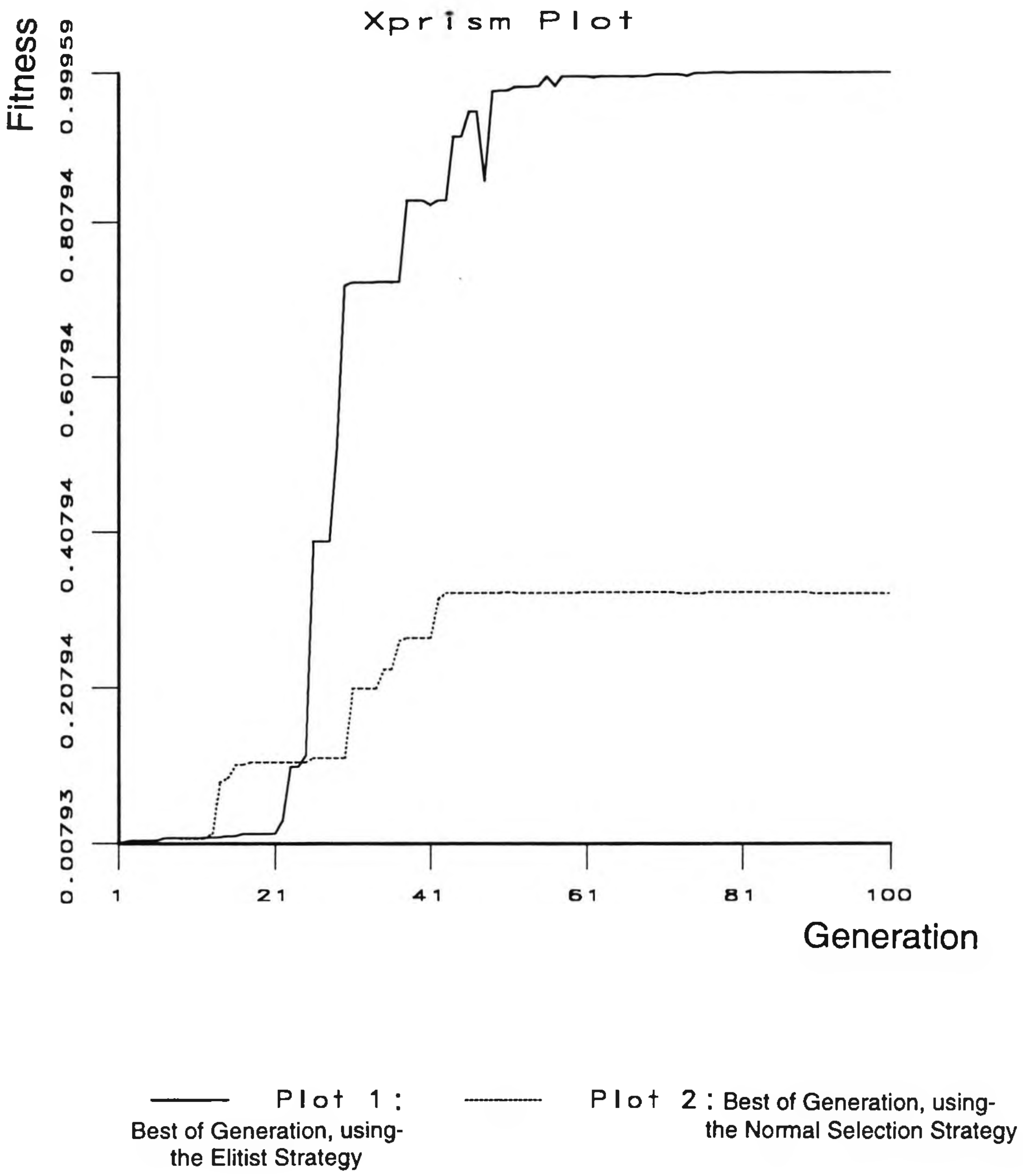


Figure 5.4 Pressure Loading Design Problem (Normalized Formulation)

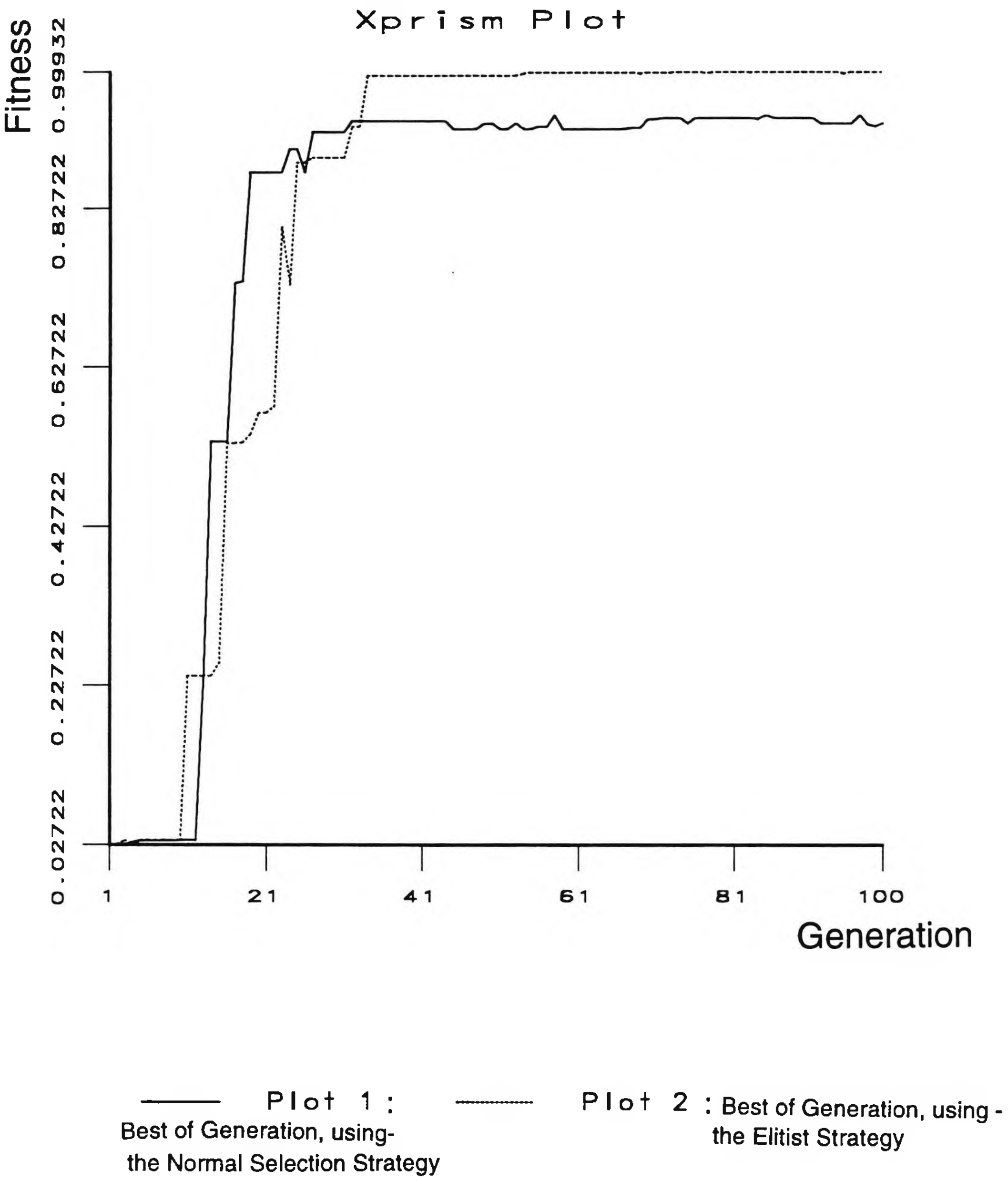


Figure 5.5 Force Loading Design Problem (Normalized Formulation)

Looking at figure 5.6, plot-1, we observe that the average performance is oscillating about the highly sub-optimal solution, with no hope of further improvement.

This degradation of performance is caused by using a very low initial population size, resulting in an increase in stochastic effects due to insufficient processing of too few schemata and fast convergence to a sub-optimal solution. These stochastic errors will result in a significant loss of useful schemata contained within the population resulting in the best members of population to fail to produce offspring according to their actual expected values.

In order to remedy this problem higher population sizes can be used which results in a degradation of on-line performance. An alternative strategy is to fix this potential source of loss by copying the best member of each generation into the succeeding generation. This strategy has been called the Elitist method; although it might increase the speed of domination of a population by a super-individual, but on balance has been shown to improve the performance of genetic algorithms on unimodal functions (DeJong, 1975).

In order to confirm these observations, an improved genetic algorithm was implemented such that the best string is always preserved in subsequent generations. As seen in plot-1 of figure 5.4 and plot-2 of figure 5.5, the genetic algorithm using a significantly lower population size has obtained near optimal solutions with a significant improvement in both on-line and off-line performance. This is also confirmed by looking at figure 5.6, plot-2, in which the average performance has improved dramatically. Looking at figure 5.4 and figure 5.5, the optimal solutions are reached after approximately 40 generations for both design example problems. This means $(40\times30) = 1200$ function evaluations, which is a significant reduction in the number of function evaluations as compared to the previous simulations.

5.5.4 Design of corrugated diaphragms with optimal characteristics

In this section, we will reformulate the diaphragm design problem as a non-linear programming problem (NLP). This reformulation is for the purpose of optimization of the characteristic performance of corrugated diaphragms such

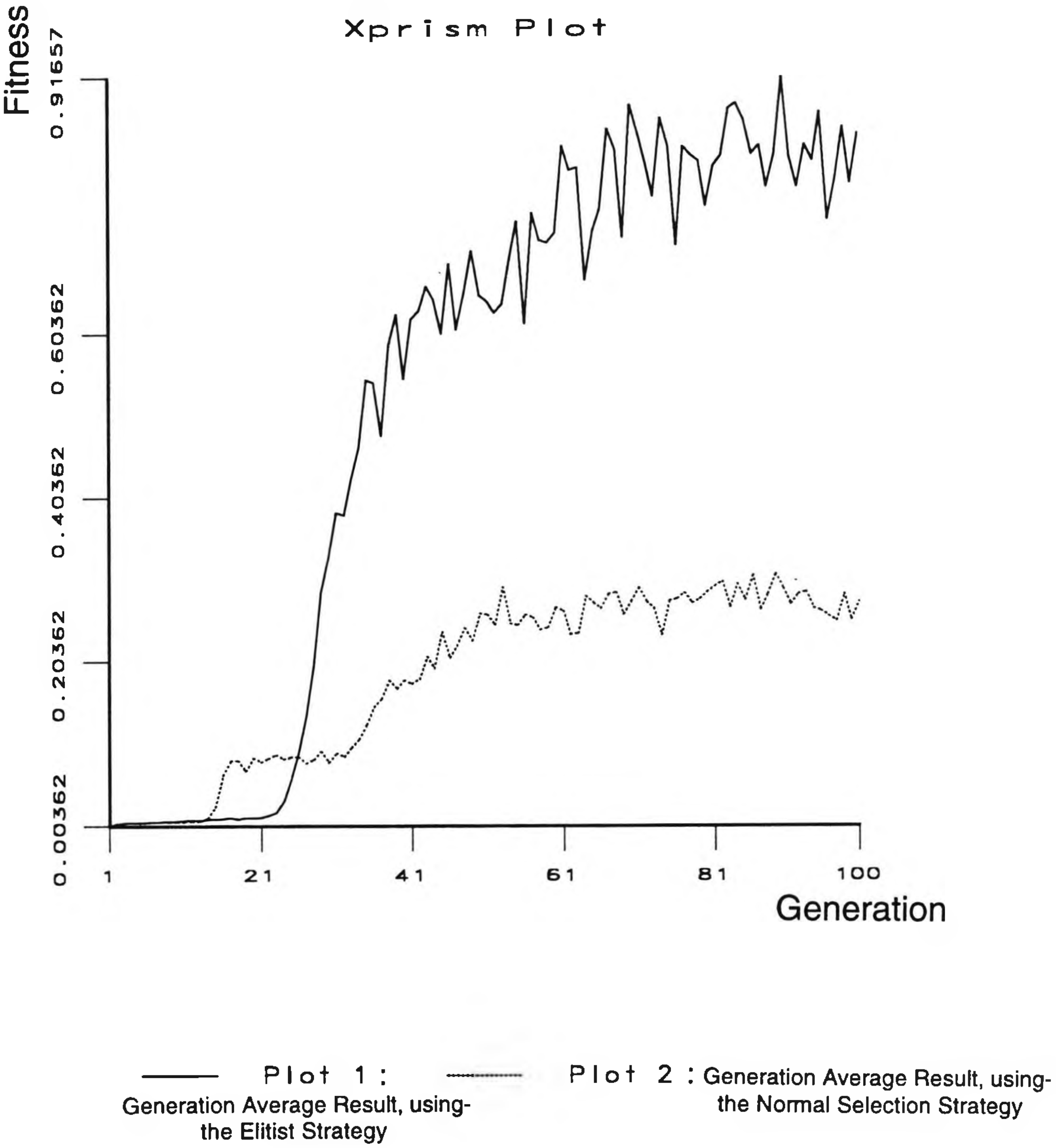


Figure 5.6 Pressure Loading Design Problem (Normalized Formulation)

that a minimum % nonlinearity error is obtained for a user specified maximum allowable pressure and displacement.

From equation (5.5.2.1), section 5.5.2, the general pressure loading characteristic is shown as:

$$p = A\cdot\omega_0 + B\cdot\omega_0^3$$

Where :

$$A = \frac{E}{R^4}\cdot h^3\cdot a_p \qquad , \qquad B = \frac{E}{R^4}\cdot h\cdot b_p$$

The program is required to derive optimal diaphragm designs, according to the following user specified requirement :

1- p_{max} : Maximum applied pressure (kg/cm²)
2-ω_{max} : Maximum displacement (mm)
3- Acceptable non-linearity error

The ideal sensitivity of the required diaphragm is given by p_{max}/ω_{max}. The non-linearity is defined as the ratio of deviation from best fit line or terminal line to the difference of maximum and minimum output pressure in percentage, as shown in figure 5.7 below:

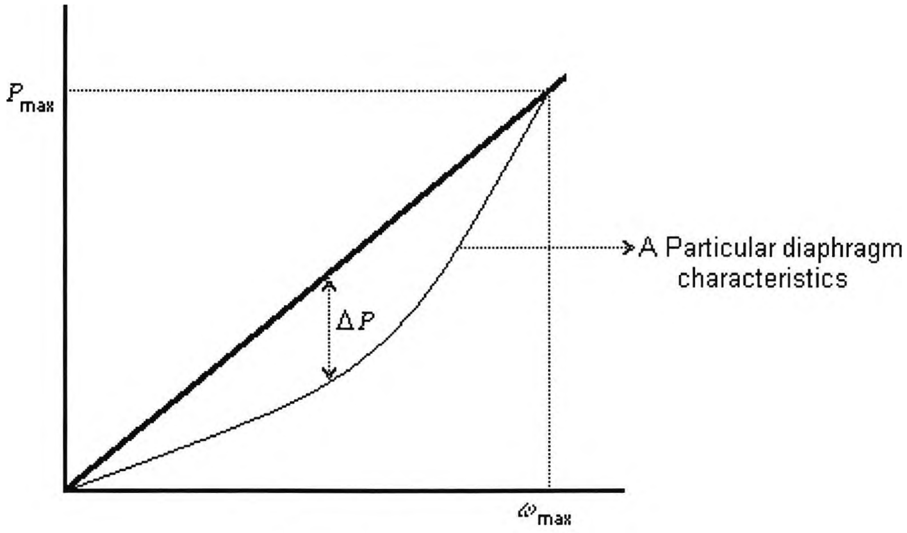


Figure 5.7

$$\%\ \text{nonlinearity} = \frac{\Delta p}{p_{\max}}\times 100 \qquad (5.5.4.1)$$

As indicated by equation (5.5.2.1), the maximum deviation from the best fit line ocuurs at (1/2)$\cdot p_{max.}$ This indicates the first deviation measure from the ideal linear characteristics (We denote this deviation by $ER1$). A given diaphragm design might not satisfy the maximal end pressure for the specified maximal centre displacement. This indicates the second deviation from the ideal linear characteristics (We denote this deviation by $ER2$). The following non-linear programming problem results :

Minimize :

$$F\left(\theta,n,h,{}^{H}\!/\!{}_{h}\right) = ER1\left(\theta,n,h,{}^{H}\!/\!{}_{h}\right) + ER2\left(\theta,n,h,{}^{H}\!/\!{}_{h}\right) \qquad (5.5.4.1)$$

Where :

$$ER1\left(\theta,n,h,{}^{H}\!/\!{}_{h}\right) = \frac{\left|A\left({}^{\omega_{\max}}\!/\!{}_{2}\right) + B\left({}^{\omega_{\max}}\!/\!{}_{2}\right)^{3} - \left({}^{P_{\max}}\!/\!{}_{2}\right)\right|}{P_{\max}} \times 100$$

$$ER2\left(\theta,n,h,{}^{H}\!/\!{}_{h}\right) = \left|A(\omega_{\max}) + B(\omega_{\max})^{3} - P_{\max}\right|$$

Subject to :

$$1 \le n \le 10$$

$$5mm < R < 60mm$$

$$15^{o} < \theta < 30^{o}$$

$$1 \le {}^{H}\!/\!{}_{h} \le 17$$

$$0 \cdot 0513mm \le h \le 0 \cdot 41mm$$

Where :

n	:	number of corrugations
R	:	the active radious (mm)
θ	:	profile angle
H/h	:	relative corrugation depth
h	:	the diaphragm thickness (mm)

Acceptable ranges for user specfdied maximum applied pressure are between 1/30 bar → 300 bar .

Corrugated diaphragms can have very large deflections exceeding the diaphragm thickness by a factor of 10 or more. Based on these observations, the acceptable user specified $p_{\max}$ and $\omega_{\max}$ ranges are :

$$\begin{gathered} 0\cdot00034 \, {}^{kg}\!/_{mm^2} \le p_{\max} \le 3\cdot0591 \, {}^{kg}\!/_{mm^2} \\ 0\cdot5mm \le \omega_{\max} \le 8\cdot2mm \end{gathered} \qquad (5.5.4.2)$$

This particular constrained non-linear programming problem is not easy to solve using conventional optimization methods (such as gradient based optimization techniques) as the first and second order partial derivatives of our objective function are themselves highly non-linear. In practice, to study NLP problems and their solution in depth, it is first necessary to identify and classify problems into different types so that each type may be studied separately. However, using genetic algorithms, we can treat the design optimization problem as a black box. Genetic algorithms require only a fitness measure (i.e., some non-negative figure of merit) and a finite problem coding.

As we have formulated the problem as a minimization, we transform the objective function to a fitness function using the following fitness mapping (as explained in section 5.5.3) :

$$u\left(\theta, n, h, {}^{H}\!/_{h}\right) = \frac{1\cdot0}{1\cdot0 + F\left(\theta, n, h, {}^{H}\!/_{h}\right)} \qquad (5.5.4.3)$$

For the specified search space, the design variables were discretized by mapping each variable in its specified range onto a 10 bit, binary unsigned integer. To form a complete representation of the problem, the parameter codings were concatenated to form a 4×10 = 40 bit string representing a particular diaphragm design.

In the following simulations, the Grefenstette's proposed cross-over and mutation rates, as described in section 5.5.3, are used. These settings are proved to give the best on-line performance. However, as in these simulations, we are also concerned with off-line performance and the ultimate global best design, we will use a higher population size. The population size affects both the ultimate performance and the efficiency of genetic algorithms. Genetic algorithms generally do poorly with very small populations because the population provides an insufficient sample size for most schemata. A large population is more likely to contain representatives from a large number of schemata. Hence, the genetic algorithm performance is improved by doing a more imformed search. As a result, a large population size discourages premature convergence to sub-optimal solutions. On the other hand, a large population requires more evaluations per generation, possibly resulting in an unacceptably slow rate of convergence. Based on these observations the population size is set to 100.

The following two design problems are tackled in this section :

Design-Problem-1 :

Required $p_{\max}$ = 2·0 kg/cm^2
Required $\omega_{\max}$ = 1·0 mm
Find a diaphragm with optimal characteristics and - minimal non-linearity error

Design-Problem-2 :

Required $p_{\max}$ = 150·5 kg/cm^2
Required $\omega_{\max}$ = 2·0 mm
Find a diaphragm with optimal characteristics and - minimal non-linearity error

These user specified requirements were chosen because one represents typical moderate settings for $p_{\max}$ and $\omega_{\max}$ and the other represents a typical high value of $p_{\max}$ and $\omega_{\max}$.

In all the optimization that follow, the Elitist strategy has been used. We recall that, the elitist strategy ensures the best performing structure to survive intact from one generation to the next. In the absence of this strategy, it is possible that the best structure disappears due to cross-over and mutation.

Initially, the three operator genetic algorithm was run using the stochastic remainder without replacement. For brevity, we call this algorithm as St-El genetic algorithm.

For each specified problem, five different random seeds were used. The best performance curves for each set of trials are shown in figure 5.8 and figure 5.9. The optimal solutions in each case are tabulated in table 5.2 below :

	Independent Design Parameters :				Dependent Design Parameters :			
	n	θ(rad.)	H/h	h(mm)	R(mm)	TER	%NLT	fitness
Problem-1	4	0·571	12·85	0·14	23·6	0·0074	0·1431	0·993
Problem-2	1	0·541	12·38	0·377	19·7	0·3111	0·18	0·762

Table 5.2 Best trial solutions using St-El Genetic Algorithm

As seen from table 5.2, the design solutions given by St-El are nearly optimal for practical purposes. However, there is scope for improvement here specifically for design problem-2. By refering to design requirements for our design problem-2, we note that a maximum pressure of 150·5 kg/cm^2 has forced the genetic algorithm to decrease the number of corrugations to a minimum value within the specified search space. In fact, the suggested number of corrugations, for this problem, has reached the constraint boundary. Therefore, the only way that the genetic algorithm can hope to reach better optimal designs is to manipulate the other three of the specified independent variables; keeping the number of corrugations at its constraint boundary. Our specified design problem-2 is a limiting case with respect to the restricted search space and proves to be much more difficult to solve relative to design problem-1. In fact, in this case, we are confronted with premature convergence. Looking at the gene pool after iteration 200, it was noted that most strings had almost similar fitness values. The main reason for this behaviour lies with the selection method we are using. We recall that, in schotastic remainder without replacement, each chromosome produces a number of offspring proportional to its fitness; with the restriction that the total number of chromosomes per generation remains constant. Thus, if the fitness of one chromosome is twice that of another, the superior chromosome

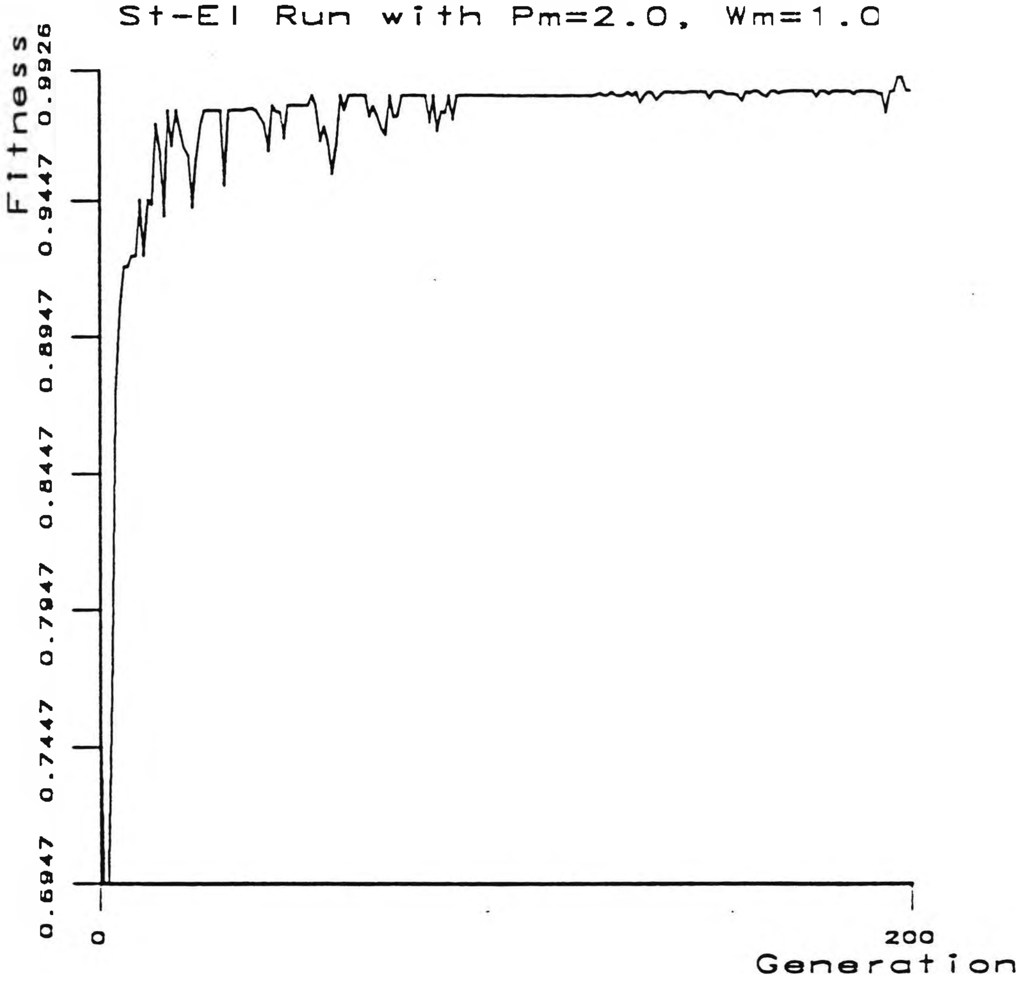


Figure 5.8 Best of Run, using the St-El selection (Design Problem-1)

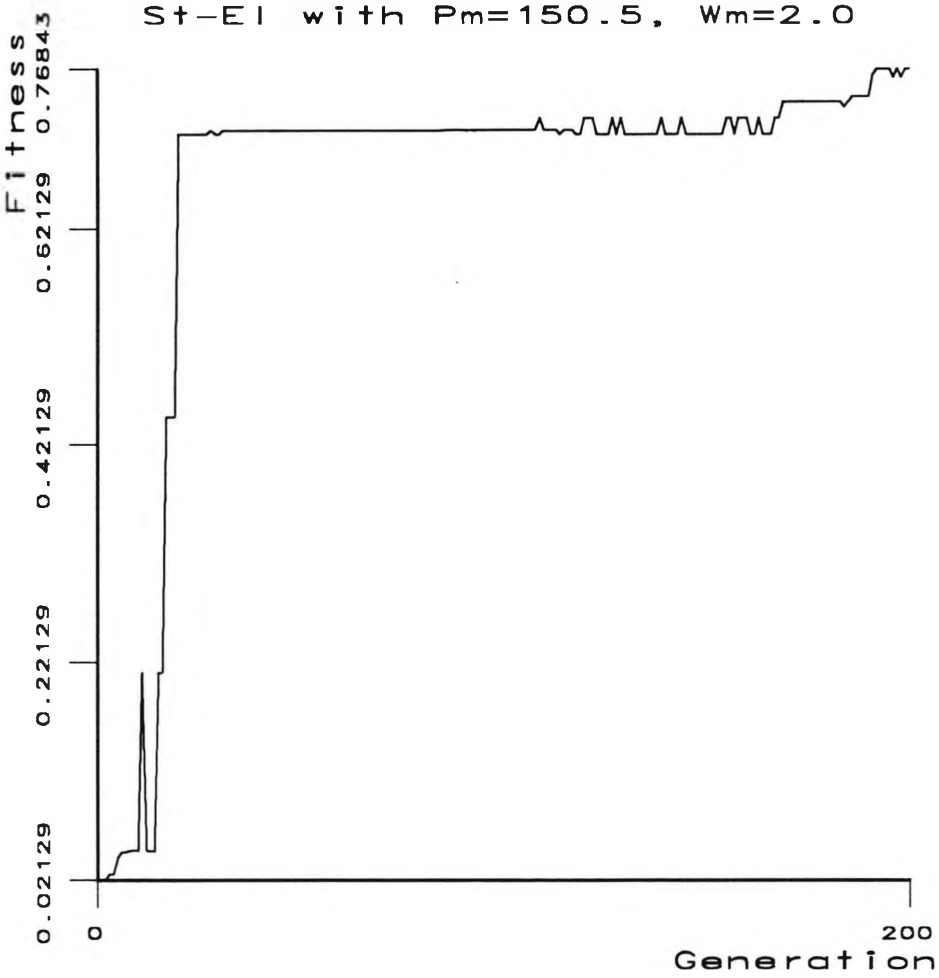


Figure 5.9 Best of Run. using the St-El selection (Design Problem-2)

would produce twice as many offspring. However, as we recall from section 5.2.1, there are two major problems with this method :

1- If all chromosomes have a similar fitness, each member in the population will produce one offspring. This results in little pressure to improve the fitness of the population.

2- If one chromosome has a fitness much larger than any other, that chromosome will create most, if not all, of the new offspring. The chromosome will dominate the population, resulting in a loss of genetic diversity.

These problems are the major causes of premature convergence. In order to remedy the first problem, we define a new parameter $u'_{\min}$ and rate each structure against this standard (Grefenstette, 1986). In our experiments, $u'_{\min}$ was set to the minimum $u(x)$ in the first generation. For each succeeding generation, those structures whose evaluation are less than $u'_{\min}$ were ignored in the selection procedure. In this method, we also define a scaling window parameter W which determines the updating period of the $u'_{\min}$ parameter with respect to the generation number. If $1 \langle W \langle 7$, then we set $u'_{\min}$ to the least value of $u(x)$ which occured in the last W generations. For example, suppose after several generations, the current population include only structures x for which $105 \langle u_i(x) \langle 110$. At this point, no structure in the population has a performance which deviates much from the average. This reduces the selection pressure towards the better structures, and the search stagnates. Using our scaling window, if $u(x_i)=110$ and $u(x_j)=105$, then if $u'_{\min}$=100, we can rate each structure against $u'_{\min}$. In this case $u'(x_i)=u(x_i)-u'_{\min}$=10, and $u'(x_j)=u(x_j)-u'_{\min}$=5; the performance of x_i now appears to be twice as good as the performance of x_j.

As before, for each of our specified design problems, five different runs using five different random seeds were performed. These results were obtained with the scaling window set to 1, i.e., $u'_{\min}$ was updated after each iteration.

The best performance curves, for each set of trials, are shown in figure 5.10 and figure 5.11. The optimal solutions in each case are tabulated in table 5.3.

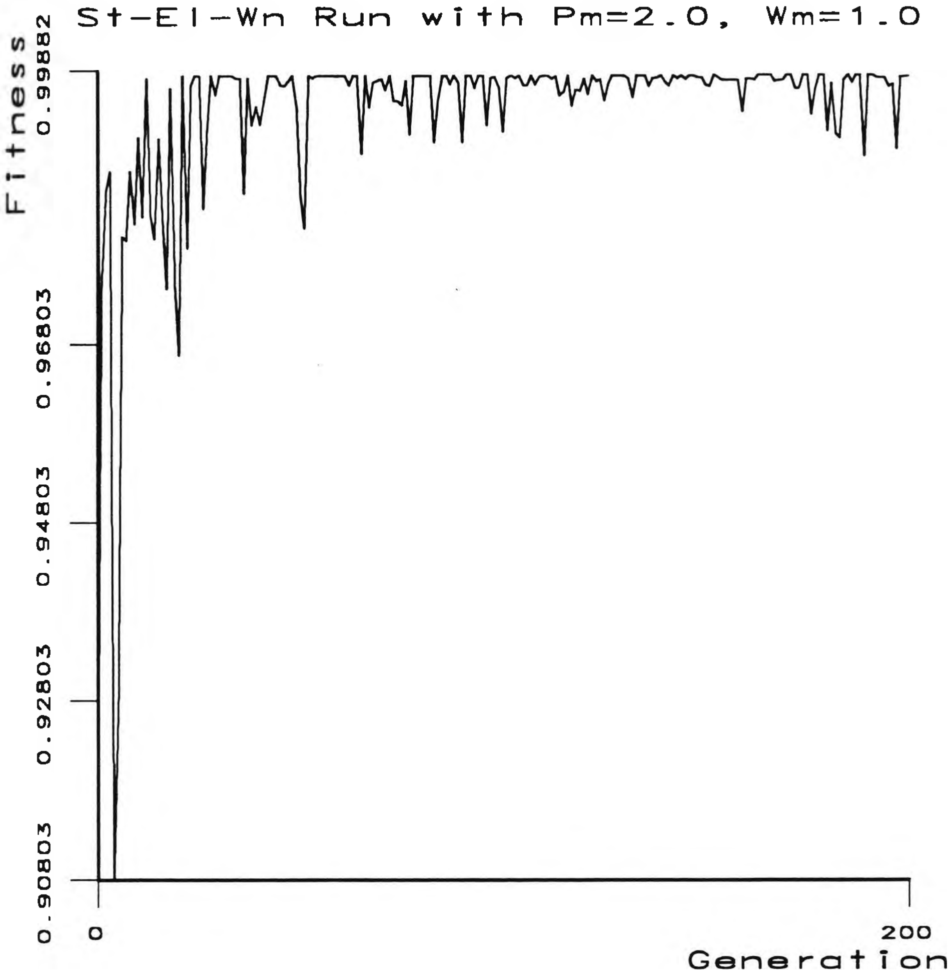


Figure 5.10 Best of Run, using the St-El-Wn selection (Design Problem-1)

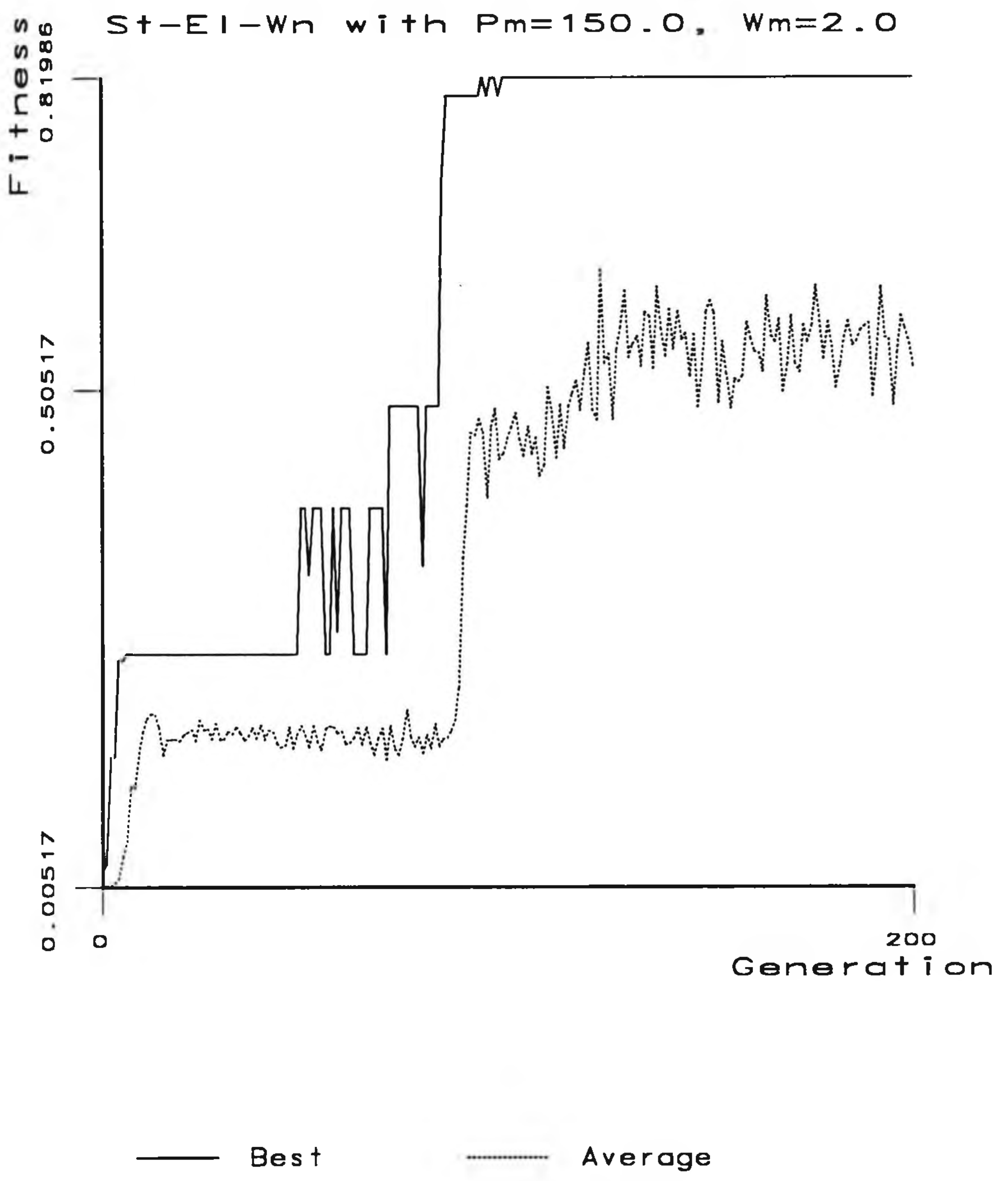


Figure 5.11 Best of Run, using the St-El-Wn selection (Design Problem-2)

	Independent Design Parameters :				Dependent Design Parameters :			
	n	θ(rad.)	H/h	h(mm)	R(mm)	TER	%NLT	fitness
Problem-1	3	0·499	11·63	0·39	49·2	0·0012	0·0036	0·998
Problem-2	1	0·51	15·53	0·30	17·9	0·2190	0·219	0·819

Table 5.3 Best trial solutions using St-El-Wn Genetic Algorithm

For brevity, the windowing technique used in conjunction with the St-El algorithm is called St-El-Wn. As seen from table 5.3, best trial solutions, using St-El-Wn genetic algorithm, are superior as compared to results obtained from St-El. In effect stochastic errors due to our proportionate selection algorithm have been reduced.

Although St-El-Wn can reduce the stochastic errors due to similarity of fitnesses near convergence, it is not effective in maintaining diversity across the gene pool. i.e., a chromosome with substantially higher fitness, will tend to dominate the population, causing premature convergence.

In order to investigate a different selection strategy, a ranking selection method (Baker, 1985) has been implemented. In this method, the whole population is first sorted by fitness. The number of offspring each chromosome generates is determined by how it ranks in the population. With the ranking method implemented, the top 5% of the population is allocated two offspring deterministically. The bottom 5% receives no offspring and the rest is allocated one offspring each. In this way, no one chromosome can overpower the population in a single generation, and no matter how close the actual fitness values are, there is always pressure to improve. The primary disadvantage of ranking is speed; because better chromosomes can not easily guide the population, forcing good answers to develope more slowly.

As before, the best performance curves, for each set of trials, are shown in figure 5.12 and figure 5.13 up to and including generation 600. The optimal results, in each case, are tabulated in table 5.4.

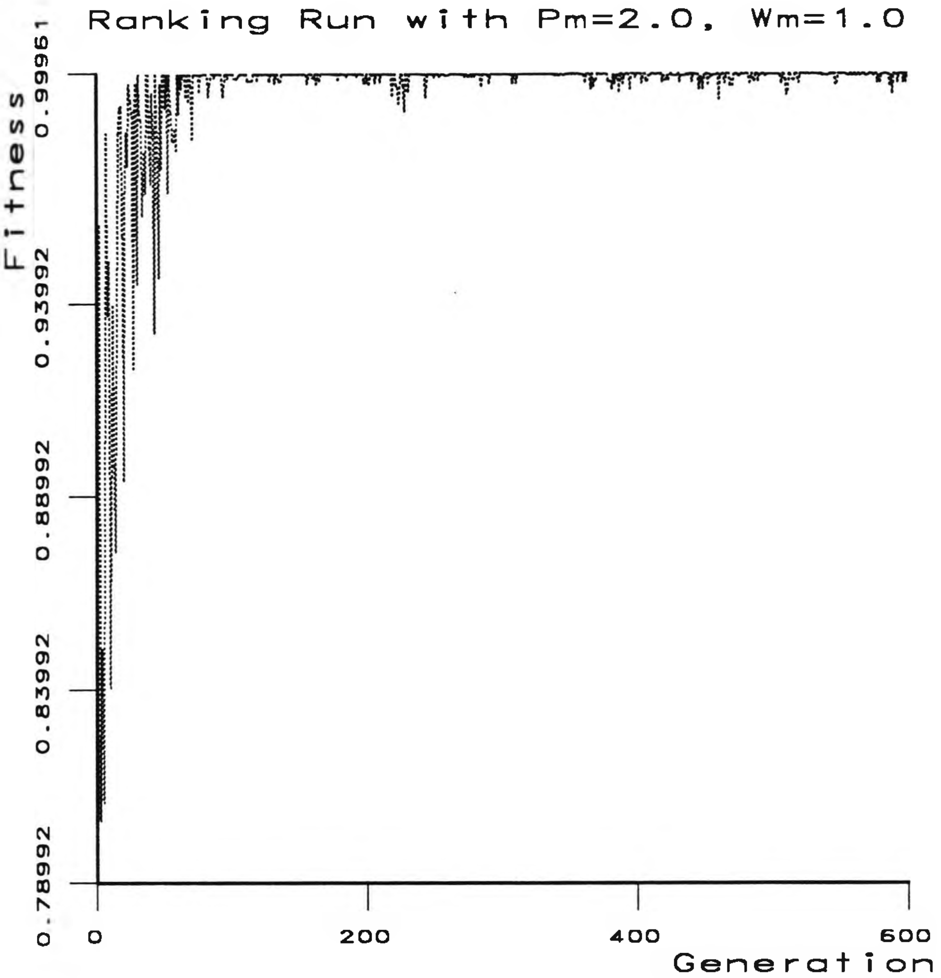


Figure 5.12 Best of Run, using the Ranking selection (Design Problem-1)

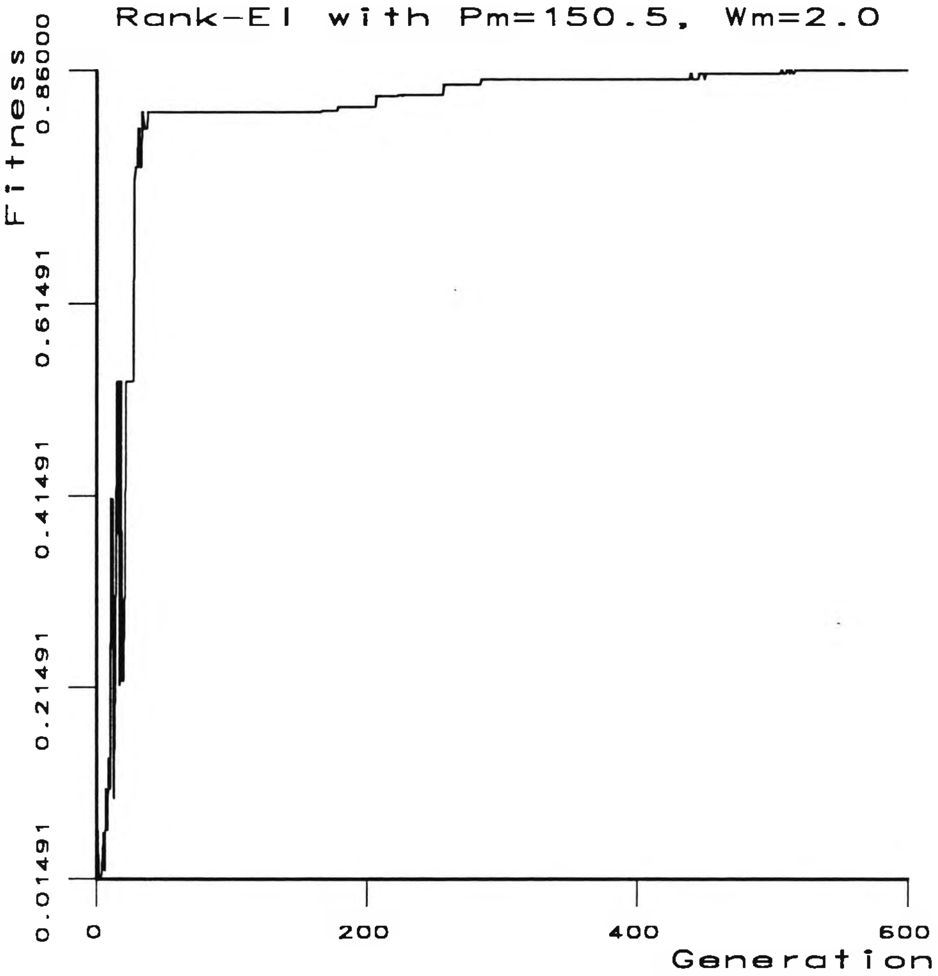


Figure 5.13 Best of Run, using the Ranking selection (Design Problem-2)

	Independent Design Parameters :				Dependent Design Parameters :			
	n	θ(rad.)	H/h	h(mm)	R(mm)	TER	%NLT	fitness
Problem-1	3	0·52	15·96	0·4	55·5	$3{\cdot}3\times10^{-4}$	$2{\cdot}8\times10^{-4}$	0·9996
Problem-2	1	0·50	14·4	0·4	21·8	0·162	$7{\cdot}7\times10^{-3}$	0·87

Table 5.4 Best trial solutions using Rank-El Genetic Algorithm

The results obtained from Rank-El are substantially better as compared to previous results and the characteristics of the near optimal designs are ideal in terms of percentage nonlinearity.

The ranking method used has been an effort to stop premature convergence. As discussed before, one cause of premature convergence may be a super genotype that has an unusually high fitness ratio and thus dominates the search process. The ranking method is effective in controlling the number of offspring allocated to each structure. Also, ranking method completely solves the scaling problem.

Figure 5.14 gives a comparison of best performance curves, for the three selection strategies used, up to and including generation 800.

In order to analyse the result obtained so far, we look back at equation (5.5.2.1). we notice that A and B can take on a wide range of possible values. This explains why the non-linear programming problem introduces a vast search space to our genetic algorithms. The overall performance of our genetic algorithms can be substantially improved if we introduce some restrictions on the possible variation of the starting population members. This is possible by reformulating the non-linear programming problem, as specified by equations (5.5.4.1) :

Knowing that (equation 5.5.2.1) :

$$p = A \cdot \omega_0 + B \cdot \omega_0^3$$

Where :

$$A = \frac{E}{R^4} \cdot h^3 \cdot a_p \qquad , \qquad B = \frac{E}{R^4} \cdot h \cdot b_p$$

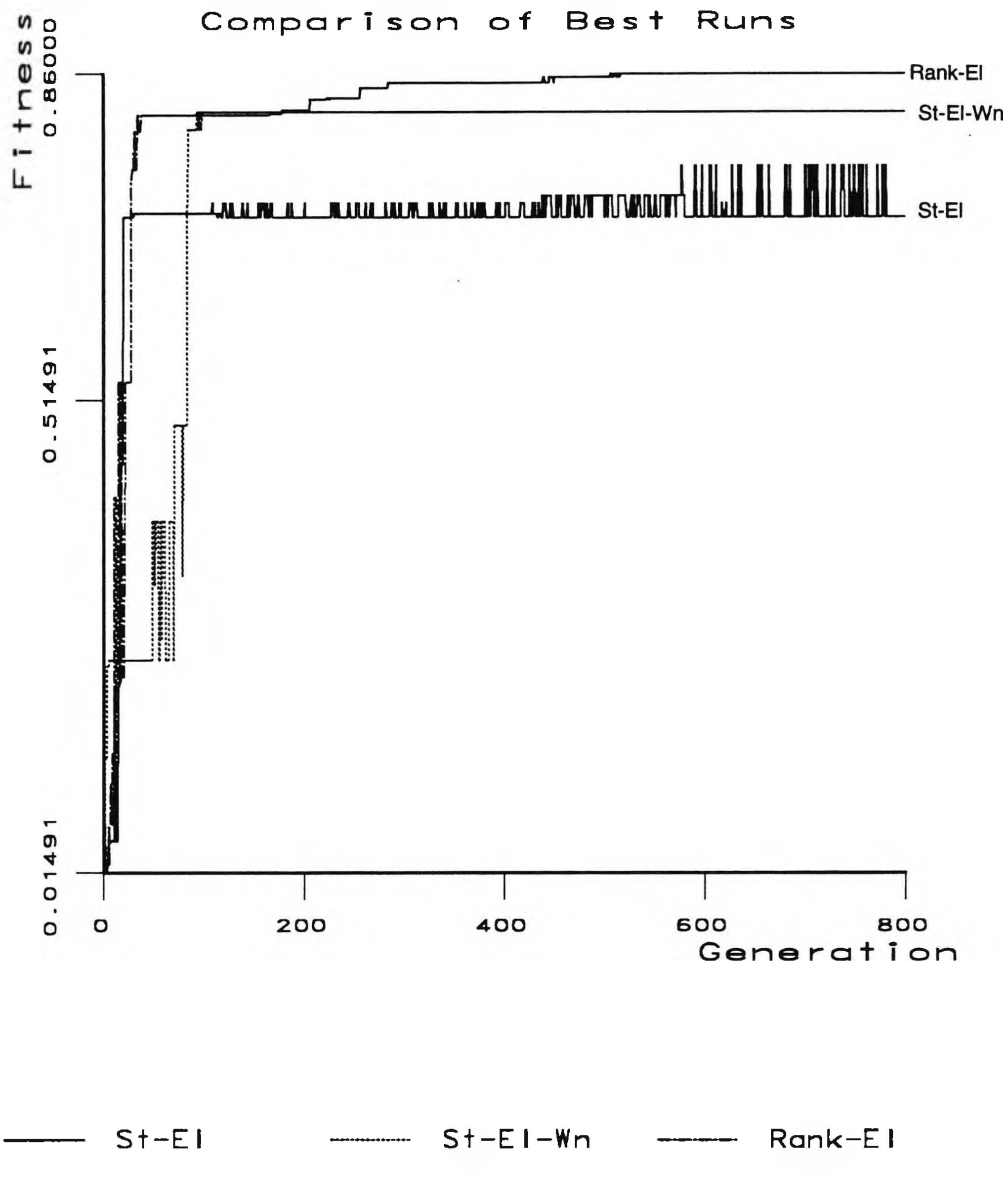

Comparison of Best Runs
Fitness
0.86000
0.51491
0.01491
0
200
400
600
800
Generation
Rank-El
St-El-Wn
St-El
St-El
St-El-Wn
Rank-El

Figure 5.14

$$\therefore \quad \frac{A}{B} = \frac{h^3}{h} \cdot \frac{a_p}{b_p} \longrightarrow B = \frac{A \cdot b_p}{h^2 \cdot a_p} \tag{5.5.4.4}$$

Using equation (5.5.4.4) the ideal design solution occurs at (when there are no constraints on the independent design variables) :

$$A \cdot \omega_{\max} + \left(\frac{A \cdot b_p}{h^2 \cdot a_p} \right) \cdot \omega_{\max}^3 = p_{\max} \tag{5.5.4.5}$$

$$A \longrightarrow \frac{p_{\max}}{\omega_{\max}}$$

$$B \longrightarrow 0$$

We recall from section 5.5.1 that :

$$a_p = \frac{2(3+\alpha)(1+\alpha)}{3 \cdot k_1 \left(1 - \frac{\mu^2}{\alpha^2} \right)}$$

$$b_p = \frac{32 \cdot k_1}{\alpha^2 - 9} \cdot \left[\frac{1}{6} - \frac{3-\mu}{(\alpha-\mu)(\alpha-3)} \right]$$

Where :

$$\alpha = \sqrt{k_1 \cdot k_2}$$

Analysis of a_p and b_p functions reveals that a_p is an strictly decreasing function of H/h. Also as the profile angle is increased, the non-linearity error decreases. Similarly, increases in H/h decreases the overall non-linearity error. These facts are only valid within the restricted ranges of our independent design variables. In fact, our fitness or utility function is characterized as having only one optimal global solution.

Equation (5.5.4.5) indicates that the optimal linear term of the diaphragm characteristics must be as close as possible to $p_{\max}/\omega_{\max}$ ratio.

The reformulation, as specified by equation (5.5.4.5), enables us to introduce the linear term of the diaphragm characteristics as an independent design variable. In this way, we have control over the ranges of the A parameter and can restrict the search space. This enables us to reduce the population size.

For design problem-2, the linear term of the characteristics is restricted to the following range :

$$1\cdot0 \langle A \langle 2\times\left({p_{\max}}\big/{\omega_{\max}} \right)$$

For design problem-2 we had :

$p_{\max}$	:	150·5 (kg/cm^2)
$\omega_{\max}$	:	2·0 mm

$$\therefore\ 1\cdot0 \langle A \langle 150.0$$

By restricting the linear term of the characteristics in this manner, the initial randomly generated population would be more likely to contain diaphragm designs with linear terms close to the ideal characteristics.

The population size, in the following simulations, has been decreased to 30. figure 5.15 shows the improved performance of the St-El-Wn as compared to figure 5.11. Figure 5.16 shows the relative superiority of the Rank-El as compared to St-El-Wn. Rank-El has been able to find the ideal global solution to this problem, within our restricted search space, after only 150 generations. The optimal results in each case are tabulated in table-4 below:

	Independent Design Parameters :				**Dependent Design Parameters :**			
	n	θ(rad.)	H/h	h(mm)	R(mm)	TER	%NLT	fitness
St-El-Wn	1	0·571	16·9	0·4	23·9	0·078	7·7×10^{-3}	0·92
Rank-El	1	0·576	17·0	0·4	24·2	0·06	1·6×10^{-4}	0·93

Table 5.5 Optimal solutions using restriction on *A*

5.6 Optimization under constraints

In previous section, we transformed our constrained NLP problem into a maximization problem by directly mapping each inequality constraint on a particular independent design variable (in its specified range) onto a 10 bit, binary unsigned integer. This process eliminates the inequality constraints.

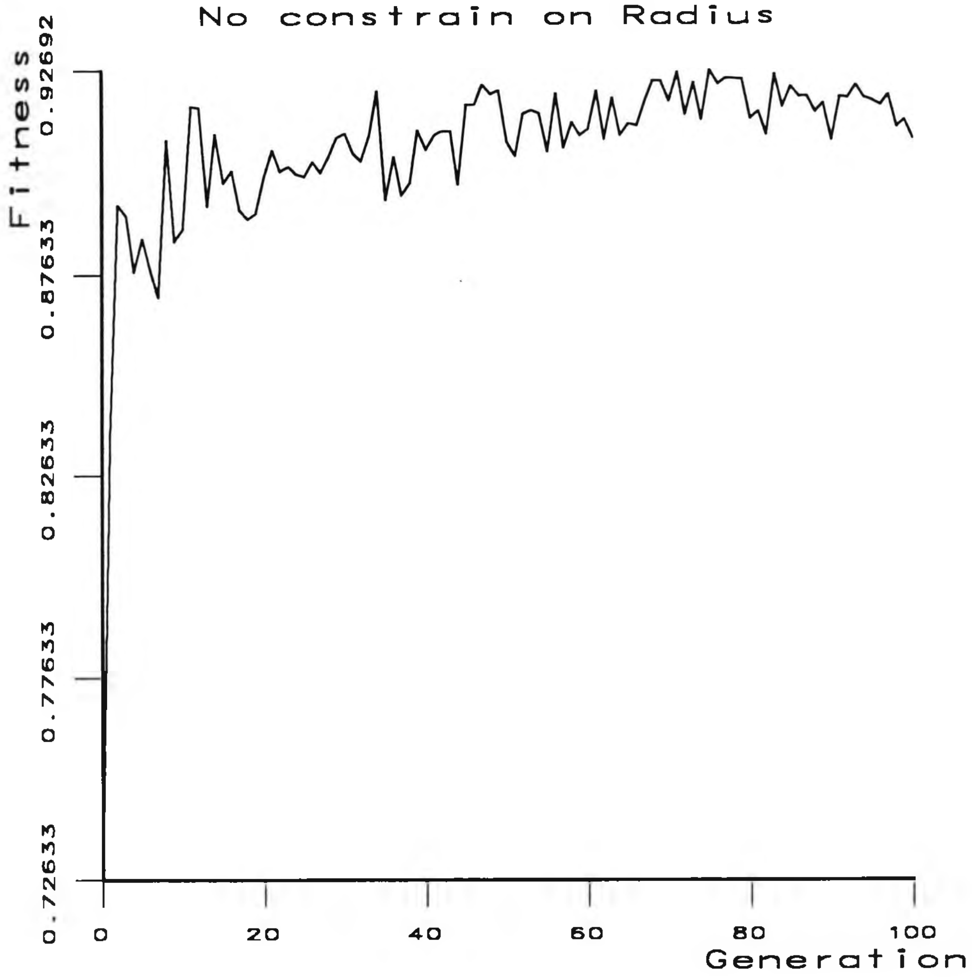


Figure 5.15 Best of Run, using the St-El-Wn selection with restriction on-the Linear Term of the Characteristics

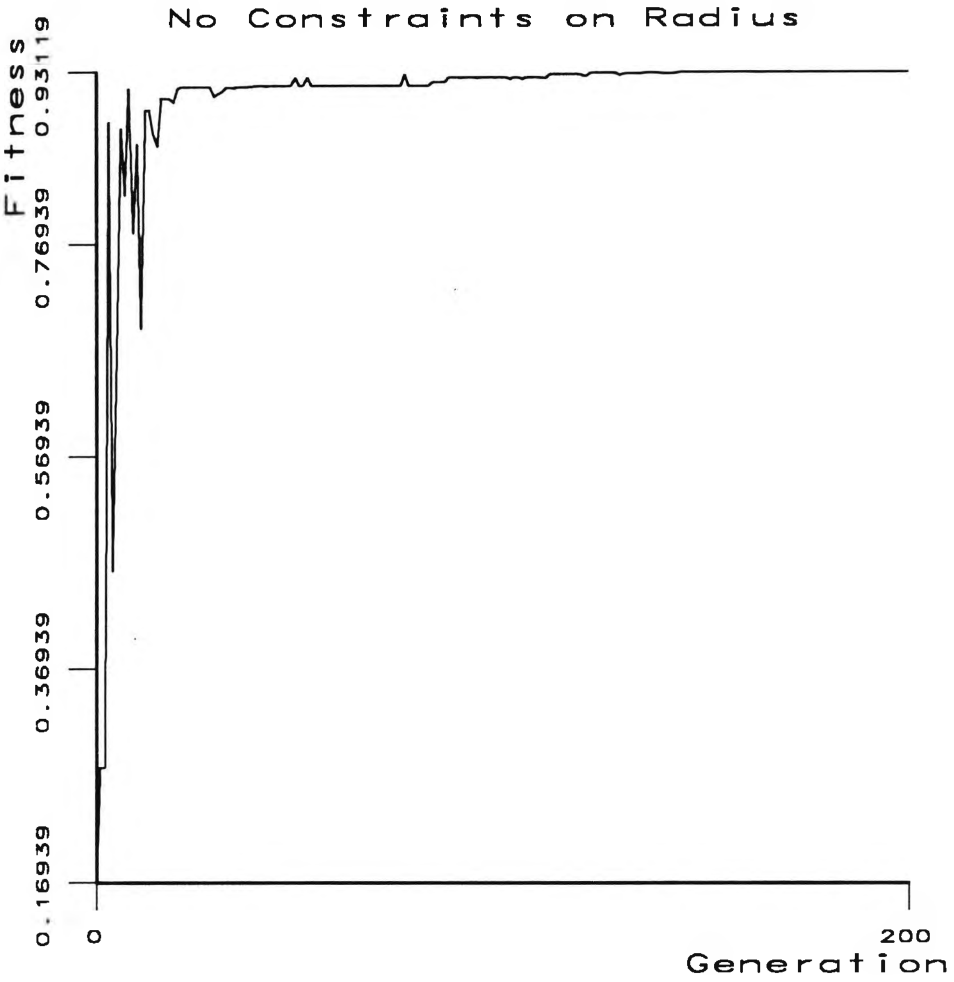


Figure 5.16 Best of Run, using the Ranking selection with restriction on- the Linear Term of the Characteristics

However, in practice there might be some constraints imposed on a set of independent design variables and this direct method of elimination will not be suitable. For example, in previous section, optimal diaphragm characteristics with minimal non-linearity error were obtained at certain optimal R values. These optimal unconstrained R values might not correspond to the user specified R value or ranges of acceptable R values.

Genetic algorithms are ideally suitable for unconstrained optimization problems. For constrained optimization problems, it is necessary to transform the optimal constrained problem into an unconstrained problem so that genetic algorithms can solve it.

Transformation techniques as suggested by Fiacco and McCormick (1968), using penalty functions, achieve this. Such transformations are ideally suitable for sequential search, where auxiliary derivative information and application dependent metrics is available. Genetic algorithms perform the search in parallel using pay-off information to direct the search. Hence, modifications in penalty techniques are necessary to make them practical for use in conjunction with genetic algorithms.

Considering the following problem with the single constraint $h(X)=0$.

$$\begin{array}{ll} \textit{Minimize} & f(X) \\ \textit{subject to} & h(X)=0 \end{array}$$

Suppose that this problem is replaced by the following unconstrained problem, where $\mu \rangle 0$ (i.e., the penalty constant) is a large number.

$$\begin{array}{ll} \textit{Minimize} & f(X)+\mu \cdot h^2(X) \\ \textit{Subject to} & X \in R_n \end{array}$$

We can intuitively see that, an optimal solution to the above problem must have $h^2(X)$ close to zero because otherwise a large penalty $\mu \cdot h^2(X)$ will be incurred. Now consider the following problem with the single inequality constraint $g(X)\leq 0$.

$$\begin{array}{ll} \textit{Minimize} & f(X) \\ \textit{subject} & g(X)\leq 0 \end{array}$$

It is clear that the form $f(X)+\mu\cdot g^2(X)$ is not suitable, since a penalty will be incurred whether $g(X)\langle 0$ or $g(X)\rangle 0$. In this case, a penalty is desired only if the point X is not feasible, that is, $g(X)\rangle 0$. A suitable unconstrained problem is, therefore, given by :

$$\begin{aligned} &\textit{Minimize} && f(X)+\mu\cdot\max\langle 0\,,\,g(X)\rangle \\ &\textit{Subject to} && X\in R_n \end{aligned}$$

If $g(X)\leq 0$, then $\max\langle 0,g(X)\rangle=0$, and no penalty is incurred. On the other hand, if $g(X)\rangle 0$, then $\max\langle 0,g(X)\rangle\rangle 0$, and the penalty term $\mu\cdot g(X)$ is applied.

5.6.1 Constrained optimization using genetic algorithms

In general, a suitable penalty function must incur a positive penalty for unfeasible points and no penalty for feasible points. If the constraints are of the form $g_i(X)\leq 0$ for i = 1, 2, ···, m , and $h_i(X)=0$ for i = 1, 2, ···, l , then a suitable penalty function α is defined by:

$$\alpha(X)=\sum_{i=1}^{m}\Phi[g_i(X)]+\sum_{i=1}^{l}\Psi(h_i(X)) \qquad (5.6.1.1)$$

Where Φ and Ψ are continuous functions satisfying the following :

$$\begin{aligned} &\Phi(y)=0 \;\; \textit{if} \;\; y\leq 0 \quad \textit{and} \quad \Phi(y)\rangle 0 \;\; \textit{if} \;\; y\rangle 0 \\ &\Psi(y)=0 \;\; \textit{if} \;\; y=0 \quad \textit{and} \quad \Psi(y)\rangle 0 \;\; \textit{if} \;\; y\neq 0 \end{aligned}$$

Typically, Φ and Ψ are of the forms :

$$\begin{aligned} &\Phi(y)=[\max\{0,y\}]^p \\ &\Psi(y)=|\,y\,|^p \end{aligned} \qquad (5.6.1.2)$$

Where p is a positive integer. Thus the penalty function α is usually of the form:

$$\alpha(X)=\sum_{i=1}^{m}[\max\{0,g_i(x)\}]^p+\sum_{i=1}^{l}|h_i(x)|^p \qquad (5.6.1.3)$$

Fiacco and McCormick (1968) have proved that the optimal solution of the penalty function can be made arbitrarily close to the optimal objective value of the original primal problem by choosing the penalty coefficient μ arbitrarily large. In other words, the optimal solution of the penalty function formulation will approach to that of the primal problem as μ becomes relatively large, and at $+\infty$, the optimal solution of the penalty function is, in fact, equivalent to that of the primal problem. Penalty function methods, as used in conventional optimization, do not impose any restrictions on f, g and h other than that of continuity. Therefore, theoretically, they can be used for non convex programming problems where lagrangian methods fail to provide an optimum solution (because of the presence of a duality gap (Bazaraa, 1979).

We racall that, Genetic algorithm search techniques use only pay-off information to direct the the search, making them independent of a particular application domain. Penalty methods, as described by equation (5.6.1.1), have been used in conjunction with conventional search techniques, where auxiliary derivative or other local information is available.

Direct use of exterior penalty methods, as suggested by equation (5.6.1.1), is unsuitable in GA search. This is due to the fact that, conventional penalty methods impose harsh restrictions on a GA search so that a GA will avoid forbidden non feasible spaces. In genetic algorithms, search is based on using and combining partial information from all search points. Therefore, the infeasible solutions should provide information and not just be thrown away.

The penalty techniques used in conjunction with GAs is constructed by using a linear combination of a cost function and a penalty function. The cost function is typically well defined from the problem formulation, but writing the penalty function and combining it with the cost function is a difficult research issue.

Theoretical investigations (Richardson, 1989), suggest that on sparsely feasible problems, with relatively few constraints, penalties formed only based on the number of violated constraints will not produce satisficing solutions. However, if given information about how far from feasible the points are, the genetic algorithm then has some idea of how to order the search points in a way leading to feasibility. Therefore, penalty functions which are functions of the distance from feasibility perform better than those which impose harsh restrictions on violations of constraints.

As a practical matter, in genetic algorithmic search, penalty coefficients (i.e., μ values) are sized for each type of constraint so that moderate violations of the constraints yield a penalty which is a significant percentage of some nominal operating cost. A number of alternatives exist for the penalty function $\alpha(X)$ (equation 5.6.1.1) used in conjunction with GAs. A quadratic function (i.e., p=2 in equation 5.6.1.3) is usually satisfactory, but some experimentation is needed in relating the sizes of penalties to the degree of violation of the constraint.

Suppose that for design problem-2 of section 5.5.4, the user specifies the following constraint on the R value :

$$R \leq 15mm$$

Looking back at table 5.5, we know that the optimal, unconstrained R value for this problem is 24·2 mm. In order to penalize a constraint violation on the desired R value, we will subtract 100% of our nominal fitness value for a 10% violation of the constraint. This gives a μ value of 0·25. Therefore, our constraint formulation becomes :

$$u'\left(\theta,n,h,{}^{H}\!/_{h}\right) = u\left(\theta,n,h,{}^{H}\!/_{h}\right) - 0\cdot 25 \times \left[\min\{0,(15-R)\}\right]^{2} \qquad (5.6.1.4)$$

It is important to contrast above formulation with equation (5.6.1.3). The difference is due to the fact that we want to maximize our utility function.

As before five different runs, using five different random seeds, were performed using our St-El-Wn and Rank-El algorithms. Best of run performance curves, for each algorithm, are shown in figure 5.17 and figure 5.18. The optimal solutions in each case are tabulated in table 5.6 below:

Problem-2	μ	n	R(mm)	θ(Rad)	H/h	h(mm)	TER	%NLT	Fitness
St-El-Wn	0·25	1	14·43	0·52	16·93	0·21	0·31	0·2	0·76
Rank-El	0·25	1	15·13	0·55	16·97	0·21	0·27	0·18	0·78

Table 5.6 μ=0·25

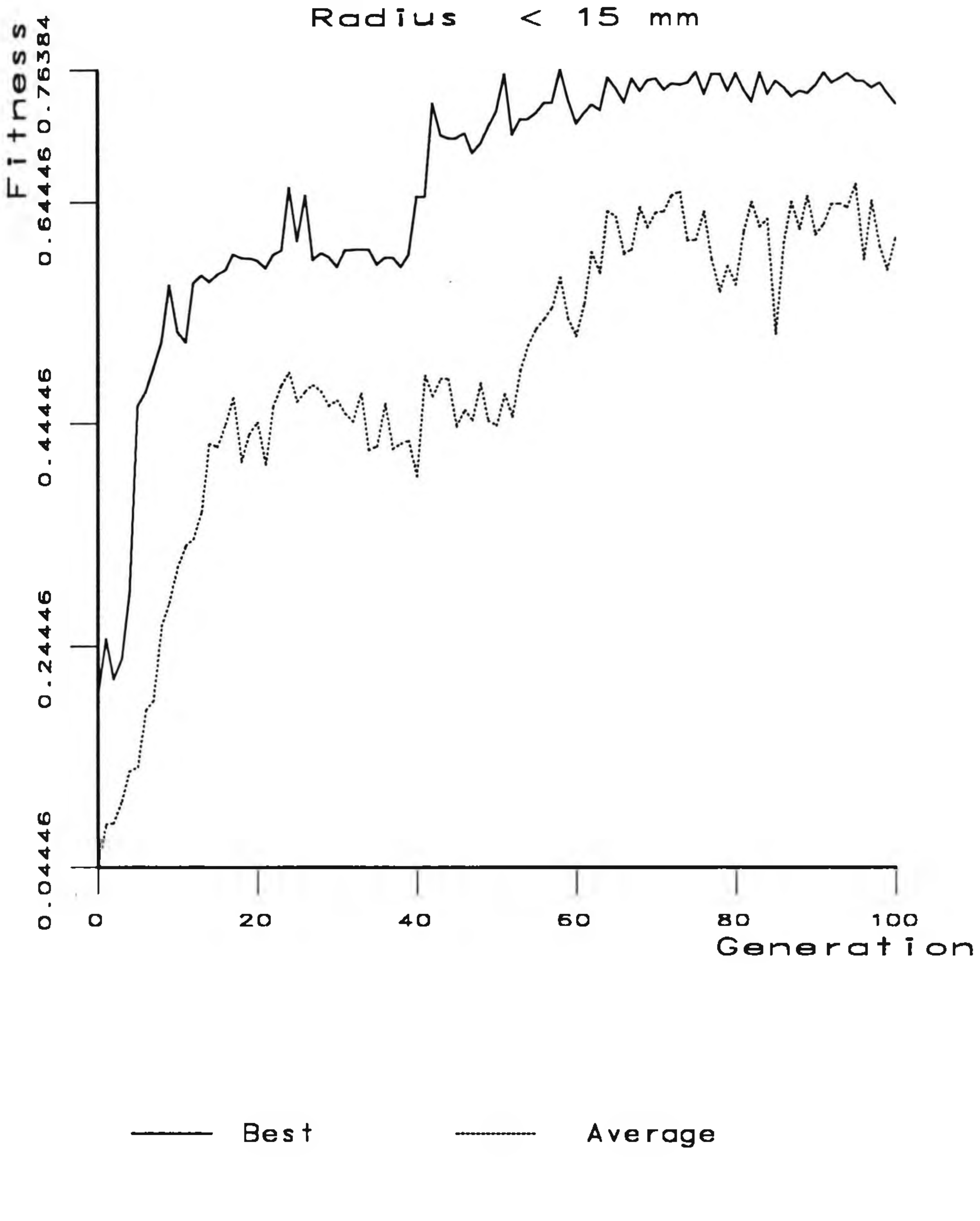
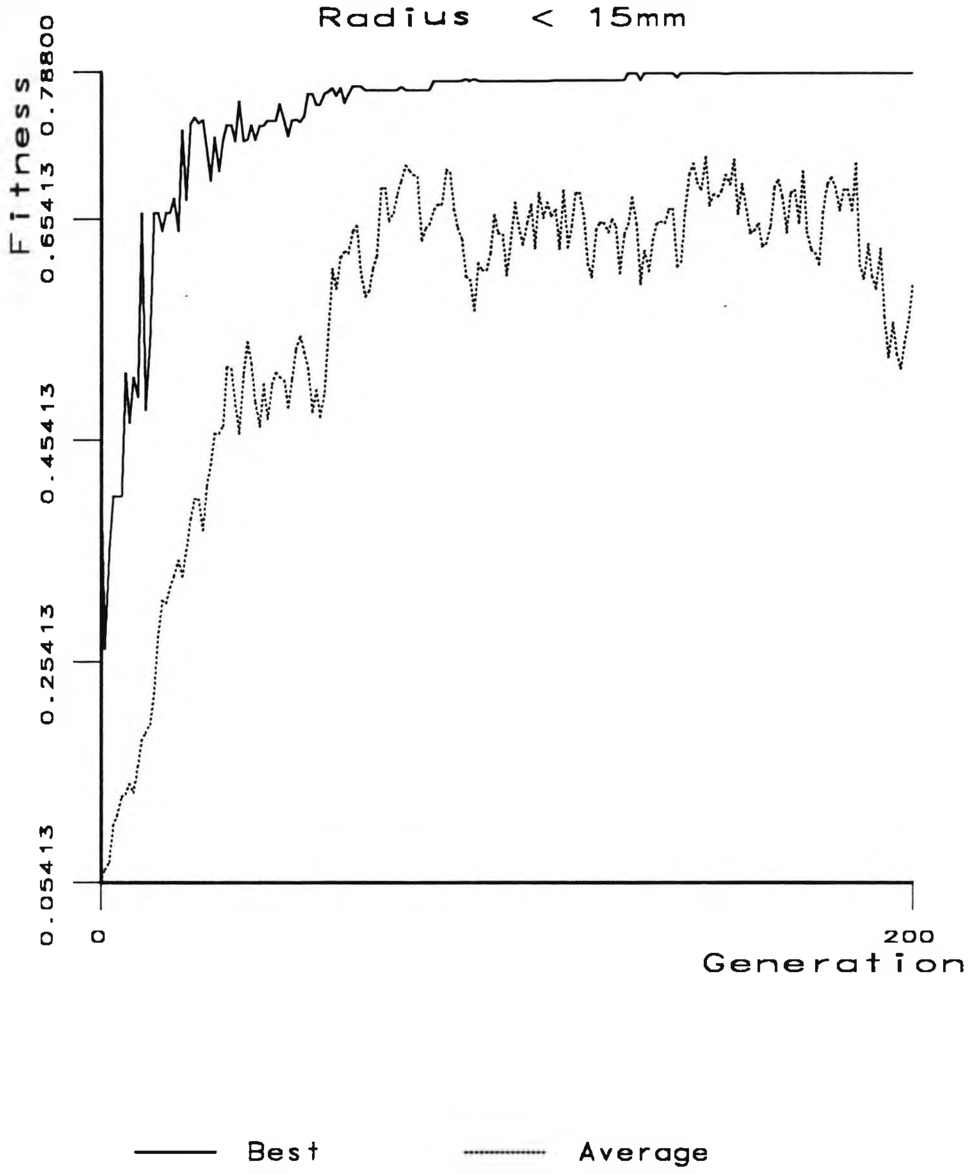


Figure 5.17 Best of Run, using the St-El-Wn selection with constraint on the active radius of the Diaphragm



Fiqure 5.18 Best of Run. usinq the Rankinq selection with constraint on-the active radius of the Diaphraqm

From table 5.6 above, we note that, user specified restrictions on R value, lower than unconstrained optimal R value, will produce solutions with a higher overall % non-linearity error. Also, in the case of Rank-El, the lower % non-linearity is obtained in the expense of a 0·13 mm constraint violation resulting in an increase in the overall fitness value.

There is a trade-off between the severity of the penalty imposed upon the constraint on R and the % non-linearity obtained. If we subtract 50% of our nominal fitness value for a 10% violation of the constraint (i.e., reduce our μ to 0·125), we would expect to obtain a lower % non-linearity error in the expense of a higher R value. Table 5.7 below, using Rank-El, confirms this expectation.

Problem-2	μ	n	R(mm)	θ(Rad)	H/h	h(mm)	TER	%NLT	Fitness
Rank-El	0·125	1	15·4	0·577	16·99	0·2	0·26	0·15	0·79

Table 5.7 μ=0·125

Therefore, the reduced penalty coefficient means a more relaxed penalty imposed upon the constraint violation. These trade-offs must be considered when there are conflicting design objectives.

From our results, obtained so far in the context of design optimization of corrugated diaphragms, we conclude that our ranking selection strategy exhibits substantially better performance as compared to our proportionate selection schemes. This observation will be exploited in the next section where we will consider the design optimization of linear variable differential transformers.

5.7 Genetic Algorithms for the design optimization of LVDTs

In this section, we apply our three operator genetic algorithms to the problem of design and optimization of LVDTs using an analytical model.

The LVDT is an inductive displacement sensor. It operates on the transformer principle with varying mutual inductance couplings between a primary and two secondary coils. This coupling is induced by the movement of a plunger (magnetic iron material) along the axis of the coils. In a standard design of LVDT, the three coils are placed symmetrically side by side on the same axis, the primary being in the middle. The plunger (armature) is attached with a non-magnetic shaft or rod to any assembly from which displacement is required to be measured.

In the following sections, an analytical model for the analysis and design of LVDTs is presented together with objective functions and constraints. The objective function and constraints are incorporated into a penalty function procedure for optimization of LVDTs. The penalty function is of the form explained in section 5.6.1. It is used for fitness mapping; making it appropriate for genetic algorithmic manipulations.

5.7.1 An analytical model for the design of LVDTs

Atkinson and Hynes (Neubert, 1975) developed an analytical mathematical model by assuming a distribution for the magnetic flux generated by the primary coils. As shown in figure 5.19, next page, this assumes that the flux is zero outside the armature region, linearly varying through the primary region, and constant in the other two regions.

The coils are surrounded by a stator. The centre coil (primary) is energised with alternating current, and the outer coils (secondaries), which are identically wound and symmetrically spaced with respect to the primary, are connected in series opposition. The induced differential voltage in the secondaries varies according to the position of the armature, thus providing a means of measuring the displacement of the armature from some null position.

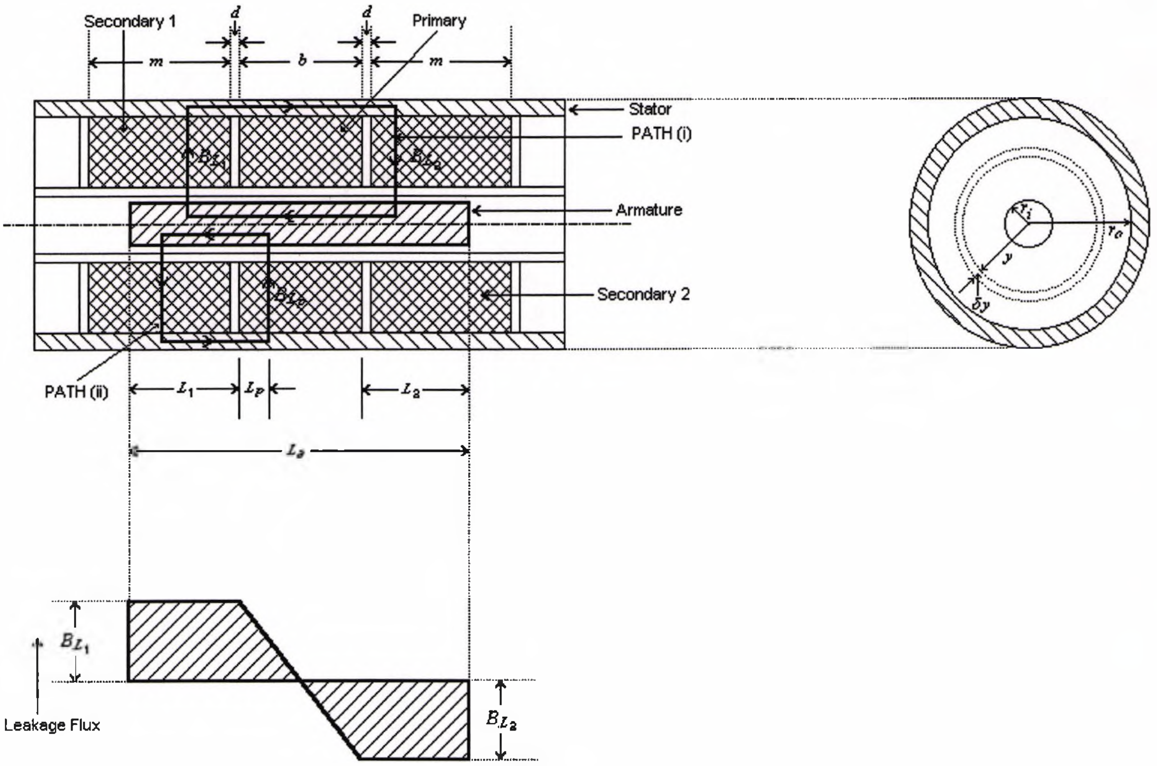


Figure 5.19

The armature is an iron cylinder of length L_a and radious r_i. The coils have external radious r_o and internal radious r_i. The primary has width b and the secondaries have width m; the secondaries are separated from the primary by a distance d (Figure 5.19).

Considering a magnetic path such as (i) in figure 5.19, if A represents the ampere-turns of the primary, l the path length and H the field strength, then:

$$\oint H \cdot dl = A = N_p \cdot I_p \tag{5.7.1.1}$$

Where I_p is the primary current and N_p its number of turns. The contour integral of the closed path can be expressed by:

$$\oint H \cdot dl = \int_{air\ path} H \cdot dl_{air} + \int_{iron\ path} H \cdot dl_{iron} \tag{5.7.1.2}$$

The difference in reluctance R between the air gap and the iron is large ($R_{air} \rangle\rangle R_{iron}$). The integral of the field intensity over the iron path is thus

negligible as compared to the corresponding integral over the air gap. From equation (5.7.1.1) and (5.7.1.2), it follows that:

$$\int_{air\ path} H \cdot dl = N_p \cdot I_p \qquad (5.7.1.3)$$

Further, the flux from the end faces of the slug has been neglected; Fringing can be decreased by making the two end faces hemispherical. This is in fact the case with the movable core of this transducer.

If y is the radious of an elementary ring width δy (figure 5.19) in the cross section of a secondary coil, then the flux density in the ring is $B_L \cdot {}^{r_i}\!/\!_{y}$, where B_L is the leakage flux density at the surface of the armature ($y = r_i$). If B_{L_1} and $-B_{L_2}$ denote these leakage flux densities at the armature surface over L_1 and L_2 respectively, we have:

$$\int_{air\ path} H \cdot dl = \frac{1}{\mu_0} \int_{r_i}^{r_o} \left(B_{L_1} - B_{L_2} \right) \frac{r_i}{y} \cdot dy \qquad (5.7.1.4)$$

Assuming that the magnetic material is linear within the saturation limit, (i.e., $B = \mu_0 \cdot H$), where μ_0 is the permeability of free space.

Substituting equation (5.7.1.4) into equation (5.7.1.3) and integrating, we have:

$$B_{L_1} - B_{L_2} = \frac{\mu_0 \cdot N_p \cdot I_p}{r_i \cdot \ln\left({}^{r_o}\!/\!_{r_i} \right)} \qquad (5.7.1.5)$$

Similarly we can consider a path such as (ii) in figure 5.18. Consider a cross-section at a distance L_p from one end of the primary. The ampere-turns in the volume thus cut off is $N_p \cdot I_p \cdot {}^{L_p}\!/\!_{b}$. If B_{L_p} denotes the leakage flux densities at the surface of the armature at this cross-section we have, by the same argument as before:

$$B_{L_p} = B_{L_1} - \frac{\mu_0 \cdot N_p \cdot I_p \cdot L_p}{b \cdot r_i \cdot \ln\left({r_o}/{r_i} \right)} \tag{5.7.1.6}$$

$$\textit{when} \qquad L_p = 0\,,\, B_{L_p} = B_{L_1}$$

$$\textit{and when} \quad L_p = b\,, B_{L_p} = B_{L_2}$$

To obtain a further relation between B_{L_1} and B_{L_2}, the flux on the surface of the armature is considered. From Gauss' theorem, the flux over the armature surface with no sources is given by:

$$B_{L_1} \cdot s_1 + B_{L_2} \cdot s_2 + \int_s B_{L_p}(s) \cdot ds = 0 \tag{5.7.1.7}$$

Where s, s_1 and s_2 are the areas over which the corresponding flux densities are measured, so in this case:

$$B_{L_1} \cdot 2\pi\, r_i \cdot L_1 + 2\pi\, r_i \int_0^b B_{L_p} \cdot dL_p + 2\pi\, r_i \cdot B_{L_2} \cdot L_2 = 0 \tag{5.7.1.8}$$

Substituting from equation (5.7.1.6) for B_{L_p} into equation (5.7.1.8) and integrating we get:

$$B_{L_1} = - B_{L_2} \frac{(2L_2 + b)}{(2L_1 + b)} \tag{5.7.1.9}$$

Substituting for B_{L_2} from above into equation (5.7.1.5), we get:

$$B_{L_1} = \frac{(2L_2 + b)}{2 \cdot L_a} \cdot \frac{\mu_0 \cdot N_p \cdot I_p}{r_i \cdot \ln\left({r_o}/{r_i} \right)} \tag{5.7.1.10}$$

Using equations (5.7.1.9) and (5.7.1.5) for B_{L_2} we get:

$$B_{L_2} = \frac{-(2L_1 + b)}{2 \cdot L_a} \cdot \frac{\mu_0 \cdot N_p \cdot N_s}{r_i \cdot \ln\left({r_o}/{r_i} \right)} \tag{5.7.1.11}$$

To find the induced voltages in the secondaries, let $\Phi_{x'}$ be the flux linking an elementary coil, width $\delta x'$, distance x' from the inner side of one of the secondaries, say coil 1. Let the armature penetrate a distance x_1 as shown in figure 5.20 below:

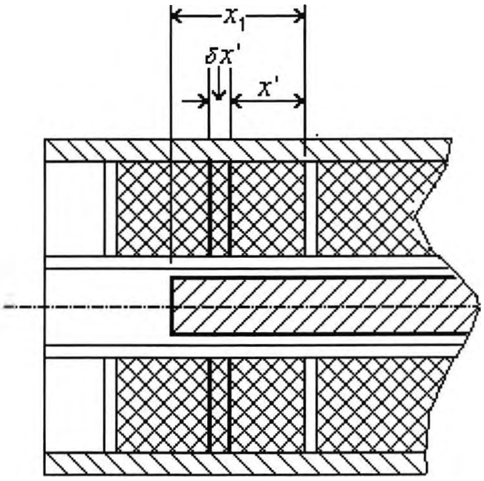


Figure 5.20

Then, if N_s is the total number of turns on the secondary, the number of turns in the elementary coil is $N_s \cdot {}^{\delta x'}\!/_{m}$ and the flux linking it is $\Phi_{x'} = B_{L_1} \cdot 2\pi\, r_i \cdot x'$. Hence, the total flux turns linked in the secondary are:

$$\lambda_1 = \int_0^{x_1} \Phi_{x'} \cdot \frac{N_s}{m} \cdot dx' \qquad (5.7.1.12)$$

which is equal to:

$$\begin{aligned} \lambda_1 &= \frac{2\pi \cdot r_i \cdot B_{L_1} \cdot N_s}{m} \int_0^{x_1} x' \cdot dx' = \frac{\pi \cdot r_i \cdot B_{L_1} \cdot N_s \cdot x_1^2}{m} \\ &= \frac{\pi \cdot r_i \cdot N_s}{m} \cdot \frac{(2L_2 + b)}{2 \cdot L_a} \cdot \frac{\mu_0 \cdot N_p \cdot I_p}{r_i \ln\left({}^{r_o}\!/_{r_i}\right)} \cdot x_1^2 \end{aligned} \qquad (5.7.1.13)$$

If the primary is excited with alternating current of frequency f, i.e., if we let:

$$A_p = \hat{A} \sin 2\pi f \cdot t$$

Then the induced r.m.s voltage, in coil-1, is given by:

$$e_1 = \frac{d\lambda_1}{dt} = \frac{4\pi^3}{10^7} \cdot \frac{f \cdot I_p \cdot N_p \cdot N_s}{\ln\left({}^{r_0}\!/_{r_i}\right)} \cdot \frac{(2L_2 + b)}{m \cdot L_a} \cdot x_1^2 \qquad (5.7.1.14)$$

Where I_p is the r.m.s value of the primary excitation current and N_p the number of turns of the primary coil.

Similarly, the induced r.m.s voltage e_2, in coil-2, is given by:

$$e_2 = \frac{d\lambda_2}{dt} = \frac{4\pi^3}{10^7} \cdot \frac{f \cdot I_p \cdot N_s \cdot N_p}{\ln\left(\frac{r_o}{r_i}\right)} \cdot \frac{\left(2L_1 + b\right)}{m \cdot L_a} \cdot x_2^2 \qquad (5.7.1.15)$$

The differential voltage $e = e_1 - e_2$ is then given by:

$$e_1 - e_2 = \frac{4\pi^3}{10^7} \cdot \frac{f \cdot I_p \cdot N_p \cdot N_s}{\ln\left(\frac{r_o}{r_i}\right) \cdot m \cdot L_a} \cdot \left\{\left(2L_2 + b\right)x_1^2 - \left(2L_1 + b\right)x_2^2\right\} \qquad (5.7.1.16)$$

Letting $\frac{4\pi^3}{10^7} \cdot \frac{f \cdot I_p \cdot N_p \cdot N_s}{\ln\left(\frac{r_o}{r_i}\right) \cdot m \cdot L_a} = EXPR$ and knowing that (figure 5.19):

$$L_1 = x_1 + d$$
$$L_2 = x_2 + d$$

We can rewrite equation (5.7.1.16) as :

$$e_1 - e_2 = 2 \cdot EXPR \cdot \left(x_1 - x_2\right)\left\{2x_1 x_2 + (b + 2d)\left(x_1 + x_2\right)\right\}$$

or:

$$e_1 - e_2 = 4 \cdot EXPR \cdot \left(\frac{x_1 - x_2}{2}\right)\left\{(b + 2d)\left(\frac{x_1 + x_2}{2}\right) + x_1 x_2\right\} \qquad (5.7.1.17)$$

We note that $\left(\frac{x_1 - x_2}{2}\right)$ is the armature displacement which we denote as x. Denoting $\left(\frac{x_1 + x_2}{2}\right)$ by x_0, with some simple manipulation we can rewrite equation (5.7.1.17) as :

$$e = e_1 - e_2 = 4 \cdot EXPR \cdot x \cdot \left(1 - \frac{x^2}{(b + 2d) \cdot x_0 + x_0^2}\right)$$

or :

$$e = e_1 - e_2 = k_1 \cdot x \cdot \left(1 - \frac{x^2}{k_2}\right) \qquad (5.7.1.18)$$

Where :

$$k_1 = \frac{16 \cdot \pi^3 \cdot f \cdot I_p \cdot N_p \cdot N_s \cdot \{(b+2d)x_0 + x_0^2\}}{10^7 \cdot \ln\left({r_o}/{r_i} \right) \cdot m \cdot L_a}$$

$$k_2 = (b+2d) \cdot x_0 + x_0^2$$

$$x = \frac{1}{2}(x_1 - x_2)$$

$$x_0 = \frac{1}{2}(x_1 + x_2) = \frac{1}{2}(L_a - b - 2d)$$

k_1 represents the sensitivity of the device (i.e., ${e}/{x}$) and ${x^2}/{k_2}$ represents the error in linearity, since it is a measure of the departure of $e = k_1 x\left(1 - {x^2}/{k_2}\right)$ from the straight line $e = k_1 x$.

The impedance of the LVDT primary is inductive, $(R_p + jX_p)$. The inductance is comparatively small, so the power factor is close to unity in the low frequency range of operation. The differential secondary impedance is also inductive (i.e., $R_s + jX_s$) . An LVDT, having a special excitation voltage rating or primary impedance, can often be built to satisfy a particular application. Since the primary-to-secondary coupling is relatively loose, the secondary impedance and loading have little influence on the primary impedance or input current. Therefore, having specified a nominal operating excitation voltage, the primary excitation current is calculated as :

$$I = \frac{V}{R_p + j \cdot X_p} \qquad (5.7.1.19)$$

Where :

$$V = v_o e^{j\omega t}$$

$$I = i_o e^{j\left(\omega t - \arctan\left({X_p}/{R_p}\right)\right)}$$

5.7.2 The electrical parameters of the LVDT

The electrical resistance of each one of the coils is made up of N turns, each turn having a resistance r given by:

$$r = \rho \cdot \frac{l_{avg}}{a}$$

Where:

ρ	:	resistivity of wire $(\Omega \cdot m)$
l_{av}	:	Mean length of turn (m)
a	:	Cross sectional area of wire (m^2)

The total resistance of each coil is then:

$$R = \rho \cdot N \cdot \frac{l_{av}}{a}$$

Where:

$$l_{av} = 2\pi \frac{r_i + r_o}{2} = \pi \left(r_i + r_o\right) ,$$

and r_i and r_o are the inside and outside radii of the coil respectively. Thus:

$$R = \rho \cdot \pi \cdot \frac{N}{a}\left(r_i + r_o\right) \qquad (5.7.2.1)$$

If, for each coil of a candidate LVDT design, the following parameters are given:

r_i	:	Inner radius of coil
r_o	:	Outer radius of coil
l	:	Length of coil
N_v	:	Number of turns per unit volume

We can calculate the cross sectional area of wire as follows (neglecting air spacing between wires at this stage):

$$a = \frac{\pi\left(r_e^2 - r_i^2\right)\cdot l}{l_{av} \times N} \tag{5.7.2.2}$$

a : Cross sectional area of wire

Taking into account the air spacing between the wires (assuming a compact winding), the actual cross sectional area would be :

$$a' = 0\cdot 92 \times a \tag{5.7.2.3}$$

Using equations (5.7.2.1), (5.7.2.2) and (5.7.2.3), the D.C. resistance of each coil is obtained.

For calculation of self-inductance of the coils, we consider the ideal solenoid through the centre of which a core of infinite length is inserted. The self-inductance of such a coil per unit length is then given by :

$$\mu_0\pi\, N^2\cdot\left\{R^2 + \left(\mu_r - 1\right)\cdot R_c^2\right\} \tag{5.7.2.4}$$

Where :

R : mean radius of solenoid

R_c : radius of wire

μ_o: permeability of air

μ_r : relative permeability of the core material

N : number of turns per unit length

In our case, the length of the armature core is negligible relative to the air gap. Therefore, the electrical self inductance of each coil, to a first approximation, is equivalent to the inductance of short solenoids. The flux density B, along the axis of symmetry of such a coil, is given by :

$$B = \frac{\mu\cdot N\cdot i}{l\sqrt{\left(\frac{4\cdot R^2}{l^2} + 1\right)}} \tag{5.7.2.5}$$

In equation (5.7.2.5), we have:

R : mean radius of solenoid
l : length
μ : magnetic permeability
Ni : Ampere-turns through solenoid

The total flux linkage λ through the coil is given by :

$$\lambda = N \cdot B \cdot A \tag{5.7.2.6}$$

Where :

A : cross sectional area of magnetic path

From the definition of self-inductance, we have :

$$L = \frac{\lambda}{i}$$

$$L = \frac{\mu \cdot N^2 \cdot A}{l \sqrt{\left(\frac{4 \cdot R^2}{l^2} + 1 \right)}} \tag{5.7.2.7}$$

Equations (5.7.2.1) and (5.7.2.7) give expressions for the nominal D.C. resistance and inductance of the coils, neglecting the eddy currents.

The presence of eddy currents results in energy loss and is manifested as a change in the D.C. resistance and inductance of the transformer coils. The consideration of eddy currents aims to establish the amount by which the magnetic field strength in the ferrite core changes and what these currents depend on. It has been established that eddy currents depend on the geometry of the material, on its resistivity and the frequency of the alternating flux. Bozorth (1964) has stated these relationships; and they must be considered at high frequencies.

5.7.3 Design objectives

Given a nominal operating input voltage excitation, the objectives of the optimization process are to select design parameters, which, under the particular configuration of the LVDT, would minimize non-linearity error and would maximize the sensitivity. We know from equation (5.7.1.18) that the differential output voltage, in terms of the incremental displacement, is given by :

$$e = k_1 \cdot x\left(1 - \frac{x^2}{k_2}\right)$$

The optimized LVDT must have a minimum non-linearity error at the user specified maximum displacement.

The non-linearity error of the transformer is defined as the difference between the expected and actual value at the output for a given user required maximum displacement, as defined below :

$$ER1 = Nonlinearity\ measure = \left\{k_1 \cdot x_m\left(1 - \frac{x_m^2}{k_2}\right)\right\} \qquad (5.7.3.1)$$

where x_m is the maximum displacement. The above expression must satisfy a user specified minimal non-linearity error. Also, the sensitivity of the device must be as close as possible to a user specified sensitivity. This is given by :

$$ER2 = sensitivity = k_1\left(1 - \frac{x_m^2}{k_2}\right) = \frac{e}{x_m} \qquad (5.7.3.2)$$

5.7.4 Design constraints

The LVDT design parameters are restricted to certain user specified maximum values and geometrical constraints. The design parameters are given below :

d : spacing between coils
b : length of primary coil
m : length of secondary coil
L_a : armature length
r_i : inner radius of coils
r_o : outer radius of coils
f : excitation frequency
N_v : number of turns per unit volume

The geometrical constraints are given below:

1- The length of the LVDT must not exceed a user specified maximum size. I.e. :

$$C_1 : b + 2m + 2d - (\max length) < 0 \qquad (5.7.4.1)$$

2- The length of the armature must not be less than the length of the primary coil. I.e. :

$$C_2 : L_a > b \Rightarrow b - L_a < 0 \qquad (5.7.4.2)$$

3- The armature must not emerge from the secondary coils of the LVDT. I.e. :

$$C_3 : L_a + 2 \cdot x_m - \{b + 2(m + d)\} < 0 \qquad (5.7.4.3)$$

Where x_m is the maximum displacement.

4- The distance between inner and outer radii of the coils must not be less than a minimal feasible length. I.e. :

$$C_4 : R_i - R_o + (minimal\ feasible\ length) \leq 0 \qquad (5.7.4.4)$$

In our case, we choose a minimal feasible distance of 1mm.

5.7.5 The search space

As discussed before, genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite length string. The design parameters were given in section 5.7.4. As an example, we consider the following parameter ranges for our search space :

$$\begin{aligned}
&1 \cdot 0 \times 10^{-3} m \leq b \leq 3 \cdot 0 \times 10^{-2} m \\
&1 \cdot 0 \times 10^{-3} m \leq m \leq 3 \cdot 0 \times 10^{-2} m \\
&1 \cdot 0 \times 10^{-3} m \leq L_a \leq 2 \cdot 7 \times 10^{-2} m \\
&1 \cdot 0 \times 10^{-3} m \leq r_i \leq 6 \cdot 0 \times 10^{-3} m \\
&2 \cdot 0 \times 10^{-3} m \leq r_o \leq 1 \cdot 0 \times 10^{-2} m \\
&10\, Hz \leq f \leq 400\, Hz
\end{aligned} \qquad (5.7.5.1)$$

These ranges have been selected to contain existing industrial LVDT parameters for measurement of a maximum input displacement of 1·27mm. Different parameter ranges and maximal displacements can be simply specified by using the implemented software. The number of turns per unit volume (N_v) and the spacing between coils were kept fixed to reduce the dimensionality of the search space. This will not affect the optimal design results considerably. The actual number of turns of the primary and secondary coils are a function of the overall dimensions of the LVDT and the spacing between coils is considered negligible as compared to the primary and secondary coil lengths. The fixed design parameters of the LVDT are given below :

supply voltage = 2·7 volts
max displacement = 1·27 mm
resistivity of wire = $1 \cdot 8 \times 10^{-8}$ Ωm
number of turns/unit vol. = $162 \cdot 929 \times 10^{7}$ turns/m^3

As before, for the specified search space, the independent design variables are discretized by mapping each variable in its specified range into a 10 bit binary unsigned integer. To form a complete representation of the problem, the 6 parameter coding were concatenated to form a 6 × 10 = 60 bit string representing a particular LVDT design.

5.7.6 Formulation of the utility function

The objective of our design optimization is to find a set of optimal design parameters satisfying a user required minimal non-linearity error and maximal sensitivity. The design objectives and constraints were given in sections 5.7.3 and 5.7.4.

In order to transform the design objectives to a maximization problem, the following fitness mapping is used:

$$U(X) = F_1(X) + F_2(X) \qquad (5.7.6.1)$$

Where:

$$F_1(X) = w_1 \left\{ \frac{F_n(X)}{R_n} \right\}^{r_1}$$

$$F_2(X) = w_2 \left\{ \frac{F_s(X)}{R_s} \right\}^{r_2}$$

and:

X : the design parameter vector
w_1, w_2 : the weighting factors
R_n : desired minimum % non linearity
$F_n(X)$: actual % non linearity for a design candidate
R_s : Desired sensitivity
$F_s(X)$: actual sensitivity for a design candidate

$$r_1 = \begin{cases} +1 & when \quad F_n(X) \leq R_n \\ -1 & when \quad F_n(X) \rangle R_n \end{cases}$$

and

$$r_2 = \begin{cases} +1 & when \quad F_s(X) \leq R_s \\ -1 & when \quad F_s(X) \rangle R_s \end{cases}$$

The utility function, as represented by equation (5.7.6.1), is based on normalized performance criteria. The maximum of each normalized component occurs when the performance of an actual design candidate approaches to the user specified performance criteria. When a particular normalized component is greater than 1, its reciprocal is substituted in the utility function. This ensures that the normalized components are symmetrical functions about the desired parameter values.

The weighting coefficients have been included for scaling purposes. The magnitude of each weighting factor depends on the degree of emphasis required for each performance criterion and can be determined empirically. In our case, both performance criteria, i.e., the required minimum %non-linearity error and maximum sensitivity, have equal importance. Therefore, a value of 0·5 is chosen for each weighting factor, i.e.:

$$w_1 = w_2 = 0 \cdot 5 \qquad (5.7.6.2)$$

A constraint violation is squared and subtracted from the actual utility function after multiplication by an appropriate penalty coefficient.

Therefore, our constraint formulation becomes:

$$U'(X)=U(X)-\sum_{i=1}^{4}\mu_i\cdot\Phi(C_i(X)) \qquad (5.7.6.3)$$

Where :

μ_i: penalty coefficients

$C_i(X)$: the design constraints

and

$$\Phi(C_i(X))=\begin{cases}\{C_i(X)\}^2 & if\ C_i(X)>0\\ 0 & if\ C_i(X)\le 0\end{cases}$$

Penalty coefficients are sized for constraints C_1, C_3, and C_4 so that a maximum violation of the order of 1mm yields a penalty that is equal to 20% of our nominal fitness value. This gives a μ value of 100,000 for each constraint, i.e. :

$$\mu_{i=1,3,4}=100{,}000 \qquad (5.7.6.4)$$

These coefficients were selected to be as relaxed as possible but sufficient to avoid non-feasible regions. In practice, penalties are occasionally critisized because of the steep ridges they impose on otherwise smooth problems. These ridges can cause difficulty among search techniques which depend upon a particular shape of local search surface. As the performance of genetic algorithm does not depend on the continuity or derivative existence, their performance is much less affected by the shape of the local search surface.

For constraint C_2 we had :

$$C_2: L_a > b \Rightarrow b - L_a < 0$$

This constraint must be satisfied always, i.e., the length of the armature must be greater than the length of the primary coil. If this constraint is not satisfied, there will be no possibility for variation of mutual inductance couplings by the movement of the armature. A candidate design not satisfying this constraint can not have a meaningful fitness value. We, therefore, assign zero fitness to such solutions.

Looking at our search space, an initial randomly generated population has a high probability of generating LVDT designs not satisfying the above constraint. This is due to the fact that the implemented pseudo-random number generator (based on subtractive method, Knuth (1981)) produces uniform deviates lying within 0 and 1 with equal probability and the probability of generating a 1 or a 0 bit for each position of a chromosome has been set to 0·5 for all simulations. In fact the expected number of zero fitness candidate designs, in an initial randomly generated population, is approximately 0·5*pop-size. Also, constraint C_4 specifies a minimal feasible length for the distance between the inner and outer radii of the coils. The penalty coefficient for this constraint has been specified but there is an exceptional case that must be accounted for. This is when a candidate design is produced either randomly during the initial generation or as a result of mutation and/or cross-over operations for which we might get:

$$R_o < R_i$$

This type of solution is physically non-realizable and consequently the fitness of such designs are set to zero during our simulations.

From above discussion, we note that the search space is highly contaminated with zero fitness individuals and in an initial random population more than half of the candidate designs might have a zero fitness value. One way to rectify this situation is to implement special purpose routines to avoid non-realizable designs. For example, some heuristics can be used, in conjunction with the crossover and mutation operations, to avoid the production of non-feasible designs. Also, the randomly generated initial population can be searched and non-realisable solutions replaced by randomly generated realizable designs. These remedies are highly problem specific and artificial. Our main premise for using genetic algorithms have been to avoid problem dependent techniques and exploit the robustness of genetic algorithms. Also, these suggestions are highly expensive in terms of computer time. The simplest approach is to increase the population size. A larger population is more likely to contain representative schemata from a larger realizable designs. Therefore, the population size in the following simulations has been doubled and set to 200.

5.7.7 Optimization results

As stated in section 5.7.6, our search space is highly contaminated with zero fitness individuals. It is important to note that, non realisable solutions might contain problem related schemata which when combined might lead to realizable and even optimal designs. Therefore, zero-fitness designs must also have a chance for reproduction. We recall that, by using our ranking selection method in conjunction with the elitist approach, only the bottom 5% of the population receives no offspring and chromosomes with equal fitnesses are ranked according to the processing sequence. Therefore, even in a search space which is highly contaminated with zero fitness individuals, a ranking selection policy will perform a more informed search.

In the following simulations, we use our Ranking selection strategy in conjunction with the elitist approach. As before, we use Grefenstette's proposed cross-over and mutation rates. I.e.:

$$\begin{aligned} P_c &= 1 \cdot 0 \\ P_m &= 0 \cdot 01 \end{aligned} \qquad (5.7.7.1)$$

Initially, a user required sensitivity of 400 v/m and a minimal nonlinearity of 0·8% is specified for the optimization process.

The performance curve, up to and including generation 100, is shown in Figure- 5.21. The optimal design parameters, obtained, are tabulated in table 5.8 below:

Independent design Parameters						Dependent Variables						
B(mm)	M(mm)	L_a (mm)	R_i (mm)	R_o (mm)	F(Hz)	I_p (A)	N_p	N_s	IMP(Ω)	%NLT	sensitivity	Fitness
1· 0	14· 9	27	1· 98	7· 83	63	0· 2	294	4369	8·92	0·88%	399·97vm^{-1}	0·9999

Table 5.8 Optimal results for %NLT = 0·8, sensitivity = 400 vm^{-1}

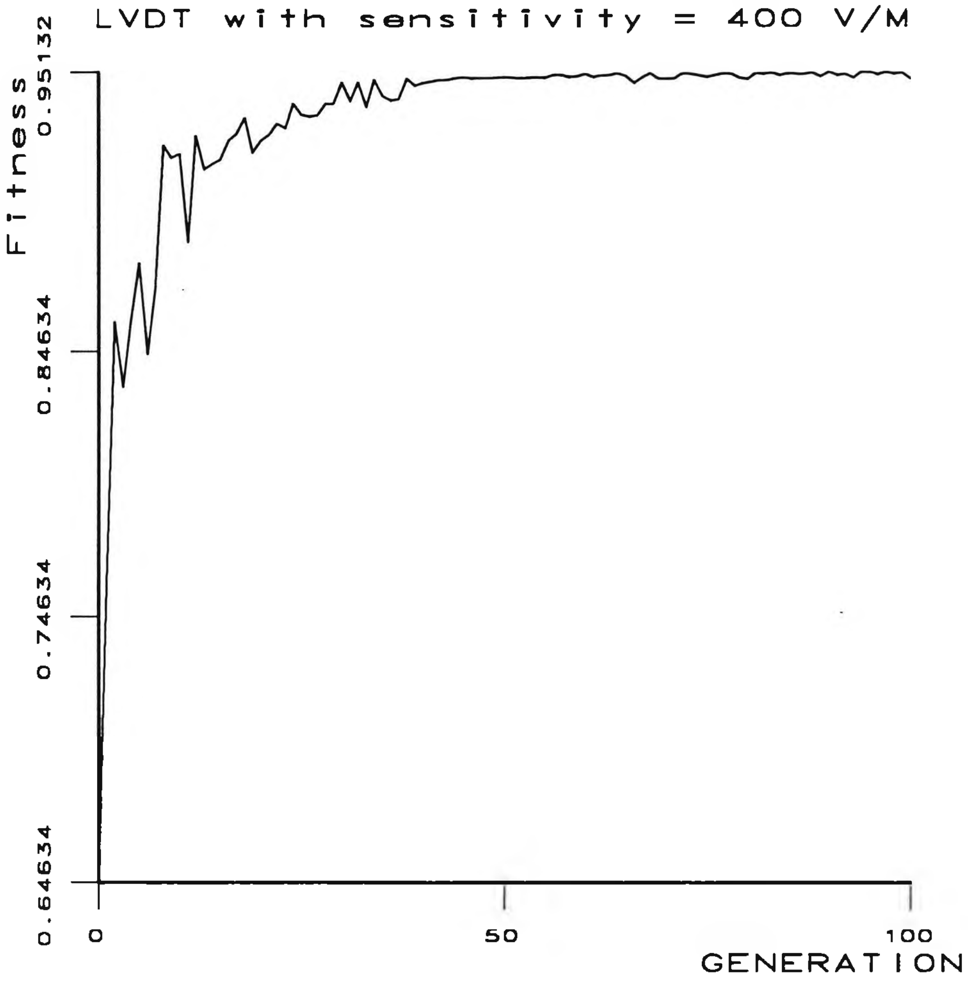

LVDT with sensitivity = 400 V/M
Fitness
0.95132
0.84634
0.74634
0.64634
0
50
100
GENERATION

Figure 5.21

As seen from table 5.8, the candidate design, given by our Rank-El, is optimal and has satisfied both performance criteria; although the %non-linearity error of this design is 0·08% higher than specified. In order to gain an understanding of the reason behind the higher %non-linearity arrived at, we perform two simulations in which we increase the desired sensitivity to 700 vm^{-1}· In the first simulation, the desired %non-linearity is set to 1% and in the second it is set to 0·8%. The performance curves, up to and including generation 100, are shown in Figure 5.22 and Figure 5.23. The optimal results, for each case, are tabulated in table 5.9 below:

	Independent design Parameters						Dependent Variables						
	B(mm)	M(mm)	L_a(mm)	R_i(mm)	R_o(mm)	F(Hz)	I_p (A)	N_p	N_s	IMP(Ω)	%NLT	sensitivity	Fitness
Case-1	11·7	17·5	26·7	2·2	9·9	399	0· 004	5614	8406	727	1%	700vm^{-1}	0·9999
Case-2	1	20	27	2·0	9·6	116	0·14	453	9222	19·22	0·88%	700vm^{-1}	0·9513

Table 5.9 Optimal results for two design cases

From table 5.9, the design, given for 1% required minimal non-linearity error, is a perfectly optimal solution but for the case of 0·8% minimal specified non linearity, there is still a 0·08% discrepancy. Further examination of the candidate design for this case reveals that, the optimal values of primary length and armature length have been suggested to be at their minimum and maximum constraint boundaries. This is not the case for more relaxed minimal non-linearity of 1%. Looking back at equation (5.7.1.18), we note that % non- linearity will be minimal for maximum k_2 values. From equations (5.7.1.18) we have :

$$k_2=0{\cdot}5(b+2d)(L_a-b-2d) + 0{\cdot}25(L_a-b-\ 2d)^2 \qquad (5.7.7.2)$$

A reduction in values of b and d, in the above equation, will increase k_2 and an increase in the armature length will increase the second term by the square. This implies that the primary coil for this LVDT arrangement, at least, must be as short as possible. In other words, the distance separating the two secondary coils must be minimal. Furthermore, it can be noted, from equations (5.7.1.18), that increases in the armature length has an increasing effect on the overall sensitivity of the LVDT. It is observed that, the armature length is the most sensitive parameter. This observation has been already confirmed by Rahman (1979), using sensitivity analysis.

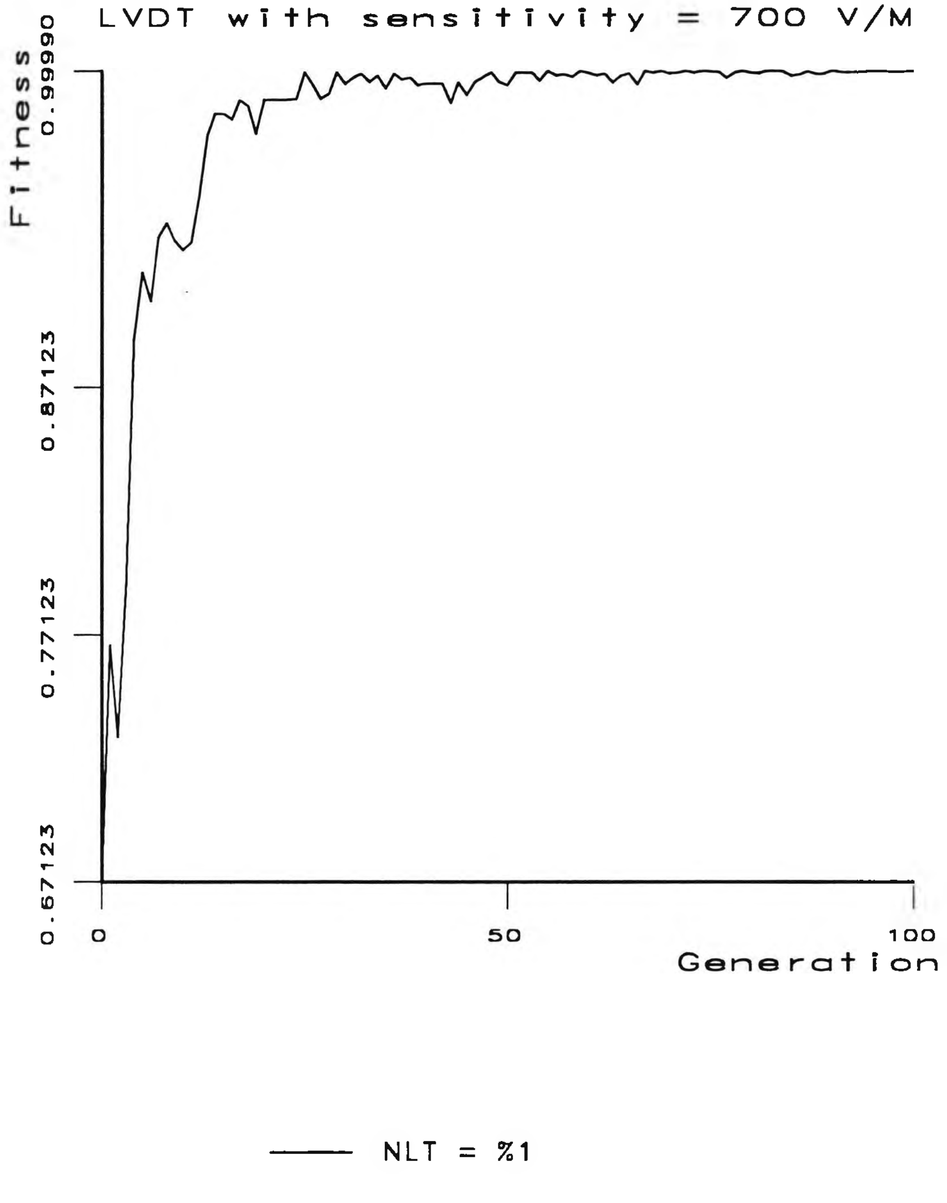
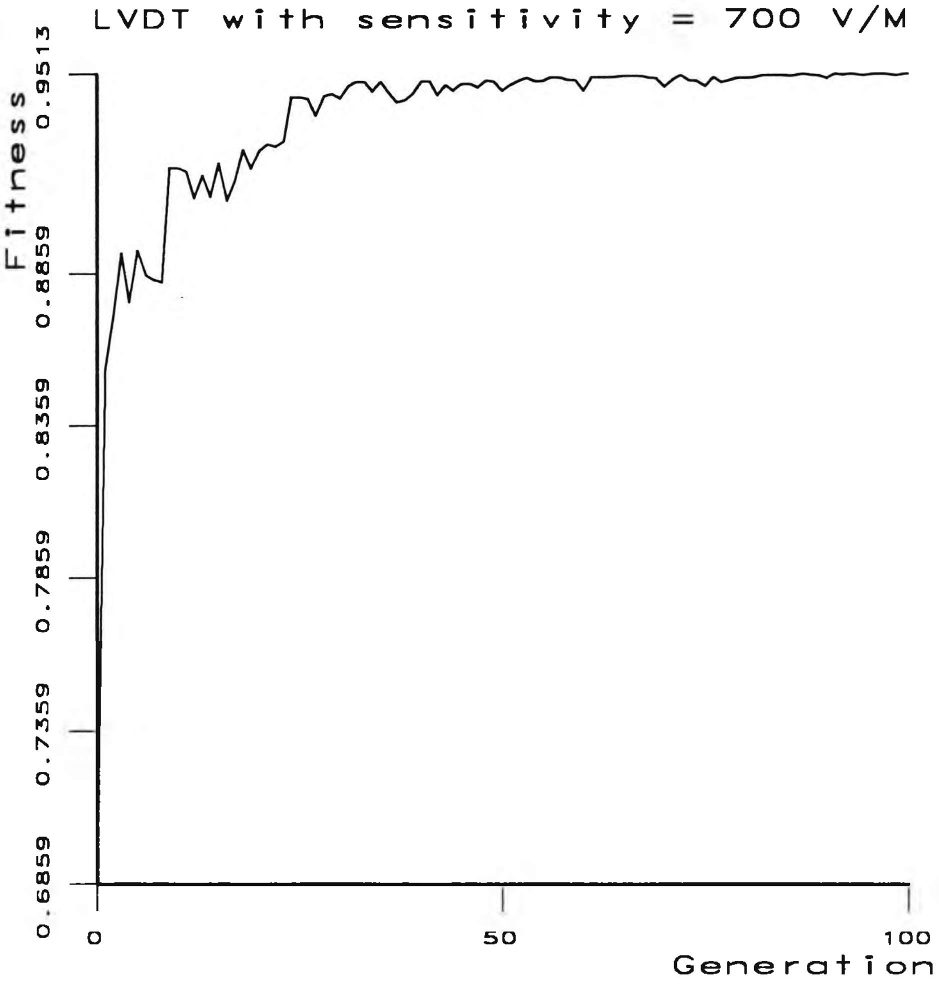

LVDT with sensitivity = 700 V/M
Fitness
0.99990
0.87123
0.77123
0.67123
0
50
100
Generation
NLT = %1

Figure 5.22


LVDT with sensitivity = 700 V/M
Fitness
0.9513
0.8859
0.8359
0.7859
0.7359
0.6859
0
50
100
Generation

NLT = %0.88

Figure 5.23

In fact, the only way that we can hope to achieve lower %non-linearity would be to allow a higher permissible length for the armature and/or lower permissible values for the length of the primary coil. Hence the design solutions, suggested by our Rank-El genetic algorithm, are optimal across the specified search space.

In order to identify a lower bound on the maximum possible sensitivity attainable across our search space, we will tackle the following design problems :

1- Required sensitivity = 1000 vm^{-1}
2- Required sensitivity = 2000 vm^{-1}
3- Required sensitivity = 2800 vm^{-1}
4- Required sensitivity = 3500 vm^{-1}

The required minimal %nonlinearity for all of the above problems is set to the lowest attainable.

Figure 5.24, shows the performance curve, up to and including generation 200, for design problem-2. Table 5.10 below gives the optimal results obtained for design problem-1 and design problem-2.

Independent design Parameters							Dependent Variables						
	B(mm)	M(mm)	L_a (mm)	R_i (mm)	R_o (mm)	F(Hz)	I_p (A)	N_p	N_s	IMP(Ω)	%NLT	sensitivity	Fitness
Problem-1	1	21	27	3·9	6·3	140	0·5	126	2725	4·22	0·88%	1000vm^{-1}	0·951
Problem-2	3	20	27	3·0	9·5	377	0·04	1234	8238	75·0	0·88%	2000vm^{-1}	0·991

Table 5.10 Optimal results for Design Problems 1 and 2

As observed from the above table, both design candidates are optimal with respect to the design performance criteria.

For design problem-3, we run two independent simulations, using two different random seeds, to investigate the possibility of alternative optimal designs. Table 5.11, below, shows the results of these two independent runs up to and including generation 200.

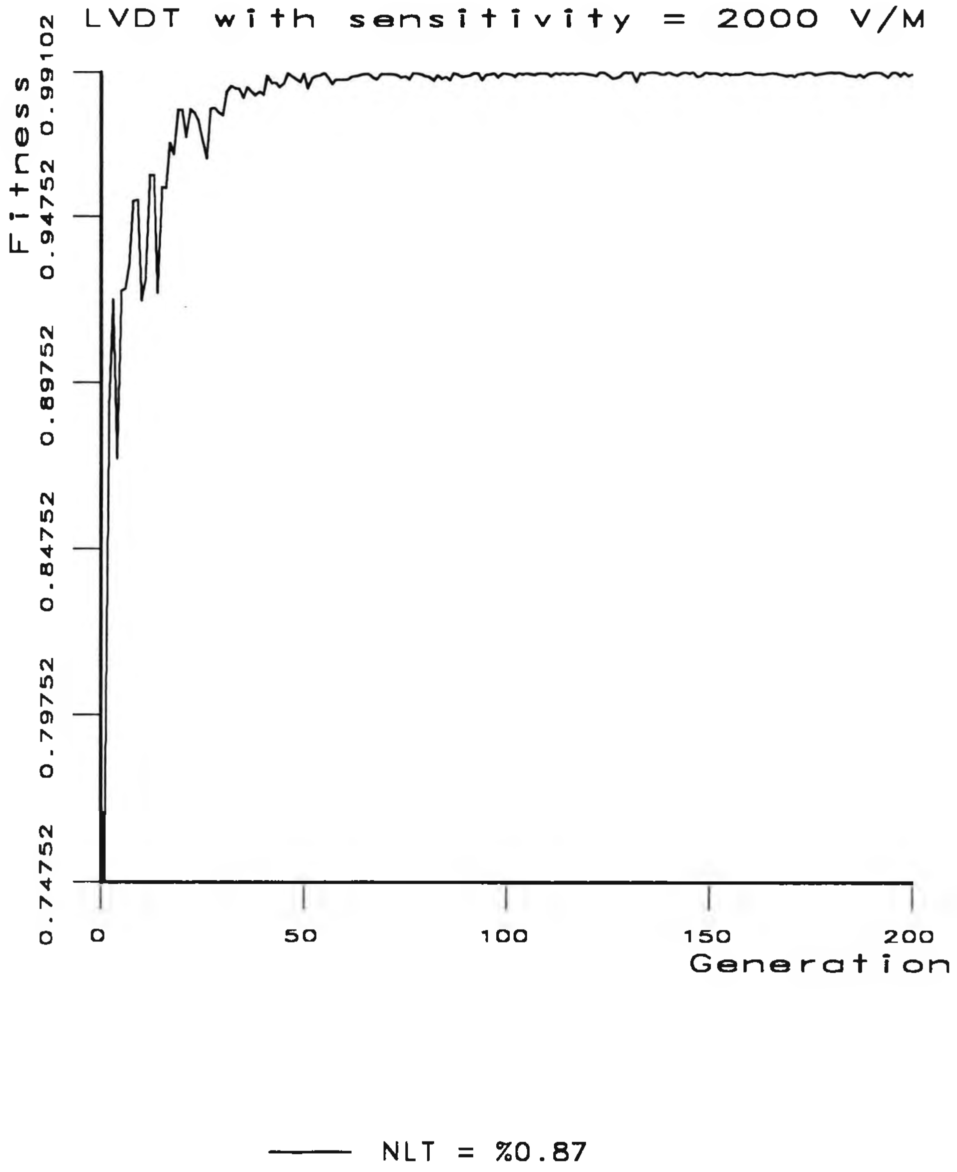

LVDT with sensitivity = 2000 V/M
Fitness
0.99102
0.94752
0.89752
0.84752
0.79752
0.74752
0
50
100
150
200
Generation
NLT = %0.87

Figure 5.24

	Independent design Parameters						Dependent Variables						
	B(mm)	M(mm)	L_a (mm)	R_i (mm)	R_o (mm)	F(Hz)	I_p (A)	N_p	N_s	IMP(Ω)	%NLT	sensitivity	Fitness
Case-1	1·0	15	27	3·6	6·0	395	0·7	121	1818	3·6	0·88%	$2800 vm^{-1}$	0·951
Case-2	1·0	25	27	3·1	4·9	392	1·6	75	1863	1·7	0·88%	$2800 vm^{-1}$	0·991

Table 5.11 Two alternative designs for Design Problem-3

Both design solutions, suggested by our Rank-EI selection strategy, are optimal with respect to our performance criteria. We note that, apart from the B and L_a values, these designs do not agree at other design parameters. We conclude that our NLP optimization problem, at least for more relaxed design criteria, is characterized as having alternative optimal designs. This issue will be further investigated in the next section.

Finally, we look at our last design problem. Figure 5.25, shows the performance curve up to and including generation 150, and table 5.12 below gives the result of this simulation.

Independent design Parameters						Dependent Variables						
B(mm)	M(mm)	L_a (mm)	R_i (mm)	R_o (mm)	F(Hz)	I_p (A)	N_p	N_s	IMP(Ω)	%NLT	sensitivity	Fitness
1·0	18·6	27	5·9	6·9	400	0·7	66	1235	3·5	0·88%	$2904 vm^{-1}$	0·86

Table 5.12 Optimal result for Design Problem-4

A sensitivity requirement of 3500 vm^{-1} has proved to be unattainable. The optimal result satisfies a maximum sensitivity of 2904 vm^{-1}· We must note that, the suggested excitation currents in tables 5.11 and 5.12 are very large and might exceed practical current rating limits of the primary coil wires. This problem can be simply resolved by incorporating an additional constraint concerning the current rating limits of coil wires. However, at this stage, our concern has been to confirm the efficiency of the ranking selection strategy and extract some useful design heuristics (concerning the influence of design parameters on the overall performance of LVDTs) using our rather less complicated formulation. These high primary excitation currents indicate that, to achieve the highest possible sensitivities, the primary coil impedance must be minimal. Therefore, within our restricted search space, by incorporating more severe constraints on the current rating of the coil wires, we do not

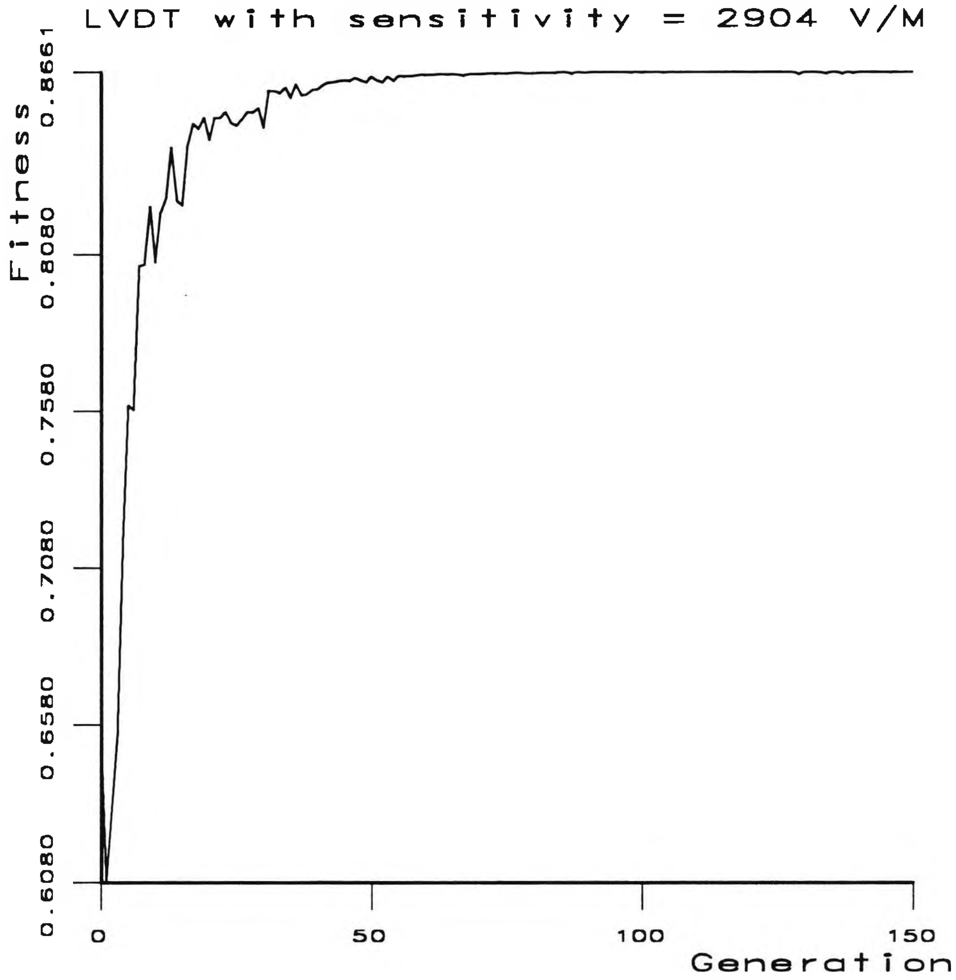

LVDT with sensitivity = 2904 V/M
Fitness
0.8661
0.8080
0.7580
0.7080
0.6580
0.6080
0
50
100
150
Generation

Figure 5.25

expect to find optimal designs for these high sensitivity requirements. By looking at our results so far, the following design heuristics emerges:

The primary coil, for this LVDT arrangement, must be as short as possible resulting in the distance separating the secondary coils to be minimal. The excitation frequency is suggested to be at its maximum constraint boundary. This is expected as, according to our mathematical model, sensitivity is linearly increased with respect to the excitation frequency.

The optimal inner radius is maximized. This means that the armature diameter must be maximized for high sensitivity requirements. The distance between the inner and outer radii of the coils is minimized within the specified constraints. This results in a substantial decrease of the primary coil impedance of the LVDT and increase in its excitation current. Again increases in primary excitation current will linearly increase the sensitivity of the LVDT.

Furthermore, the secondary coil length has been increased to accommodate the required increase of the armature length for higher sensitivities and lower non-linearities.

From above discussion it follows that, to attain higher sensitivities and lower %non-linearities, we must allow higher excitation frequencies and higher armature length within our search space.

It is important to consider that, in our mathematical model, the leakage from the ends of the armature and also any modifications of the flux distribution due to the proximity of the shielding case has been neglected. Furthermore, the assumption of linear dependence of sensitivity to the input excitation frequency is only valid up to 500 Hz. At higher frequencies, the relation is not linear mainly due to the effect of eddy currents which have been neglected in our mathematical model.

These facts make it necessary to use more sophisticated mathematical models. In the past, highly sophisticated distributed parameter mathematical models, based on the direct solution of flux distribution within the LVDT, has been suggested and used for the analysis and design of specific LVDT configurations (Rahman, 1979). These models are based on the solution of partial differential equations derived from maxwell's equations and are expected to be more accurate, because they take into account all the geometric features. The differential equations are mostly solved by using finite difference and/or finite element methods.

The use of a more sophisticated mathematical model by no means implies the need for modifications in our genetic algorithm utility function or any of its parameters. It only means that we are using a more accurate environment interfaced to our genetic algorithm. That is to say, our genetic algorithm will have a more realistic image of its design environment.

5.8 Multimodal function optimization using genetic algorithms

In sections 5.5, 5.6 and 5.7, using three operator genetic algorithms, we investigated the design optimization of corrugated diaphragms and LVDTs. Although the three operator genetic algorithms were shown to be suitable for these problems, in practice, we might like to investigate the existence of alternative designs having similar performance criteria and/or investigate the multimodality of the optimization problem. In section 5.7.7, using two different random seeds (please refer to table 5.11), we concluded that the LVDT optimization problem, at least for more relaxed user specified design criteria, has a number of alternative design solutions. In this section, we further elaborate these issues by exploring a number of suggested techniques for the purpose of multimodal function optimization.

In optimization of multimodal functions, a three operator genetic algorithm might not be able to maintain controlled competition among the competing schemata corresponding to different peaks, and the stochastic error associated with the genetic operators causes the population to converge to one alternative or another. As we recall from section 5.2.1, this problem with finite populations is known as genetic drift (DeJong, 1975). Moreover, in dealing with multimodal functions with peaks of unequal value, a simple three operator genetic algorithm converges to the best peak; whereas, in addition to wanting to know the best solution, one may be interested in knowing the location of other local optima. To overcome these limitations, some modifications in our genetic algorithm are necessary.

In nature, a species is a collection of organisms with similar features. The subdivision of environment on the basis of an organism's role reduces inter-species competition for environmental resources, and this reduction in competition helps stable subpopulations to form around different niches in the environment. A number of methods are suggested to introduce this concept in genetic algorithms.

In DeJong's (1975) crowding, separate niches are created by replacing existing strings according to their similarity with other strings in an overlaping population.

Two parameters, generation gap (G) and crowding factor (CF), are defined for this purpose. Generation gap G dictates the use of an overlaping population model in which only a proportion G of the population is permitted to reproduce in each generation. To induce niche in the population, the following approach is used. When selecting an individual to die, CF individuals are picked at random from the population, and the one which is most similar to the new individual is chosen to be replaced, where similarity is defined in terms of the number of matching alleles. The new individual (chosen by usual selection methods) then replaces this chosen individual in the population. DeJong used this scheme successfully with crowding factor CF=2 and 3 and with generation gap G=0·1 on a number of multimodal optimization applications.

Goldberg and Richardson (1987) used Holland's (1975) sharing concept by dividing the population in different subpopulations according to the similarity of the individuals in two possible solution spaces : The decoded parameter space and the gene space. They defined a sharing parameter σ_{share} to control the extent of sharing, and they defined a power-law sharing function *sh(d)* as a function of the distance-metric *(d)* between two individuals as follows:

$$sh(d) = \begin{cases} 1 - \left(\dfrac{d}{\sigma_{share}} \right)^{\alpha} & if \quad d < \sigma_{share} \\ 0 & otherwise \end{cases} \qquad (5.8.1)$$

To implement the idea of sharing, an individual's payoff is degraded due to the presence of other individuals in its neighbourhood. When the proximity of the individual is defined in the decoded parameter space, it is called phenotypic sharing.

The distance metric (d_{ij}), considered in phenotypic sharing, is the distance between strings in the decoded parameter space. For a single parameter function, this may be calculated as the absolute difference of the decoded parameter values of the strings. In general, for a p-parameter function, the distance metric d_{ij} may be calculated using any suitable distance-norm in the p-dimensional space. For simplicity, the euclidian distance in p-dimensional space can be used. Therefore, for the individuals

$X_i = \left[x_{1,i}, x_{2,i}, \ldots x_{p,i}\right]$ and $X_j = \left[x_{1,j}, x_{2,j}, \ldots x_{p,j}\right]$the metric d_{ij} may be calculated as:

$$d_{ij} = \sqrt{\sum_{k=1}^{p}\left(x_{k,i} - x_{k,j}\right)^2} \tag{5.8.2}$$

Where the $x_{1,i}$, $x_{2,i}$, . . . $x_{p,i}$ are the decoded parameters. To estimate the parameter σ_{share} , imagine that each niche is enclosed in a p-dimensional hypersphere of radius σ_{share} such that each sphere encloses *1/q* of the volume of the space, where q is the number of niches in the solution space. The radius of a hypersphere containing the entire space is calculated as:

$$r = \frac{1}{2}\sqrt{\sum_{k=1}^{p}\left(x_{k,\max} - x_{k,\min}\right)^2} \tag{5.8.3}$$

The volume of the hypersphere is calculated as $V = Cr^p$ with C a constant. Dividing this volume in q parts and recognizing that the hypervolume has the same form regardless of size, σ_{share} may be calculated as follows :

$$\begin{aligned} C\sigma_{share}^{p} &= \frac{1}{q}C \cdot r^{p} \\ \sigma_{share} &= \frac{r}{\sqrt[p]{q}} \\ &= \frac{\sqrt{\sum_{k=1}^{p}\left(x_{k,\max} - x_{k,\min}\right)^2}}{2\sqrt[p]{q}} \end{aligned} \tag{5.8.4}$$

Unfortunately, DeJong's on-line and off-line performance measures are not directly suitable for judging the distribution pattern of the trials over the peaks in the case of multimodal function optimization. Moreover, a high on-line or off-line performance measure is not meaningful when the function has unequal peaks. Therefore, simple on-line and off-line performance metrics is inadequate for judging the performance of genetic algorithms in the case of multimiodal functions. To characterize the distribution of trials over the peaks, Deb & Goldberg (1989b) has suggested a chi-square-like criterion, where the actual distribution is compared to an ideal distribution. This ideal distribution is calculated based on Holland's (1975) sharing concept. Deb & Goldberg (1989b), using this performance metric on a number of test functions, has shown that a sharing scheme is higly effective in distributing trials at all the peaks. However, at this point, we are only interested in the best of run results at each design peak. Therefore, in the following experiments, after each iteration, the best candidate design in each design partiotion is registered and kept until a better candidate is generated during future generations.

In order to investigate the sharing approach, we will look back at the first design example (section 5.5.3) which has been given by Andreeva. In that example, we found the optimal profile parameters for the following user specified required characteristics and active radius:

$$p = 0{\cdot}106 \cdot \omega_0 + 0{\cdot}1055 \cdot \omega_0^3$$
$$E = 1{\cdot}35\times10^6 \text{ (kg/cm}^2\text{)}$$
$$R = 24 \text{ mm}$$

In fact, the required characteristics has a 29% non-linearity error and in this respect is a sub-optimal characteristics.

If we introduce the active radius as an independent design variable, we will expect an infinite number of solutions satisfying the above characteristics . Looking back at equation (5.5.2.1), we have:

$$for \ \frac{H}{h}=1 \ \Rightarrow \ \frac{a_p}{b_p^3}\approx 1{\cdot}5$$
$$for \ \frac{H}{h}=17 \ \Rightarrow \ \frac{a_p}{b_p^3}\approx 4{\cdot}0\times10^7$$

But:

$$R=\sqrt[8]{\frac{\frac{A}{B^3}\cdot E^2}{\frac{a}{b^3}}}$$

Hence for $1 < H/h < 17$, we would expect to find R values that satisfy the required characteristic within the following range:

$$6{\cdot}71mm < R < 57mm$$

Suppose that we would like to find 17 alternative profile geometries, each coresponding to a unit increment of H/h within its possible range. I.e., we would like to obtain 17 alternative designs. Using equation 5.8.3, σ_{share} is calculated as follows:

$$r = \frac{1}{2}\sqrt[2]{(57-6\cdot71)^2+(17-1)^2}$$

$$\sigma_{share} = \frac{r}{\sqrt[p]{q}} = \frac{52\cdot78}{2\times\sqrt[2]{17}} \approx 6\cdot40$$

In order to incorporate the sharing concept, as suggested by equation (5.8.1), few computer routines were implemented in conjunction with the three operator genetic algorithm.

A triangular sharing function was used in this case. I.e., the α value in equation (5.8.1) was set to 1·0. We will use a phenotypic sharing. I.e., the distance metric is defined over the decoded parameters. Also, we choose the euclidian distance in p-dimensional space as our distance metric.

Once we have selected a metric and a sharing function, it is a simple matter to determine the shared fitness of a string. The shared fitness of a string (f') is its potential fitness divided by its niche count m'_i:

$$f'_i = \frac{f_i}{m'_i} \qquad (5.8.5)$$

The niche count m'_i , for a particular string i, is taken as the sum of all shared function values taken over the entire population:

$$m'_i = \sum_{j=1}^{N} sh(d_{ij}) = \sum_{j=1}^{N} sh(d(x_i, x_j))$$

Note that the above sum includes the string itself. Thus, if a string is all by itself in its own niche ($m'_i = 1$) it will receive its undegraded full fitness value. Otherwise, the sharing function degrades fitness according to the degree of closeness of neighboring points.

We now evaluate the use of sharing functions through a computational experiment. Genetic algorithm parameters used are given below:

$$\begin{aligned} p_m &= 0{\cdot}001 \\ p_c &= 0{\cdot}85 \\ \textit{Pop-size} &= 170 \\ \textit{Max number of generations} &= 100 \end{aligned}$$

The stochastic remainder selection has been used in this experiment. Mutation rate has been set to a low value to test the maintenance of diversity by using our sharing function without introducing arbitrary diversity through mutation. The result of simulation at generation 100 is shown in table 3.13 below. For computer generated results refer to APPENDIX-III.

	Independent Var.s		**Dependent Design Var.s**					
Partition	H/h	R(mm)	H(mm)	A	B	TER	f	f'
1.0	1.06	56.16	0.49	0.1059	0.1055	0.000005	0.999995	0.09779
2.0	2.24	37.45	0.51	0.1041	0.1055	0.001871	0.998133	0.09117
3.0	3.08	30.08	0.48	0.1093	0.1055	0.003338	0.996673	0.10182
4.0	3.74	23.86	0.32	0.0563	0.1055	0.049700	0.952653	0.08800
5.0	5.45	18.97	0.35	0.1077	0.1055	0.001773	0.998230	0.09193
6.0	6.35	16.66	0.32	0.1107	0.1055	0.004692	0.995330	0.07867
7.0	7.11	14.99	0.29	0.1062	0.1055	0.000254	0.999746	0.11051
8.0	7.81	13.43	0.25	0.0866	0.1055	0.019372	0.980996	0.08677
9.0	9.09	11.99	0.24	0.1062	0.1055	0.000170	0.999830	0.11144
10.0	9.56	11.53	0.24	0.1120	0.1055	0.006016	0.994020	0.11333
11.0	10.98	9.80	0.19	0.0854	0.1055	0.020570	0.979845	0.06686
12.0	11.99	7.97	0.11	0.0315	0.1055	0.074470	0.930691	0.07738
13.0	13.35	8.41	0.18	0.1091	0.1055	0.003134	0.996876	0.08159
14.0	13.99	7.92	0.16	0.0959	0.1055	0.010058	0.990042	0.08591
15.0	15.47	7.28	0.16	0.1057	0.1055	0.000243	0.999757	0.10521
16.0	15.59	7.28	0.16	0.1124	0.1055	0.006455	0.993587	0.09718
17.0	16.91	7.13	0.19	0.1767	0.1055	0.070745	0.933929	0.12613

Table 5.13 Results for the multimodal design optimization of a Diaphragm

The theoretical expected number of near optimal points, for each *H/h* value in this experiment, is calculated as:

$$\mu_i = \frac{f_i}{\sum f_j} \times N = \frac{1}{17} \times 170 = 10$$

The theoretical degraded fitness, according to equation (5.8.5), is calculated as:

$$f_i' = \frac{f_i}{m_i'} = \frac{1}{10} \approx 0 \cdot 1$$

The variance of the expected number of individuals in each partition is:

$$\sigma_i^{\ 2} = N \cdot p_i \cdot (1 - p_i) = 170 \times \frac{1}{17} \times \frac{16}{17} \approx 9$$

Where :

$$p_i = \frac{\mu_i}{N} = \frac{1}{17}$$

Looking at the simulation results, we observe that the degraded fitnesses for each *H/h* value is close to the theoretical value and the genetic algorithm has found 17 alternative optimal results, for each specified design partition.

From section 5.7.7, we recall that, the LVDT design optimization problem, at least for more relaxed design criteria, was found to contain alternative optimal designs. In order to investigate this issue further, we consider the design optimization of an LVDT with the user required sensitivity of 400 vm^{-1} and a minimal non-linearity of 1·0%. We use the same design search space as specified in section 5.7.5, i.e.:

$$1 \cdot 0 \times 10^{-3} m \leq b \leq 3 \cdot 0 \times 10^{-2} m$$

$$1 \cdot 0 \times 10^{-3} m \leq m \leq 3 \cdot 0 \times 10^{-2} m$$

$$1 \cdot 0 \times 10^{-3} m \leq L_a \leq 2 \cdot 7 \times 10^{-2} m$$

$$1 \cdot 0 \times 10^{-3} m \leq r_i \leq 6 \cdot 0 \times 10^{-3} m$$

$$2 \cdot 0 \times 10^{-3} m \leq r_o \leq 1 \cdot 0 \times 10^{-2} m$$

$$10\, Hz \leq f \leq 400\, Hz$$

It is important to note that, in the above search space, there is a significant difference between the absolute values of permissible ranges of f (representing the frequency) and other independent design variables. Therefore, a direct use of the distance norm, as specified by equation (5.8.2), would result in f dominating this expression and in effect the sharing function would be strongly dominated by this variable. In order to remedy this situation, we use a normalized formulation, in which the distance norm is specified in terms of the underlying decoded unsigned fixed-point integer coding rather than the actual mapped values. Therefore, for each of our independent design variables, its normalized decoded value, to be used in conjunction with the sharing function (equation 5.8.1), is given by :

$$\hat{X} = \frac{\sum_{i=1}^{l} x_i \, 2^{i-1}}{\sum_{i=1}^{l} 2^{i-1}} \qquad (5.8.7)$$

In the above formulation, $\hat{X}$ represents the normalized decoded value for each of our independent variables, l represents the length of each X sub-string, where $X = x_l \, x_{l-1} \cdots x_2 \, x_1$ and each bit, $x_i \in \{0,1\}$, is subscripted by its position.

In this way, the normalized phenotypic values bocome independent of a particular design search space. Our new normalized sharing function takes into account the similarity of all independent design variables evenly and in a straight-forward linear function of the degree of similarity among them.

We are now in a position to run our next simulation. In this experiment, we would like to find 20 alternative LVDT designs satisfying our specified design criteria in the range :

$$0 \cdot 0 \, \langle \, \hat{d}_{ij} \, \langle \, \sqrt{6}$$

Where, 6 is the number of design variables and the distance-norm is our formulated normalized distance. Using the normalized distance norm formulation, σ_{share} is calculated as follows :

$$r = \frac{1}{2}\sqrt{6 \times \left(1 \cdot 0 - 0 \cdot 0\right)^2} \approx 1 \cdot 2247$$

$$\sigma_{share} = \frac{r}{\sqrt[p]{q}} \approx 0\cdot 7434$$

Genetic algorithm parameters used in this experiment are given below :

$$\begin{aligned} p_m &= 0{\cdot}003 \\ p_c &= 0{\cdot}95 \\ \textit{Pop-size} &= 400 \\ \textit{Max number of generations} &= 140 \end{aligned}$$

The fitness function for this experiment is as defined by equation 5.7.6.1. The mutation rate is chosen in line with DeJong's (1975) suggestion. (i.e., $p_m \approx$ 1/*Pop-size*). Following the considerations of section 5.7.6 (i.e., the expected number of candidate designs having near zero fitness value is more than 50 %), we have increased the population-size to 400. Table 3.14 shows the result of the simulation, for 11 consecutive normalized partitions, at generation 140. For computer generated results, refer to APPENDIX-IV.

	Independent design Parameters						**Dependent Variables**				
Partition	B(mm)	M(mm)	L_t (mm)	R_i (mm)	R_o (mm)	F(Hz)	%NLT	sensitivity	Normalized Distance	Fitness	Shared Fitness
0.7961	1.6	10.6	19.3	1.3	3.8	83	1.48%	$389.2vm^{-1}$	0.8327	0.7763	0.0418
0.9185	1.5	13.2	23.6	1.1	2.3	63	1.16%	$380.5vm^{-1}$	0.9778	0.9247	0.0644
1.0410	2.4	13.5	25.3	1.5	4.5	64	1.00%	$398.5vm^{-1}$	1.0949	0.9956	0.0481
1.1635	7.2	13.9	26.5	1.1	2.0	62	0.995%	$398.4vm^{-1}$	1.1064	0.9965	0.0550
1.2859	6.4	13.7	26.2	2.4	6.0	64	0.998%	$398.6vm^{-1}$	1.2294	0.9981	0.0420
1.4084	8.0	27.4	26.7	1.0	2.4	64	0.996%	$399.7vm^{-1}$	1.3726	0.9989	0.0621
1.5309	1.5	14.4	25.5	3.5	9.2	63	0.997%	$400.1vm^{-1}$	1.4785	0.9979	0.0578
1.6534	1.6	14.4	25.5	5.7	8.5	59	0.996%	$399.9vm^{-1}$	1.6278	0.9990	0.0561
1.7759	4.0	13.3	25.8	5.6	10.0	61	0.995%	$401.3vm^{-1}$	1.7273	0.9962	0.0648
1.8983	8.4	29.1	26.9	5.5	9.9	64	0.991%	$401.4vm^{-1}$	1.9523	0.9977	0.0508
2.0208	6.6	28.9	26.5	5.8	10.0	62	0.981%	$399.1vm^{-1}$	1.9615	0.9914	0.0738

Table 3.14 Results for the multi-modal design optimization of an LVDT

Looking at these results, we observe that the program has found 10 alternative designs in the range:

$$0{\cdot}918 < \hat{d}_{i0} < 2{\cdot}021$$

Where $\hat{d}_{i0}$ represents the normalized distance of a candidate design's independent design variables from the origin of the hyperspace (i.e., individuals 8, 9, · · ·, 17). At other partitions, relative local optima has been found which are sub-optimal with respect to the desired performance criteria. This is of no surprise as, at low and high limits of $\hat{d}_{i0}$, we don't expect to find optimal designs. For example, individual number 20 (please refer to APPENDIX-IV) is expected to have a $\hat{d}_{i0} \approx 2{\cdot}4$, which means that the normalized distance of its independent design variables must not be, approximately, below 5% of the maximum normalized value. At these high values of independent design variables we don't expect to find optimal designs.

It is reassuring that near optimal alternative designs have been found by this simple procedure, confirming our expectation that, at least for more relaxed design criteria, we must have different alternative optimal LVDT designs. I.e., our sharing function has been able to maintain stable sub-populations around each normalized distance partition representing an optima. This shows how the sharing function maintains appropriate diversity (i.e, the necessary, sometimes competing schemata) required to exploit all of the design sub-spaces.

A number of suggestions, for improving the performance of the sharing scheme, has been proposed in the past (Booker, 1982; Deb, 1989a; Goldberg, 1987b). These improvements are proposed to avoid cross-over between strings on different peaks which may result in offspring that do not represent any peak. The presence of these lethal strings in the population degrade the on-line performance of the process. In nature, this problem is avoided by creating separate species (or subpopulations) corresponding to each niche (or peak) in the solution space and restricting the mating between species. Therefore, in future work, using an appropriate performance metric, we must implement and experiment with different mating restriction strategies together with the sharing scheme.

5.9 Conclusions

Genetic algorithms are highly adaptive processes which can efficiently search environments characterized as discontinous, vastly multimodal and noisy, without auxiliary information requirement.

However, genetic algorithms occasionally suffer from "premature convergence". The primary cause of this behaviour is identified as "genetic drift" that characterizes the stochastic errors caused by reproductive strategies (section 5.4.1). A secondary cause of this behaviour might be caused by genetic algorithm hard problems (section 5.4.2).

In order to investigate these issues further, the design optimization of two instrument sub-systems (i.e., corrugated diaphragms and LVDTs), using a three operator genetic algorithm, has been undertaken.

Proponents of proportionate selection strategies have devised a number of fitness scaling procedures to avoid stochastic errors due to genetic drift. However, these procedures are somehow ad-hoc and complicate the genetic algorithm simulations by adding extra parameters for controlling selective pressure.

In the context of our optimization problems, an stochastic remainder without replacement selection, in conjunction with elitisism and an scaling process (i.e ST-El-Wn), gave relatively superior, near optimal results as compared to stochastic remainder selection strategy alone.

Although St-El-Wn can reduce the stochastic errors due to similarity of fitnesses near convergence, it is not effective in maintaining diversity across the gene pool, i.e., a chromosome with substantially higher fitness, will tend to dominate the population, causing premature convergence.

In order to investigate a different selection strategy, a ranking selection has been implemented. In this method, the whole population is first sorted by fitness. The number of offspring each chromosome generates is determined by how it ranks in the population. The result obtained from our ranking selection strategy, in conjunction with elitisism, are substantially better as compared to previous selecton methods used. This comparison is based on a best of run performance metric. The ranking selection, completely solves the scaling problem and provides a consistent means of controlling offspring allocation. In general, ranking methods provide an even, controlled pressure for the selection of better individuals.

In the context of our design optimization problems, we have also considered a number of niching schemes. Theoretically, (i.e., according to the fundamental theorem of genetic algorithms) genetic algorithms must maintain useful diversity and ideally they must converge to all the peaks in a multimodal optimization problem. However, again, due to genetic drift, genetic algorithms which are only based on proportionate selection strategies, will only exploit the best peak and in a multimodal function with peaks having equal fitness they will only converge to one of the alternatives.

The sharing scheme, together with a proportionate selection strategy, is an effort to maintain a pressure to balance the sub-population sizes around each peak; by making sure that strings are reproduced in accordance with shared fitness values.

By using our sharing schemes, together with the stochastic remainder without replacement selection, alternative optimal designs have been found for both of our design optimization problems. This confirms that, for more relaxed user specified design criteria, there will be alternative optimal designs in both cases.

As mentioned above, a secondary cause of premature convergence is due to genetic algorithm hard problems. In the problems considered in this chapter, there is no significant evidence of genetic algorithm hardness. As elaborated in section 5.4.2, this problem occurs only in exceptional search spaces in which the best optimal points are surrounded by the worst.

CHAPTER 6

Classifier Systems for the Automation of Inductive Reasoning in the Design Process

6.1 Introduction

Classifier systems have been proposed as a direct result of recent theoretical investigations into the nature of inductive reasoning (chapter 4). Because classifier systems are formally defined and computer-oriented, with an emphasis on combination and competition, they offer a useful test-bed for both mathematical and simulation studies of induction.

In this chapter, we first present classifier system advantages, applications and research issues (section 6.2). Our goal, in this chapter, is to investigate the applications of classifier systems to the design of instruments. To this end, we detail a number of proposals (section 6.3). In particular, we will investigate the feasibility of simulating parametric sensitivity analysis and dimensionless analysis, as done by designers, to establish design heuristics (section 6.4). For this purpose, a functional lumped parameter mathematical model is interfaced to an implemented classifier rule-based system.

The task of the classifier system is to discover important design heuristics related to the profile geometry of the corrugated diaphragms, the size and range of each appropriate design parameter, and the nature of its influence on the overall performance of the diaphragm.

For example, a design rule such as: "Rule-1: For any Relative Corrugation Depth (i.e., H/h), the influence of the Centre Boss on the overall input-output characteristics of a Corrugated Diaphragm can not be neglected" provides us with a general rule which can be applied to all design categories, and gives us default expectations. However, the application of Rule-1 might, unnecessarily, complicate the design of a particular diaphragm. A more specific rule such as: "Rule-2: For deep Corrugation Depths, the influence of the Relative Radius of the Centre Boss, up to 0.5, can be neglected" provides us with a more specific rule. In other words, Rule-2 can be applied, only, to more specific categories of diaphragms, and will simplify the design process of such diaphragms. A cluster of rules, such as Rule-1 and Rule-2, when applied to similar class of design problems, provide us with a default hierarchy of rules (chapter 4) in which more general rules cover the general conditions and more specific, possibly overlapping rules cover the exceptions. Our main goal, in this chapter, is to investigate the application of classifier systems for the purpose of inducing such useful design heuristics, in the form of default hierarchies.

6.2 Classifier System Applications

The first application of a classifier system was presented by Holland and Rietman (1978). The 1978 implementation of Holland and Rietman, called CS-1 (Cognitive System Level One), was trained to learn two maze-running tasks. CS-1 demonstrated simple transfer of learning from problem to problem and showed that the genetic algorithm yielded learning, in that context, an order of magnitude faster than weight-changing techniques alone. The results were encouraging enough to initiate a variety of subsequent tests.

Smith (1980) completed a classifier system that competed against Waterman's poker player (Waterman & Hayes-Roth (1978); which was also a learning program) with overwhelming success.

Wilson (1982) used a classifier system with a genetic algorithm in a series of experiments involving TV-camera-mechanical-arm co-ordination, resulting in a successful demonstration of the segregation of classifiers, under learning, into sets corresponding to control subroutines.

The next major application of classifier systems was Booker's (1982) study. Booker concentrated on the formal connections between cognitive science and classifier systems. His computer simulations investigated the adaptive behaviour of an artificial creature, moving about in a two-dimensional environment containing "food" and "poison", controlled by a classifier system "brain". Booker's classifier system contained a number of innovations including the use of sharing to promote "niche" exploitation, and the use of mating restrictions to reduce the production of ineffective offspring (lethals).

Goldberg (1983) applied a classifier system to the control of two engineering systems: a pole-balancing problem and a natural gas pipeline-compressor system. Goldberg demonstrated the emergence of a default hierarchy (in his study of the use of classifier systems under the genetic algorithm) as adaptive controls for gas pipeline transmission.

More recent work on classifier systems have considered the application of classifier systems to specific engineering problems, such as maze running tasks for autonomous robots, signal processing, scheduling for operation research applications and simple medical diagnosis (Davis, 1987; Schaffer, 1989; Belew, 1991). However, there is very little research concerned with the application of classifier systems to engineering design automation.

6.2.1 Classifier Systems advantages and research problems

The use of genetic algorithms as the primary rule discovery component in classifier systems is highly advantageous. The genetic algorithm, operating on classifiers, discovers potentially useful building blocks, tests them, and recombines them to form plausible new classifiers. This is done at the large "speed up" implied by implicit parallelism (Holland, 1975), during which large numbers of building blocks are searched while relatively few classifiers are manipulated.

Competition based on rule strength, in conjunction with the parallelism of classifier systems provides several additional advantages. New rules can be added without imposing the severe computational burden of checking their consistency with all the existing rules. In fact, the system can retain large numbers of mutually contradictory, partially confirmed rules; an important advantage because these rules serve as alternative hypotheses to be invoked when currently more plausible rules prove to be inadequate. Moreover, this approach in conjunction with the genetic algorithm provides the overall system with robust incremental means of handling noisy data. The system does not need a large data-base of all past examples; its memory is composed of the sets of competing alternative rules.

The above highly favourable characteristics make classifier systems general-purpose learning systems. They can be programmed initially to implement whatever expert knowledge is available to the designer; learning then allows the system to expand, correct errors, and transfer information from one domain to another. However, at this stage of classifier systems development, it is important to provide ways of instructing such systems so that they can generate rules - tentative hypotheses - on the basis of advice. Little work has been done in this direction (Holland, 1986a).

The most serious problem encountered, in the classifier system applications, concerns the stability of emergent default hierarchies. The hierarchies do emerge (Goldberg (1983) provides one of the few known examples), but in long runs there may be a catastrophic collapse in which whole subsets of good rules are lost. The rules, or rules similar in function, are then recreated, but this instability is highly undesirable.

Forrest (1985) has demonstrated that semantic nets can be implemented directly with coupled classifier rules, but the question of how such structures can emerge in response to interactions with the environment has not been tackled. However, this remains a research objective and not a fault to the overall approach.

There are also no appropriate guidelines as to the functioning of the bucket brigade when the rule sequences are relatively long and intertwined. Again, there are no uncovered faults; there is simply very little knowledge.

6.3 Application of Classifier Systems to the design of - Instruments

In the past, mathematical models have been used extensively to automate the design process of instrument transducers.

The building of mathematical models involves listing all possible physical laws and processes that may affect the response of the proposed transducer concept. The modeller must then use his engineering judgement, based on past experience to ascertain whether any of the listed phenomena may be neglected. Furthermore, it may be appropriate to make some simple approximations regarding the sub-assemblies or constituents of the transducer. These simplifications lead to a set of design parameters that are, by simplification and abstractions, mapped into "functional elements" (generalized resistances, capacitances, inductances, transformers, etc.) and their interconnections. This newly obtained mathematical description is called a "functional model". For example, in the case of electromechanical transducers, this means making some simple approximation to a generated magnetic flux distribution. However, it may be necessary to develop more accurate models that are called "physical models". Generally, these models would be distributed parameter models based on the solution of the appropriate equations (i.e. differential, etc.). These equations emerge as a direct application of the basic physical laws to the design concept. Consequently, the produced models enable us to evaluate the performance of a transducer in terms of its geometric dimensions and material properties. Results from the validated mathematical models for a specific designed prototype are then used to generate a general design methodology (Mirza, 1992). A variety of techniques are used for this purpose:

1- The methods of sensitivity analysis (Tomovic, 1970) can be used to determine the important variables for a particular fully specified design configuration. Engineering common sense guided by geometric constraints can be used to estimate the values for the other parameters.

2- The basic mathematical models can be used in larger computer aided design system to produce optimal designs with constraints. In this type of computer aided design system the basic models are used within a larger program that uses optimization techniques to find the optimal set of independent design variables to meet a particular set of design criteria. In chapter 5, sections 5.5, 5.6, and 5.7, using genetic algorithms, it was shown that useful design heuristics can be obtained by extracting data from several optimization simulations for a particular transducer. In order to automate the process of heuristic extraction, we can interface the training data (extracted from simulations for a transducer) to a distributed inductive learning system (e.g., classifier systems or neural networks). For example, in the case of LVDTs the training set consists of a set of representative LVDT configurations. A representative LVDT configuration includes design parameters such as the overall parametric dimensions of the transducer, physical layout and its corresponding performance function values. The inductive learning system uses the performance function values, constituting the performance metrics of the LVDT, to establish design heuristics that relate the influence of design parameters to the overall performance of the instrument. The performance metric, for a particular LVDT configuration, includes performance indicators such as the input/output sensitivity, % non-linearity and dimensional constraints such as size and weight of the transducer.

3- The methods of dimensional analysis (Massey, 1971) can be invoked to form non-dimensional performance curves that characterize a particular design prototype. The concept of non-dimensional curves is based on similarity. The curves are obtained for a specific case but are applicable to any design candidate having the same parameter ratios. The performance curves contain all the essential information regarding the class of instrument. The results can be re-arranged in different ways to facilitate the establishment of a simple design methodology (Mirza, 1983).

During sensitivity analysis and dimensional analysis, the designer, using his set of data obtained for a device or system, is trying to find the relationship underlying empirically observed values of the design variables to generate a set of generalized empirical relationships for design purposes. In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way. These methods (for design heuristic extraction) are recognized as learning synchronic and diachronic rules from noisy and uncertain data and their automation is possible by using classifier systems.

In the next section, we will study this issue, further, by applying a classifier system to the task of design heuristic extraction for corrugated diaphragms.

6.4 The application of a classifier system, for design heuristic-discovery and learning, to corrugated diaphragms

In this section, we investigate the feasibility of applying classifier rule-based systems to the design rule discovery and learning for corrugated diaphragms. The design parameters for a corrugated diaphragm are:

θ_0 : Profile-angle (Radians)
H : Profile depth (mm)
h : Diaphragm's thickness (mm)
E : Young's modulus (kg/m^2)
R_o : Diaphragm effective radius (mm)
R_b : Diaphragm center-boss radius (mm)

The aim of these investigations is to implement a system that discovers the most important of these parameters and generates design heuristics relating the size and range of each parameter and the nature of its influence on the overall performance of the diaphragm. This process is similar to parametric sensitivity analysis and dimensionless analysis used by designers for the extraction of design heuristics (Mirza, 1983).

In order to investigate the feasibility of this idea, a classifier system is implemented in which the system must discover design rules relating the centre-boss size, the diaphragm radius and relative corrugation depth to the overall characteristics of a corrugated diaphragm type.

Corrugated diaphragms have a metallic disk welded or soldered to their middle part that is called a rigid centre (the centre boss). If the dimensions of the rigid centre are small, its influence on the diaphragm deflections is negligible. However, if the centre boss is large, its influence must be taken into account. The derivation of the characteristic equation for a diaphragm with a rigid centre is based on the same differential equations (generated for a flat anisotropic diaphragm with large deflections) just as they are for a diaphragm without a rigid centre. The only difference in the derivation is in the boundary conditions (Andreeva, 1966).

The following characteristic equation for corrugated diaphragms with a rigid centre is obtained as a result:

$$\frac{PR^4}{Eh^4} = \eta_p \cdot a_p \cdot \frac{\omega_0}{h} + \xi_p \cdot b_p \cdot \frac{\omega_0^3}{h^3} \tag{6.4.1}$$

The coefficients a_p and b_p in the above formula are determined as before (please refer to chapter 5) and the correction coefficients η_p and ξ_p depend on both the corrugation geometry and the relative radius ϑ_o of the rigid centre:

$$\eta_p = \frac{(3-\alpha)(1-\alpha)}{(3+\alpha^2)(1-\vartheta_o^4)+\left(\frac{4\alpha}{1-\vartheta_o^{2\alpha}}\right)\left[2\vartheta_o^{\alpha+1}(1+\vartheta_o^2)-(1+\vartheta_o^{2\alpha})(1+\vartheta_o^4)\right]} \tag{6.4.2}$$

$$\xi_p = \frac{1}{(1-\vartheta_o^2)^4(1+\vartheta_o^2)\left[\frac{1}{6}-\frac{3-\mu}{(\alpha-\mu)(\alpha+3)}\right]} \times \left\{\frac{1-\vartheta_o^6}{6}-\frac{3-\mu}{1-\vartheta_o^{2\alpha}} \times \left[\frac{(1-\vartheta_o^{\alpha+3})^2}{(\alpha-\mu)(3+\alpha)}+\frac{(\vartheta_o^{\alpha}-\vartheta_o^3)^2}{(\alpha+\mu)(3-\alpha)}\right]\right\}$$

Where $\vartheta_o = \frac{R_b}{R}$ is the relative radius of the rigid centre.

Using the dimensionless formulation, represented by equation (6.4.1), Andreeva (1966) has derived dimension-less curves, representing families of characteristics, for diaphragms of two types: with shallow (H/h = 2·0) and deep (H/h = 10·0) corrugations and various dimensions of the centre-boss. Andreeva, by using these curves, has developed the following design heuristics:

"A rigid centre with a relative radius ϑ_o not exceeding 0·3 has only a small influence on the characteristics of shallow-corrugation diaphragms. In the case of deeply corrugated diaphragms, a rigid centre with a relative radius up to 0·5 can be neglected. Further increase of ϑ_o in both cases leads to a sharp increase of the diaphragm spring rate. The deeper the corrugations, the smaller the influence of the rigid centre on the diaphragm characteristics. This is due to the fact that, in general, the centre beads are much less deformed than the edge beads, so that the substitution of a rigid centre for the centre beads will not have an important influence on the characteristics if the beads are sufficiently deep."

In order to simulate Andreeva's empirical rule discovery approach, we will interface the design environment, as represented by equation (6.4.1), to an implemented classifier rule-based system.

In the following sections, we, first, introduce the design environment of the classifier system from which design heuristics are to be established. We then study the structure of the learning classifier system by detailing each of its components: The rule and message system, the apportionment of credit system and the genetic algorithm system.

6.4.1 The design environment

As explained in section 6.4, Andreeva, using the following analytical model, has developed design heuristics that relate the centre-boss size, the effective radius and relative corrugation depth to the overall characteristics of corrugated diaphragms that have different profile geometries:

$$\frac{PR^4}{Eh^4} = \eta_p \cdot a_p \cdot \frac{\omega_0}{h} + \xi_p \cdot b_p \cdot \frac{\omega_0^3}{h^3}$$

Using Andreeva's analytical model, the design environment is represented to the classifier system in terms of random message strings containing information regarding the radious ratio (ϑ_o) of the centre-boss and the relative corrugation depth (*H/h*), for a particular diaphragm design task, as follows:

H/h	ϑ_o
Sub-string-1	Sub-string-2

Figure 6.2

The number of bits for each sub-string parameter provides the degree of accuracy required. Our design parameters obey the following constraints:

$$\begin{aligned} &0 \leq \vartheta_o \leq 1 \\ &1 \leq {}^{H}\!/_{h} \leq 17 \end{aligned} \qquad (6.4.4.1)$$

Each design parameter is represented by a 3-bit sub-string. This gives the following accuracy, per bit increment, for each parameter:

$$\begin{aligned} &\Pi(\vartheta_o) = \frac{1-0}{2^3-1} \approx 0{\cdot}143 \\ &\Pi\left({}^{H}\!/_{h}\right) = \frac{17-1}{2^3-1} \approx 2{\cdot}286 \end{aligned} \qquad (6.4.4.2)$$

The environment has been designed for the simulation of design heuristic extractions, used by designers during the initial stages of the design process. It is based on the dimensionless analytical model formulation, represented by equation 6.4.1.

Andreeva (1966) used two *H/h* values, i.e. *H/h = 2.0* and *H/h = 10.0*, to represent two classes of corrugated diaphragms with shallow and deep corrugations. Furthermore, she used only 5 discrete values for the ϑ_o parameter. Therefore, her induced design heuristics are based on only 10 design classes. However, because of the well-defined, monotonous behaviour of the input-output characteristics of corrugated diaphragms, in-between the discrete values, Andreeva's inductive approach is valid and covers all design cases to a good approximation. Our environmental formulation is much more accurate than Andreeva, because it represents 64 design classes as the basis for inductive learning. In more complex inductive learning applications, i.e., non-monotonic and/or erratic environments, we can simply increase the resolution of the design parameters.

The design environment has the following reward criterion to allocate reward to the successful classifier rules:

$$Criterion(H/h,\vartheta_o) = Boolean\left(\frac{\left|output\_2(H/h,\vartheta_o) - output\_1(H/h,\vartheta_o)\right|}{output\_2(H/h,\vartheta_o)} \le E\right)$$

$$Reward = \begin{cases} 1 & if \quad classifier\_output = Criterion(H/h,\vartheta_o) \\ 0 & if \quad classifier\_output \neq Criterion(H/h,\vartheta_o) \end{cases} \qquad (6.4.4.3)$$

Where:

E : acceptable range of error = Maximum 10.0% deviation

$$output\_1 \;=\; \frac{PR^4}{Eh^4} = a_p \cdot \frac{\omega_0}{h} + b_p \cdot \frac{\omega_0^3}{h^3}$$

$$output\_2 \;=\; \frac{PR^4}{Eh^4} = \eta_p \cdot a_p \cdot \frac{\omega_0}{h} + \xi_p \cdot b_p \cdot \frac{\omega_0^3}{h^3}$$

$\frac{\omega_0}{h} = 6 \cdot 0$ → Dimensionless nominal value of diaphragm centre displacement

In the above formulation, *output_1* represents the analytical model of the diaphragm in which the effect of the relative radius ratio of the rigid centre(ϑ_o) and its relation to the overall profile geometry of the diaphragm has been neglected, but the *output_2* represents a more accurate mathematical model in which these influences on the overall output characteristics have been considered.

It must be emphasized that, the environment allocates reward, entirely, based on our analytical model formulation. In this particular rule discovery task, environmental messages relate to different diaphragm design problems. Therefore, different environmental messages, lead the classifier system to form different categories of design problems, for which different design actions are necessary.

6.4.2 The Rule and Message system

The rule and message system are central to the operation of the classifier system; it provides the computational framework for classifier system thought and action and is the foundation for competitive service economy and genetic algorithmic learning.

Figure 6.1 shows a schematic of the rule and message system.

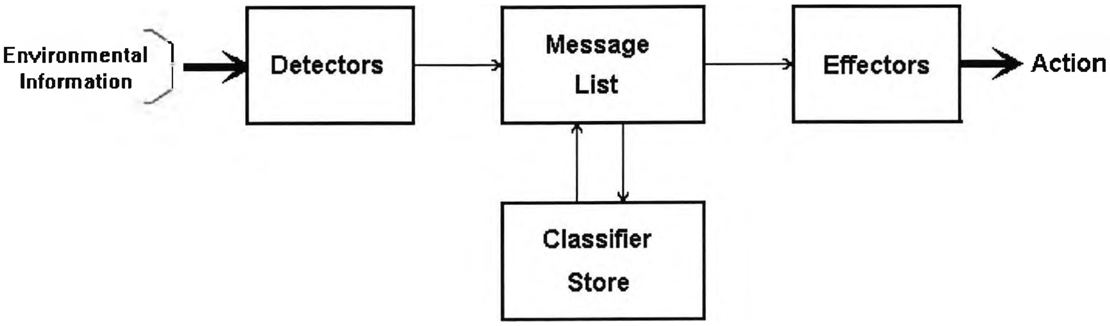


Figure 6.1 Schematic Rule and Message System

In Figure 6.1, we see that, the rule and message system receive environmental information through its sensors, called "detectors", which are decoded to some standard message format. This environmental message is placed on a "message-list" along with a finite number of other internal

messages generated from the previous cycle. Messages on the message list may activate "classifiers" (rules) in the "classifier store". If activated, a classifier may then be chosen to send a message to the message list for the next cycle. In addition, certain messages may cause external action through a number of action triggers called "effectors". In this way, the rule and message system combine both external and internal data to guide behaviour and the "state of mind" in the next state cycle.

In summary, the basic execution cycle of a classifier system consists of the following steps:

Step-1: Add all messages from the input interface (i.e., detectors) to the message list.

Step-2: Compare all messages on the message list to all conditions of all classifiers and record all matches (i.e., satisfied conditions).

Step-3: For each set of matches satisfying the condition part of some classifier, post the message specified by its action part to a list of new messages.

Step-4: Replace all messages on the message list by the list of new messages.

Step-5: Translate messages on the message list to requirements on the output interface (i.e., effectors), thereby producing the system current output.

Step-6: Return to step 1.

Individual classifier rules must have a simple, compact definition; a complex, interpreted definition makes it difficult for the genetic learning algorithm to find and exploit building blocks from which to construct new rules.

In classifier systems a message is simply a string of fixed length l, over some finite alphabet A. In this discussion, A is limited to the binary alphabet {0, 1} without loss of generality. More formally a message is defined as follows :

$$\langle message \rangle \rightarrow \{0,1\}^{l}$$

Messages may contain a variety of information, coded in any imaginable manner. At a minimum, messages carry environmental input information, internal tags, internal data and effector codings.

Messages are processed by classifier rules. Recall that classifiers are a form of rule in the tradition of rule-based expert systems. For this study, we limit classifier rules to the following form:

$$\langle classifier \rangle \rightarrow \langle condition\_1 \rangle \langle condition\_2 \rangle \langle message \rangle$$

As in rule-based expert systems, the message action of a classifier rule is fired upon satisfaction of both conditions; the overall condition is constructed by using the logical AND primitive in this case. Other logical primitives from which more complex expressions could be constructed are OR and NOT.

A condition is a recognition device that depends upon the presence of certain messages on the message list. It is important to construct conditions such that they can recognise not just a single message, but rather a class of messages with well-defined similarity. This is achieved by extending our message alphabet A by one character to the alphabet $A+ = \{0, 1, \#\}$. Therefore, a condition is defined as an l position string over $A+$:

$$\langle condition \rangle \rightarrow \{0,1,\#\}^{l}$$

Under the alphabet $A+$, at a given position, a 0 is matched by a 0, a 1 is matched by a 1, and a # is matched by either. For example, the string 1##...# designates the set of all messages that start with a 1, while the string 00...0# specifies the set {00...01, 00...00} consisting of exactly two messages and so on. In this way, the # is a wild card symbol permitting explicit recognition of any of the subset of messages with one or more similarities. This definition has immense advantage in a learning system that must generalize and create new rules from the ratings of the current rule store.

In practical classifier systems, the number of potential messages exceeds the size of the message list. In the next section, we study the way the apportionment of credit system (the bucket brigade algorithm) handles these and other conflicts which might arise.

6.4.3 Analysis of the Bucket Brigade Algorithm

The Bucket Brigade Algorithm is designed to solve the apportionment of credit problem for massively parallel, message-passing, rule-based systems.

In classifier systems, in which rules are represented by the standard Condition/Action paradigm, the overall usefulness of a rule to the system is indicated by a parameter called its strength. Each time a rule is active, the bucket brigade algorithm modifies the strength so that it provides a better estimate of the rule's usefulness in the contexts in which it is activated.

The bucket brigade algorithm functions by introducing an element of competition into the process of deciding which rules are activated. Normally, for a parallel message-passing system, all rules having condition parts satisfied by some of the messages posted at a given time are automatically activated at that time. But under the bucket brigade algorithm only some of the satisfied rules are activated. Each satisfied rule makes a bid, based in part on its strength, and only the highest bidders become active, posting their messages specified by their action parts. The size of the bid depends upon both the rule's strength and the specificity of the rule's conditions.

Specificity, for a classifier system rule, is defined as the difference between the total number of defining positions in its condition and the number of "don't cares" (i.e., "#" tokens) in the condition, and defines a measure of relevance of the rule to the particular environmental situation.

In a specific version of the algorithm, used extensively by current classifier systems, we have :

$$S_i(t+1)=S_i(t)-C_{bid}\cdot S_i(t)-C_{tax}\cdot S_i(t)+R(t) \qquad (6.4.2.1)$$

Where:

$S_i(t)$:strength of the active classifier i at time t
$S_i(t+1)$: strength of the active classifier i at time t+1
C_{bid} : bidding coefficient
C_{tax} : bidding tax coefficient
$R(t)$: environmental reward

We can rewrite equation (6.4.2.1) as:

$$S_i(t+1)=(1-k)\cdot S(t)+R(t) \qquad (6.4.2.2)$$

Where :

$k = C_{bid} + C_{tax}$

The above difference equation represents a linear discrete system. An n order difference equation can be represented as:

$$\begin{aligned} y(k) = b_0\, u(k) + b_1\, u(k-1) + \cdots + b_m\, u(k-m) + \\ - \alpha_1\, y(k-1) - \alpha_2\, y(k-2) - \cdots - \alpha_n\, y(k-n) \end{aligned} \qquad (6.4.2.3)$$

Difference equations of the type represented by equation (6.4.2.3), with m, n fixed and α_i, b_j i=1, 2, ..., n j=0,1,2, ..., m constants, describe linear discrete time causal and stationary systems. In non-stationary systems α_i, b_j are function of k.

The bucket brigade algorithm, as represented by equation (6.4.2.2), can be written as :

$$S_i(t)-\alpha_i \cdot S_i(t-1)=\beta\ u(t)$$

Where:

$$\alpha=(1-k)$$
$$\beta\ u(t)=R(t)$$

$$u(t)=\begin{cases}0 & for\ \ t=-1,-2,-3,\cdots\\ 1 & for\ \ t=0,1,2,3,\cdots\end{cases}$$

$u(t)$ represents a unit step discrete input to the system. When k=0 $S_i(0)=\beta\ u(0)+\alpha\ S_i(-1)$. The initial condition S_i(-1) is zero, therefore we get:

$$S_i(0)=\beta$$
$$S_i(1)=\beta\ u(1)+\alpha\ S_i(0)=(1+\alpha\beta\ )$$
$$S_i(2)=(1+\alpha+\alpha^2)\beta$$
$$\vdots$$
$$\therefore S_i(t)=(1+\alpha+\alpha^2+\cdots+\alpha^t)\beta\ ,\quad k=0,1,2,\cdots$$

But we have:

$$(1-\alpha^{t+1})\equiv(1-\alpha)(1+\alpha+\alpha^2+\cdots+\alpha^t)$$
$$\therefore\quad S_i(t)=\frac{\beta}{1-\alpha}(1-\alpha^{t+1}) \qquad (6.4.2.4)$$

Hence, for stability of the response, we have:

i) if $|\alpha|>1\Rightarrow\left.\begin{matrix}(1-\alpha^{t+1})\rightarrow\infty\\ t\rightarrow 0\end{matrix}\right\}\ unstable$

ii) if $|\alpha|<1 \Rightarrow \underset{t\to\infty}{(1-\alpha^{t+1})\to 1} \Rightarrow \left.\underset{t\to\infty}{S_i(t)\to\frac{\beta}{1-\alpha}}\right\}$ *response bounded*

Therefore, when $|\alpha|<1$ for large t the system's unit step response itself looks like a step signal of amplitude $\frac{\beta}{1-\alpha}$. If reward = $\beta \cdot u(t)=R_{ss}$ (constant), and knowing that $\alpha = 1-k$, we get:

$$\underset{t\to\infty}{S_{ss}=\frac{R_{ss}}{k}} \tag{6.4.2.5}$$

Knowing that $k = C_{bid} + C_{tax}$, we can rewrite equation (6.4.2.5) as:

$$S_{ss}=\frac{R_{ss}}{C_{bid}+C_{tax}} \tag{6.4.2.6}$$

In order to take into account the specificity of a classifier rule, we have:

$$B_i(t)=C_{bid}\cdot f\left(s_p\right)\cdot S_i(t) \tag{6.4.2.7}$$

Where:

$B_i(t)$: the bid of a classifier rule
$f\left(s_p\right)$: an increasing function of specificity for a rule

Therefore, taking into account the specificity of a rule, the steady state strength for a rule becomes:

$$S_{ss}=\frac{R_{ss}}{C_{bid}\cdot f\left(s_p\right)+C_{tax}} \tag{6.4.2.8}$$

6.4.4 Genetic Algorithms for Rule Discovery

The rule discovery process for classifier systems uses a genetic algorithm. Basically, a genetic algorithm is used to select high strength classifiers as "parents", forming "offspring" by recombining components from the parent classifiers. The three operator genetic algorithm (chapter 5) creates new rules by the reproduction, cross-over and mutation processes.

In this section, we concentrate on the differences between the genetic algorithm used in our classifier system and the three operator genetic algorithms used in chapter 5 for design optimization purposes. Specifically these differences include overlapping generations, crowding, Monte Carlo selection and ternary mutation as described below:

We recall from chapter 5, section 5.8, that DeJong (1975) introduced a generation gap (G) parameter (in a multimodal function optimization application), to permit overlapping populations. This parameter was defined between 0 and 1 as follows :

$G = 1$ non overlapping populations
$0 < G < 1$ overlapping populations

In the overlapping populations ($N * \quad G$) individuals are selected for further genetic action (where N is the population size). In the implemented genetic algorithms of the previous chapter, the populations were non-overlapping and we completely generated a new population at each iteration. The studies carried out by DeJong (1975) suggested that the non overlapping population model was best in most optimization studies, where off-line performance is of primary importance. However, DeJong's studies did show that on-line performance is not severely degraded by using smaller generation gap values. This fact is useful in machine learning where learning while performing well is important; in machine learning applications, we are mostly concerned with maintaining a high level of on-line performance as the system learns to perform better. In the same spirit, Goldberg (1983) used an overlapping population, successfully, in a rule learning application.

In the implemented classifier system, we use the generation gap (G) parameter and generate (No. of classifiers $* \; G$) new classifiers at each call to the genetic algorithm. Also, we use DeJong's (1975) crowding procedure (described in chapter 5, section 5.8) to encourage replacement of similar population members.

In chapter 5, we studied a number of selection strategies. Although, the ranking selection strategy has certain advantages, on the basis of the best of run performance metric as compared to the proportionate selection techniques studied, it becomes difficult and time consuming to implement with overlapping populations. Consequently, we use the Monte Carlo selection strategy (DeJong, 1975), in conjunction with the overlapping populations, which is one of the simplest of the proportionate selection strategies. This method simulates the spin of a weighted roulette wheel; where the wheel weights are given by:

$$\left(S_j \middle/ \sum S_i \right),$$

where S_j represents the strength of a particular classifier. We select (No. of classifiers $*$ G) new offspring classifiers at each genetic algorithmic invocation. In this way, the parent selection is biased towards high strength classifier rules and schemata in the population.

Mutation is also modified because classifier systems use a ternary alphabet. In order to achieve this, the probability of mutation p_m is defined as before, however, when a mutation is invoked, we change the mutated character to one of the other two with equal probability, i.e.: (0 → {1, #}, 1 → {0, #}, # → {0, 1}.

The overall software architecture of the implemented system is presented in APPENDIX-V.

6.4.5 An initial study of the classifier system under the- apportionment of credit algorithm

The implemented classifier system is initially provided with a random population of 100 rules in the following form:

$$\begin{array}{cccc} \langle H/h\,Range \rangle & \langle H/h\,Range \rangle & \rightarrow & \langle Decision\ Action \rangle \\ \textbf{Condition-1} & \textbf{Condition-2} & & \textbf{Design Action} \end{array}$$

The probability of generating a don't care token (i.e., #) is set to 0·7. Our concern, at this stage, is only to identify the above average rules in our randomly generated rule population. Consequently, no genetic rule improvement and discovery are invoked. Also, we do not use specificity dependent bidding to test the basic performance of the apportionment of credit algorithm. The governing apportionment of credit algorithm, for this simulation, is given below:

$$S_i(t+1)=S_i(t)-C_{bid}\cdot S_i(t)-C_{bid-tax}\cdot S_i(t)+R(t) \qquad (6.4.5.1)$$

Where:

$S_i(t)$:strength of the active classifier i at time t
$S_i(t+1)$: strength of the active classifier i at time t+1
C_{bid} : bidding coefficient
$C_{bid-tax}$: bidding tax coefficient
$R(t)$: environmental reward

The learning parameter values for the apportionment of credit algorithm, used in this simulation, are given in the following table:

C_{bid}	0·1
$C_{bid-tax}$	0·01
$R(t)$	1·0

Table 6.1 Learning parameters for the initial simulation

Therefore, under this scheme, each above average rule is expected to reach its asymptotic strength value of 9·09 as calculated by equation (6.4.2.6).

The classifier system, under the application of the apportionment of credit algorithm, successfully assigned high strength values to high performance rules (i.e., near to their expected asymptotic steady state strength values) and near zero strength to irrelevant or low performance rules among the initial randomly generated population. This was achieved only after 1500 iterations. Some of the high performance rules discovered by the classifier system, in this simulation, are shown below:

Strength	Rule	Interpretation
9·09	1## 1## → 1	large(H/h) AND large(ϑ_o) → strong influence
9·09	#1# 0#0 → 0	(5·6 ≤ H/h ≤ 7·8) AND (14·7 ≤ H/h ≤ 17) AND small(ϑ_o) → small influence

Table 6.2 Some of the high performance rules in an initial simulation

For example, the design rule (1## 1## → 1), in the above table, covers 16 design classes, with its first condition covering H/h values in the range $10.14 < H/h < 17.0$ and its second condition covering ϑ_o values in the range $0.57 < \vartheta_o < 1.0$. Therefore, this rule covers high values of the design parameters.

Obviously, the above rules are less than perfect and it is necessary to improve them by invoking the genetic rule-improvement module. However, at this stage, our concern has been to test the classifier system performance only under the application of the apportionment of credit algorithm. We note that the rules, shown in table 6.2, have been correct in each invocation, and have reached theirs expected steady state strength values.

Using the set of high performance rules, generated by the classifier system, they were modified to construct a set of perfect rules. The set of perfect rules partitions the design environment into six non-overlapping design regions (i.e., each region represents a particular category of design problem for which an appropriate design action is necessary) as shown in table 6.3 below:

small-influence design rules :		**strong-influence design rules :**	
1)	### 00# → 0	5)	### 1## → 1
2)	01# 010 → 0	6)	00# 010 → 1
3)	1## 01# → 0	7)	0## 011 → 1
small-influence default rule :		**strong-influence default rule :**	
4)	### ### → 1	8)	### ### → 0

Table 6.3 Two sets of hand-crafted perfect design rules

In order to test the performance of the classifier system, two cases are investigated:

1) Perfect-rule set :

To test the performance of our perfect rule set, under the apportionment of credit algorithm, the learning parameter values of table 6.1 were used.

All rules (i.e., rules (1), (2), (3), (5), (6) and (7) in table 6.3) reached theirs expected steady-state strength value of 9·09, as suggested by equation (6.4.2.6). I.e., each rule was rewarded when it matched the message-list.

2) Default Rules:

Default hierarchies encourage more efficient learning in classifier systems. There are alternative methods to encourage the formation of default hierarchies. We recall from chapter 4, section 4.2.1, that, Holland (1986) has suggested to make the "bid" of each classifier rule proportional to the product of its strength and relevance. Relevance is a function of the "specificity" of

the condition of a matched rule, and the more specific the condition of a rule, the more relevant it becomes. The simplest formulation for relevance is to define it as some linear function of "specificity". I.e.:

$$B_i = C_{bid} \cdot f\left(s_p\right) \cdot S_i \qquad (6.4.5.2)$$

Where:

B_i : bid of a candidate classifier rule
S_i : strength of the bidding classifier
s_p : specificity of a classifier rule

and we have:

$$f\left(s_p\right) = M_1 + M_2 * s_p$$

$f(s_p)$ is chosen to normalize the specificity value of a classifier rule between some initial value M_1, for $s_p = 0$, and final value $(M_1 + M_2 * s_p)$ for maximum discrete s_p value.

Under ideal conditions (i.e., when a classifier rule is rewarded any time it matches the environment and wins the competition) the steady state strength and bid values are calculated as follows (as derived in section 6.4.2):

$$S_{ss} = \frac{R_{ss}}{C_{bid} \cdot f\left(s_p\right) + C_{tax}}$$

$$B_{ss} = \frac{C_{bid} \cdot f\left(s_p\right) \cdot R_{ss}}{C_{bid} \cdot f\left(s_p\right) + C_{tax}}$$

The strength values of perfect and specific rules, contained in the two alternative default hierarchies (as shown in table 6.4 below), reached their theoretical steady state values after approximately 1000 iterations according to above equations. But, the general default rules were found to vary between two strength values as shown in table 6.4.

Default hierarchy (1) :			**Default hierarchy (2) :**		
Rule	Strength		Rule	Strength	
1) ### 00# → 0	16·67		5) ### 1## → 1	21·05	
2) 01# 010 → 0	10·25		6) 00# 010 → 1	10·25	
3) 1## 01# → 0	13·79		7) 0## 011 → 1	11·76	
small-influence default rule :			**strong-influence default rule :**		
Rule	Min-Strength	Max-Strength	Rule	Min-Strength	Max-Strength
4) ### ### → 1	21·79	25·6	8) ### ### → 0	16·7	23·34
Bad Rule :			**Bad Rule :**		
Rule	Strength		Rule	Strength	
01# 010 → 1	0·0		00# 010 → 0	0·0	

Table 6.4 Results of the default hierarchy experiment

As explained in section (6.4.2), in the standard version of the bucket brigade algorithm, used extensively by current classifier systems, we have:

$$S_i(t+1)=S_i(t)-C_{bid}\cdot S_i(t)-C_{tax}\cdot S_i(t)+R(t)$$

We can rewrite the above equation as:

$$S_i(t+1)=(1-k)\cdot S_i(t)+R(t) \qquad (6.4.5.3)$$

Where:

$$k=C_{bid}+C_{tax}$$

Equation (6.4.5.3) represents a linear discrete system. The bucket brigade algorithm, as represented by equation (6.4.5.3), can be written as:

$$S_i(t)=\beta\cdot u(t)+\alpha\cdot S_i(t-1) \qquad (6.4.5.4)$$

Where, in equation 6.4.5.4, we have:

$$\alpha = 1 - k$$
$$\beta \cdot u(t) = R(t)$$

Taking Z-transform from both sides of equation (6.4.5.4), and dropping the index i, we get:

$$Z[S(t)] = \beta \cdot Z[u(t)] + \alpha \cdot Z[S(t-1)]$$

$$Y(Z) = \left(\frac{\beta \cdot Z}{Z - \alpha}\right) \cdot U(Z) \qquad (6.4.5.5)$$

Where:

$$Z[S(t)] = Y(Z) \; and \; Z[u(t)] = U(Z)$$

Equation (6.4.5.5) enables us to determine the expected steady state strength response value of a classifier rule that remains activated resulting in a constant α. We have already mentioned that, our hand-crafted default rules are found to vary between two strength values (table 6.4). This behaviour, in general, is caused by two factors:

1- A default rule might be rewarded only a percentage of the time; the default rule number 4 (table 6.4), is only rewarded approximately 59% of the time and the general default rule number 8 (table 6.4), is only rewarded 41% of the time.

2- Even if a particular default rule is correct (or any less than perfect competing classifier rule) it might lose the competition (losing reward).

In above cases, the expected input reward signal can be approximated as a periodic intermittent signal. The general form of such a signal is:

$$U(Z) = \left(\frac{Z^N}{Z^N - 1}\right) F_1(Z) \qquad (6.4.5.6)$$

Where, the first periodic sequence, represented by $U(Z)$, is characterized by:

$$F_1(Z) = \sum_{k=0}^{N-1} f(k) \cdot Z^{-k} \quad ,$$

and, for the periodic reward signal, we have: $f(k) = f(k+N)$

N represents the period of the input reward sequence $f(k)$. In our case, the expected periodic reward sequence can be represented by a periodic sequence of kronecker delta functions with period N. We have:

$$F_1(Z) = \sum_{k=0}^{N-1} \delta(Z) \cdot Z^{-k} = 1$$

The overall strength response of each classifier rule, under intermittent input reward, using equation (6.4.5.6), becomes:

$$\begin{aligned} Y(Z) &= \left(\frac{\beta \cdot Z}{Z-\alpha}\right)\left(\frac{Z^N}{Z^N-1}\right)F_1(Z) \\ &= \left(\frac{\beta \cdot Z}{Z-\alpha}\right)\left(\frac{Z^N}{Z^N-1}\right) \end{aligned} \qquad (6.4.5.7)$$

The output response, as represented by equation (6.4.5.7), can be represented in its pole-zero form as follows:

$$Y(Z) = \frac{\beta \cdot Z^{N+1}}{(Z-\alpha)\left(Z-e^{\left(2\pi\, k/N\right)j}\right)} \quad k = 1, 2, \cdots N$$

Using partial fraction expansion, the general form of the time domain output response, as t approaches infinity, becomes:

$$f(t) = \sum_{k=0}^{N-1} \alpha_k \left(e^{j\left(2\pi k/N\right)t}\right) \qquad (6.4.5.8)$$

$k = 0, 1, 2, \ldots, N-1$

$t = 0, 1, 2, \ldots$.

Knowing that, the time domain solution (as represented by equation (6.4.5.8)) occurs in complex conjugate pairs, by using Euler's identities or otherwise, we note that the overall output response is periodic with a minimum and a maximum steady state strength values. However, the response remains stable, even with switching non-linearity resulting from interactions with noisy environments, as long as the changing α meets the stability criterion. In such cases, the apportionment of credit algorithm maintains the asymptotic strength value as the long term mean value of environmental reward amplified by $(1/\alpha - 1)$ coefficient.

6.4.6 Learning from the design environment

In this section, we examine the performance of the classifier system starting from a randomly generated population of rules. Initially, we perform two simulations. One without the genetic algorithm enabled and one with the genetic algorithm enabled. In this way, we are able to separate the learning due to apportionment of credit among the original rules and that due to the injection of new rules by the genetic algorithm.

In these initial tests, we start the classifier system from 40 rules with randomly generated conditions. Half of the actions of the rule population are set to support the strong influence design cases and half are set to support the small influence design cases.

The probability of generating a don't care token (i.e., wild card (#)) is set to 0·67. In this way a don't care symbol # is selected with probability 0·67 while a 0 or 1 is selected with probability (1-0·67)/2. Knowing the form of target set of rules for this learning task, these values provide an average density of "#" tokens (in the initial randomly generated population), which corresponds to an average number of "#"s in the target rule sets.

In the first run, the initial randomly generated population of rules is subjected to the apportionment of credit algorithm without the genetic algorithm activated. The implemented apportionment of credit algorithm for our first simulation is governed by the following difference equation:

$$S_i(t+1) = S_i(t) - C_{bid} \cdot S_i(t) - C_{bid-tax} \cdot S_i(t) + R_i(t) \qquad (6.4.6.1)$$

As before, C_{bid} is the bid paid by a classifier rule. In order to promote default hierarchy formation, the bid of a classifier is formulated as a linear function of the classifier's strength and its specificity. I.e.:

$$B_i(t) = C_{bid} \cdot f\left(s_p\right) \cdot S_i(t) \qquad (6.4.6.2)$$

Where:

$$f\left(s_p\right) = M_1 + M_2 * s_p \qquad (6.4.6.3)$$

s_p is the specificity of a classifier rule as defined before (i.e., equation 6.4.5.1). Using equation (6.4.6.3) different bidding structures may be investigated. In the following simulations, the parameters, for our apportionment of credit algorithm, are given in the following table:

C_{bid}	0·1
$C_{bid-tax}$	0·01
$R(t)$	1·0
M_1	0·25
M_2	0·125

Table 6.5 Learning parameters for the Apportionment of Credit-Algorithm

The M_1 and M_2 parameters have been set so that, the bid of a completely general rule (i.e., $s_p = 0$) will be only : (0·25 × C_{bid} × Strength), but the bid of a completely specific rule (i.e., $s_p = 6$) will be (C_{bid} × Strength).

The run (simulation-1), without genetic algorithm activation, up to and including generation 40,000, is shown in figure 6.3. Here we note how the apportionment of credit algorithm adjusts the strength values of the rules achieving a steady time averaged performance of 82% correct. The irregular and jagged curve shows the average score of the system in terms of percentage of correct answers over the last 50 running trials. Although, the 50 step average score moves above and below the steady overall time averaged performance, in the long run the classifier system is able to sustain its relative high performance. In this way, rules containing relevant schemata to the learning task have reached theirs expected asymptotic strength values, as governed by the apportionment of credit algorithm, and irrelevant rules have lost their strength in the competition.

We would not expect to achieve better performance, in the long run, under the application of the apportionment of credit algorithm alone. This is due to the fact that, in a population of 40 randomly generated rules, there is a low probability of producing a large number of relevant schemata.

For example, we can calculate the probability of randomly generating the

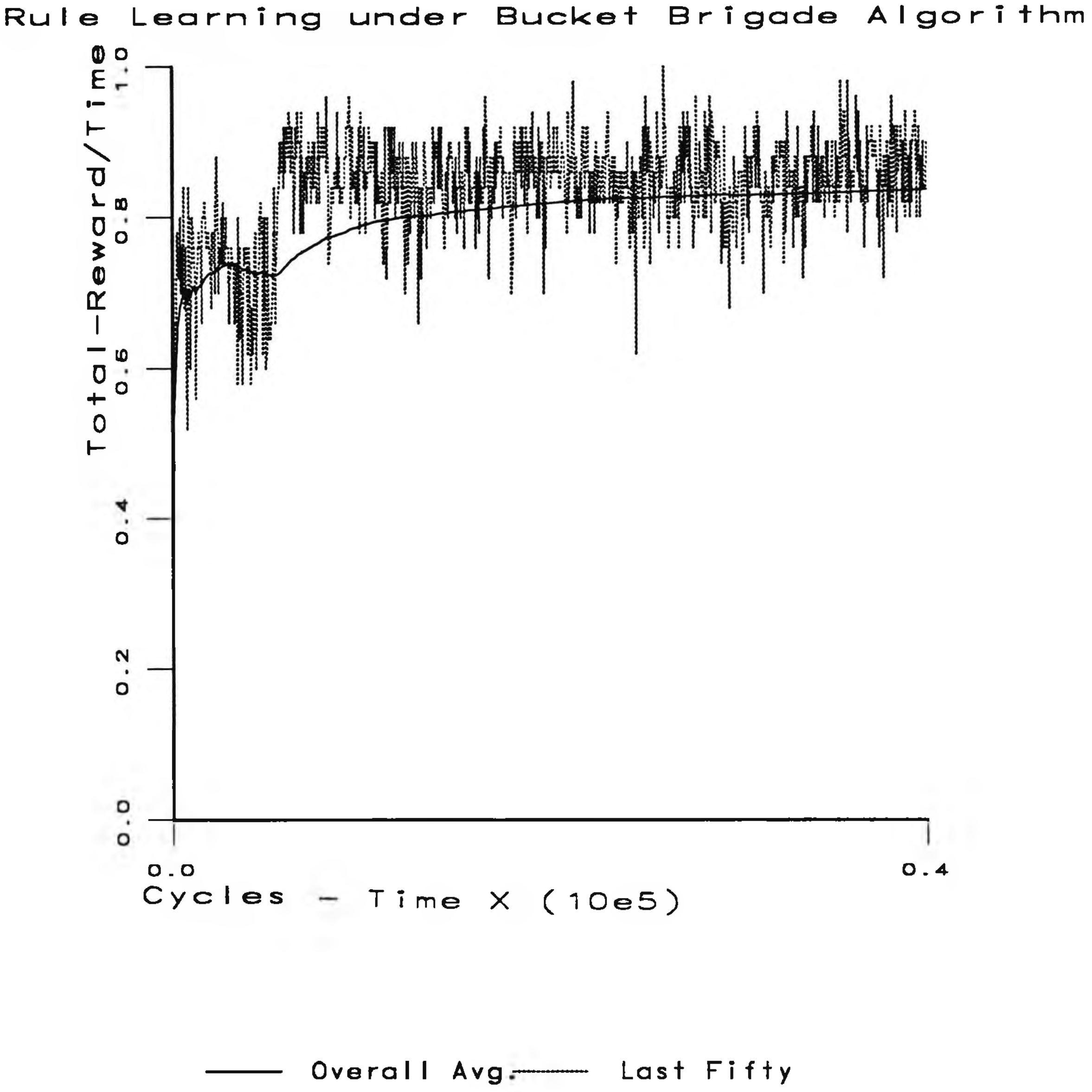

Rule Learning under Bucket Brigade Algorithm
Total-Reward/Time
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.4
Cycles - Time X (10e5)
Overall Avg.
Last Fifty

Figure 6.3

following two relevant rules:

$$C1:\ 01\#\ 010 \rightarrow 0$$
$$C2:\ 00\#\ 010 \rightarrow 1$$

The probability of a randomly generated population containing two relevant rules, in general, is derived as:

$$P_{C1,C2}(s)=\sum_{n=1}^{s-1}\left\{\binom{s}{n}p_s^n\cdot p_f^{s-n}\times\left(\sum_{k=1}^{s-n}\binom{s-n}{k}p_s'^k\cdot p_f'^{s-n-k}\right)\right\}$$
$$+\sum_{n=1}^{s-1}\left\{\binom{s}{n}p_s'^n\cdot p_f'^{s-n}\times\left(\sum_{k=1}^{s-n}\binom{s-n}{k}p_s^k\cdot p_f^{s-n-k}\right)\right\} \qquad (6.4.6.4)$$

Where:

s : population size

p_s : probability of $C1$ occurring in a single trial

p_f : probability of $C1$ not occurring in a single trial

p_s' : probability of $C2$ occurring in a single trial

p_f' : probability of $C2$ not occurring in a single trial

Noting that, in our case $p_f = p_f'$, and using binomial theorem, we simplify the above equation to the following expression:

$$P_{C1,C2}(s)=2{\cdot}0\times\left\{\left(1-p_f^s+p_f^{2s}\right)-\left(p_s+p_f^2\right)^s\right\}$$

Considering our population size of 40, and noting that we have set the probability of generating wild card (#) tokens to 0·67, the probability of randomly generating our two relevant rules, in our case, becomes:

$$p_{c_1,c_2}(40) = 2\times 2{\cdot}59\times 10^{-6}\approx 5{\cdot}18\times 10^{-6}$$

Therefore, the probability of randomly generating even two relevant rules, for our design learning task, is almost negligible. By further expansion of equation of (6.4.6.4), we note that the probability of randomly generating a default hierarchy (such as the ones shown in table 6.2) is nearly zero for any practical purpose. These calculations confirm our expectation that, by using the apportionment of credit algorithm alone (as given by equation (6.4.6.1)), we can not expect to achieve better performance and we need an inductive rule discovery mechanism (such as a genetic algorithm) to boost performance.

The apportionment of credit algorithm, as specified by equation (6.4.6.1), is based on a deterministic selection of the highest bidders. This means that, at each matching cycle, the highest bidder is deterministically selected for activation. However, the degree of determinism in the decision process is important and relates to the "exploration" versus "exploitation" trade-off faced by all inductive systems.

If for two matching classifiers A and B, the strength of A is somewhat greater than that of B, then a relatively deterministic decision (for A) should be made only if the system is quite sure that the strengths estimate accurately the resulting reward. This corresponds to exploitation of current information. On the other hand, if the system is unsure that the estimates are accurate, then the decision should be made less deterministically, resulting in more exploration of alternatives.

The main reason for this non-determinism occurs when we inject new rules by the genetic algorithm at the average strength of their parents. As a result, if these rules are ever to have a possibility of trial, their bid values must be occasionally lifted above the better rules.

In order to achieve this exploration versus exploitation trade-off in the decision process, most classifier systems have made decisions by a stochastic process in which the probability of selection was proportional to the bid of a matching classifier. In the bidding competition, the probability of selecting a particular classifier for activation is equal to its bid divided by the sum of bids of all matching classifiers. This process is similar to the Monte Carlo selection strategy (DeJong, 1975). The second technique has been suggested by Goldberg (1983). This technique is based on a "noisy auction" in which to each bid of a matching classifier a gaussian noise having a certain variance is added, and the classifier with the largest of the resulting bids (called its effective bid) is selected for activation.

An advantage of the noisy auction is that, by adjusting the noise variance, the degree of determinism in the decision can range from completely deterministic (i.e., based directly on the highest bids) to totally random.

In the following simulations, the second approach has been used. I.e., the effective bid of each classifier is calculated as the sum of its deterministic bid and a noise generator:

$$EB_i = B_i + N_i(\sigma_{bid}) \qquad (6.4.6.5)$$

Where:

EB : effective bid
B : actual bid
N_i : gaussian noise generator
σ_{bid} : standard deviation of the noise generator

For this study, the noise generator is implemented using the Box-Muller method for generating random deviates with a normal (gaussian) distribution (Press, 1988). The amount of gaussian noise imposed on an actual bid has been set to 0·075. Goldberg (1983) has observed that the introduction of even the smallest amount of randomness, in the decision process, results in a significant improvement in exploration of new alternatives. Consequently the chosen setting for C_{bid} achieves a more deterministically biased decision process. This procedure is used in conjunction with genetic algorithm activated simulations.

During the next simulations, we will be using the genetic algorithm regularly to inject new rules into the population at possibly relative high levels of strength (average fitness of parents), therefore, we must ensure that inactive rules are degraded sufficiently before they can reproduce and insert new rules themselves. If this is not done, relatively inactive rules can obtain high levels of strength and reach reproduction. This causes degradation of performance, because the genetic algorithm might replace relevant and useful rules with irrelevant and weak rules.

In order to avoid this problem, we modify our apportionment of credit-algorithm as follows:

$$S_i(t+1)=S_i(t)-C_{bid}\cdot S_i(t)-C_{bid-tax}\cdot S_i(t)-C_{life-tax}\cdot S_i(t)+R_i(t) \qquad (6.4.6.6)$$

$C_{life-tax}$ in the above equation, represents the head-tax (Holland, 1987; Riolo, 1987), which is a linear decay term. The decay term ensures that a useful

and above average classifier rule will reach its expected asymptotic strength value after a number of time steps which is in the order of hundreds. It also reduces the below average rules to theirs estimated zero steady state strengths.

A genetic algorithmic activation of 500 has been used in the initial test runs. $C_{life-tax}$ is calculated for this genetic activation period by considering the apportionment of credit algorithm as represented by equation (6.4.6.6). Since, the apportionment of credit algorithm simply reduces the strength of an inactive rule by the $C_{life-tax}$ rate, after n iterations of inactivity, we have the following value of strength :

$$S(t+n)=\left(1-C_{life-tax}\right)^n\cdot S(t) \qquad (6.4.6.7)$$

Therefore, the half-life of an inactive rule is given by :

$$n=\log(½)/\log\left(1-C_{life-tax}\right) \qquad (6.4.6.8)$$

We want to make the strength of an inactive rule to be near its half-life free fall value after 500 time steps (i.e., before the genetic algorithm is activated). A value of 0·001 has been chosen for $C_{life-tax}$. This gives a free half-life value of about 692 for below average rules. Other apportionment of credit learning parameters are as defined in table 6.5. As described in section 6.4.3, at each genetic activation, we only select a proportion of our population for selection and reproduction. This procedure is based on DeJong's crowding technique, in which we want to create separate rule niches by replacing below average rules with similar and possibly better rules.

DeJong (1975) has used this scheme successfully with a crowding factor CF=2 and 3 and with a generation gap G=0·1 on a multimodal function optimization task. On the basis of his observation, we will select only 5% of the classifier rule population at each genetic activation. Therefore, in our case, at each genetic algorithm invocation we will inject only two new trials. Also, we have chosen a crowding factor of 3, and the cross-over and mutation probabilities have been set to 1·0 and 0·01 respectively. In this way, we maintain a high level of on-line performance while exploring possibly better alternative design rules.

We summarize our learning parameter values in the following table:

AOC Parameter	Value	GA Parameter	Value
C_{bid}	0·1	$GA\ Period$	500
$C_{bid-tax}$	0·01	P_c	1·0
$C_{life-tax}$	0·001	P_m	0·01
σ_{bid}	0·075	G	5·0%
M_1	0·25	CF	3
M_2	0·125		

Table 6.6 Classifier system learning parameters

Using the above settings, three independent runs were performed, up to and including generation 40,000. The best of run (simulation-2) performance curve is shown in figure 6.4. We note that, our simulation, with genetic algorithm invoked, significantly outperforms our previous runs without genetic activation. In effect, the classifier system has learned rules sufficient to perform correctly 95-96 percentage of the time.

Contrasting this simulation with the previous one, we see that by exploring new rules, using the genetic algorithm, we can obtain a much better performance. This is achieved by our two inductive algorithms. The apportionment of credit algorithm rates extant rules and decides among competitors, while the genetic algorithm contributes new and possibly better rules into the rule population.

Figure 6.5 gives a comparison between our two simulations based on the time averaged performance.

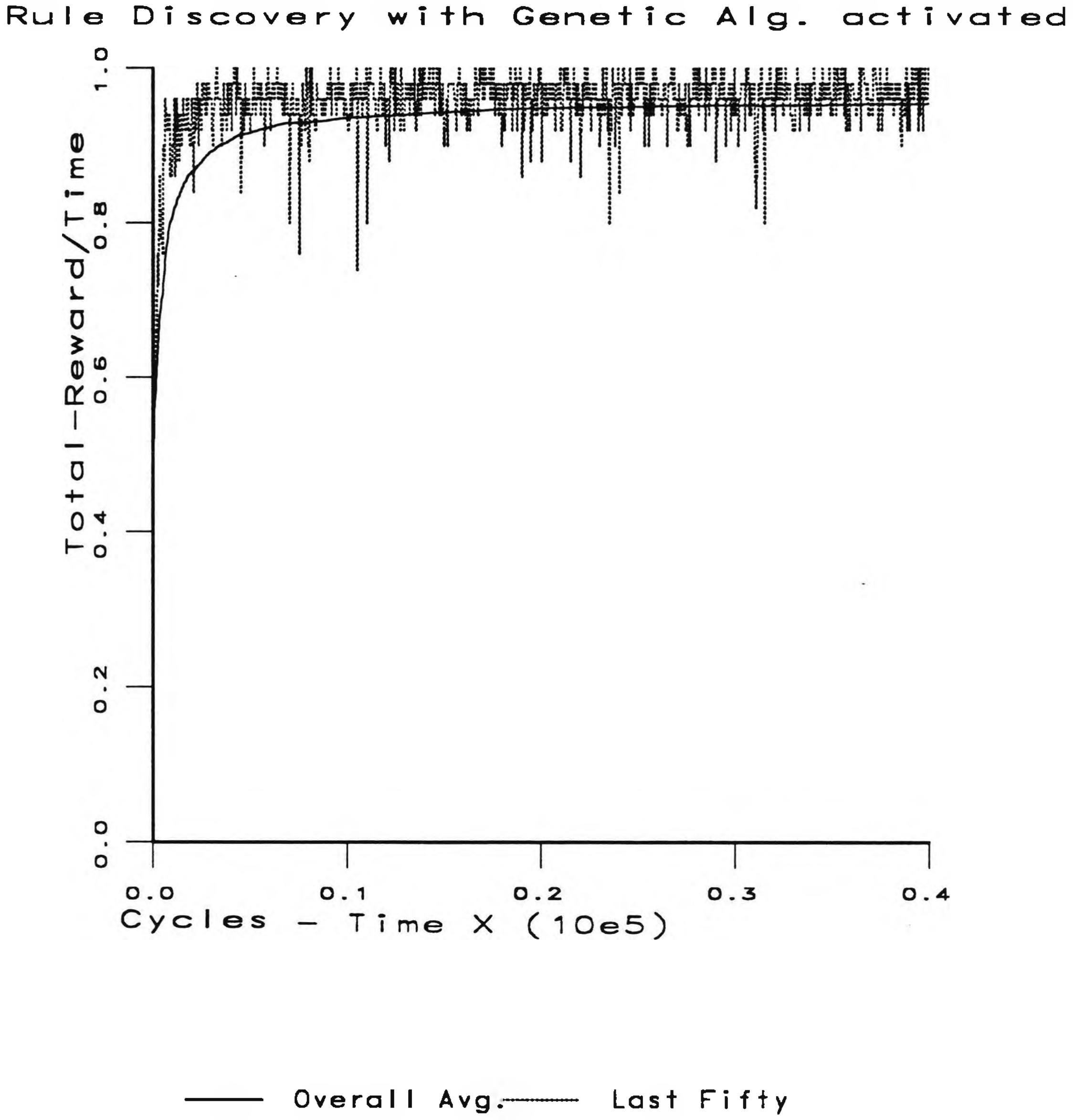

Rule Discovery with Genetic Alg. activated
Total-Reward/Time
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.1
0.2
0.3
0.4
Cycles - Time X (10e5)
Overall Avg.
Last Fifty

Figure 6.4

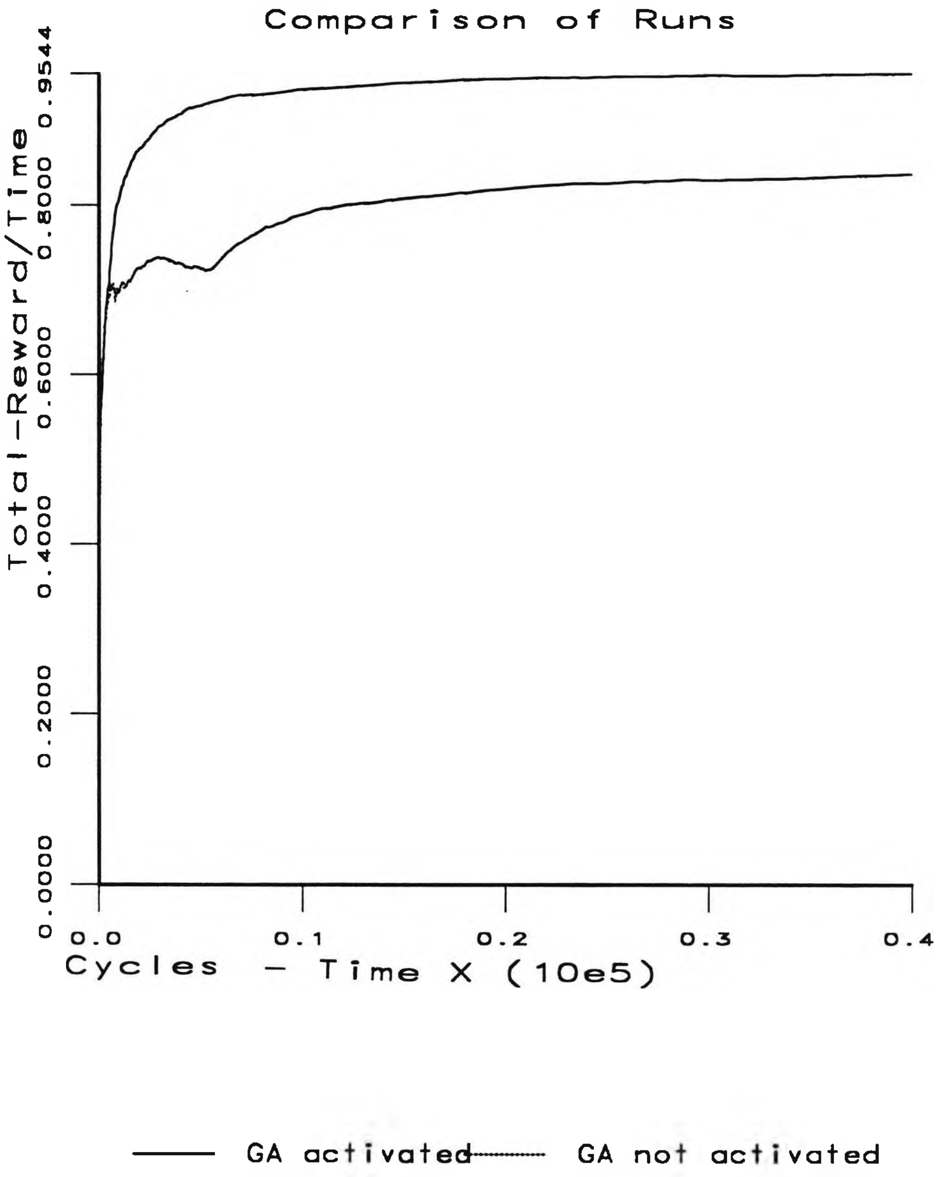

Comparison of Runs
Total-Reward/Time
0.0000
0.2000
0.4000
0.6000
0.8000
0.9544
0.0
0.1
0.2
0.3
0.4
Cycles - Time X (10e5)
GA activated
GA not activated

Figure 6.5

To get a better understanding of the type of rule learning performed by the classifier system, we examine the above average rules selected and created at the generation 40,000. These are shown in table 6.7.

small-influence design rules :		**strong-influence design rules :**	
Rule	Strength	Rule	Strength
C_{S1} : ### 00# → 0	15·49	C_{B1} : ### 10# → 1	15·42
C_{S2} : 1## 01# → 0	12·86	C_{B2} : ### 11# → 1	15·87
		C_{B3} : ### 01# → 1	7·72
Population Average		1·97	

Table 6.7 Above average rules created during simulation-2

It is remarkable that the classifier system has created a nearly perfect default hierarchy. The created default hierarchy set of co-operating rules is more intuitive than our "natural" solutions that partitioned the design space into six non-overlapping regions. The suggested default is more compact and is based on the internal dynamics of the classifier system. As shown in table 6.7, the population average is 1·97, indicating that the remaining 35 below average rules, have been rigorously reduced to their approximate half-life free fall values.

Rules C_{S1}, C_{S2}, C_{B1} and C_{B2} have nearly reached theirs expected asymptotic steady state strength values, as indicated by equation (6.4.2.8). The lower values of their strengths, as compared to the theoretical values indicated by equation (6.4.2.8) is due to the addition of the decay term.

The default rule C_{B3} has a much lower strength as could be predicted by a direct use of equation (6.4.2.8). Considering the following two of our created co-operative rules:

C_{B3} : ### 01# → 1
C_{S2} : 1## 01# → 0

Rule C_{S2} is a more specific rule, covering some of the mistakes made by the more general default rule C_{B3}. C_{S2} is always correct, and its strength is near its expected asymptotic theoretical value. C_{B3} would have had an expected fixed point strength value, equivalent to those of C_{B1}, C_{B2}, and C_{S1} if it was always correct. But, its asymptotic strength value is much lower than expected, because, it is wrong in 10 design cases for which it bids but receives no reward. This is in fact what we expect; under the apportionment of credit algorithm and by using specificity dependent bidding (equation 6.4.6.2), we have achieved a stable default hierarchy in which 8 mistakes of our more general rule C_{B3} are covered by C_{S2}. Under these conditions, the asymptotic strength values ensure that if during a matching cycle both C_{B3} and C_{S2} are active, our more specific rule C_{S2} will always win due to its higher bid value. Therefore, the created default hierarchy is stable; enabling the classifier system to perform 96·8% of the time correctly.

Unfortunately, our default hierarchy of rules is not perfect. I.e., 2 further mistakes made by C_{B3} are not covered by any rule in the rule population. To achieve a perfect rule set, the system must create a highly specific rule such as (01# 010 → 0) to cover these mistakes.

One of the reasons for this problem is that, in the initial randomly generated population of 40, only few relevant schemata for construction of this rule exist. We remember that, we have chosen a probability of 0·67 for generating # tokens. This decision was taken to promote the formation of more general and relevant rules, because our hypothetical default hierarchies, such as the ones shown in table 6.3, are dominated by highly general rules. Under this scheme, the probability of generating a relevant specific rule, such as the one we require, in a single trial is only $4{\cdot}09 \times 10^{-5}$.

One way to rectify this situation is to use a higher population size. Using a higher population size ensures a higher probability of generating larger number of relevant schemata for our design learning task. Also, a higher rate of mutation per genetic algorithm activation ensures the insertion of a higher number of new schemata not already contained in the population.

The classifier system might still encounter problems, even if it generates a relevant, useful and specific design rule. More specific rules are rewarded less often than more general rules, and might reach theirs expected asymptotic strength values more slowly. Considering our non-deterministic decision process and severe competition among active rules, some of which having unrealistically high levels of strength due to genetic activation, we must be very careful about the size of the $C_{life-tax}$ parameter. A high value of head-tax can result in unjustifiably low half-life values. Specifically, we consider the expected asymptotic strength value, under the current scheme, for a rule such as : (01# 010 → 0).

Assuming a head-tax value of zero, this rule will reach its expected asymptotic value of 10·25 according to equation (6.4.2.8). On average, this rule is expected to be active twice per 64 cycles, or once per 32 cycles. This observation is based on the stochastic nature of our design environment, where the probability of each design situation being faced by the classifier system is 1/64.

We restate the apportionment of credit algorithm, expressed by the following difference equation:

$$S_i(t+1)=S_i(t)-C_{bid}\cdot S_i(t)-C_{bid-tax}\cdot S_i(t)-C_{life-tax}\cdot S_i(t)+R_i(t)$$

In the case of a nominal periodic input with period *T*, the expected asymptotic strength value of a classifier is obtained by imposing :

$$\begin{array}{c} S(t+T)=S(t) \\ t\rightarrow\infty \end{array} \qquad (6.4.6.9)$$

The starting point t is irrelevant in this computation. We assume that t is the time-step right after activation and reception of the reward. Therefore, since the expected period between two activations of the classifier is T, we have an expected strength decay for T-1 time steps during which head-tax is paid and then one active time-step, when the classifier rule receives reward and pays bid-tax and head-tax.

Applying the condition of periodicity and solving for $s(t)$, using our apportionment of credit difference equation, we get :

$$S(t)_{t\rightarrow\infty}=S(t+T)=\frac{R_{ss}}{1-\left(1-C_{life-tax}-C_{bid-tax}\right)\left(1-C_{life-tax}\right)^{T-1}+C_{bid}\left(1-C_{life-tax}\right)^{T-1}}$$

(6.4.6.9)

Equation (6.4.6.9) gives the maximum expected asymptotic strength value attainable by a classifier rule. Under our scheme (i.e., $C_{life-tax}$ = 0·001), by using the above equation, the expected asymptotic strength value for our specific required rule is (we have s_p = 5 for this rule) approximately 7·9 units.

From above calculation we note that, the expected maximum fixed point value of strength, under our scheme, has decreased by 2·35 units. Therefore, to promote the formation of more specific rules, we are justified to decrease the head-tax value by half. This gives an expected maximum fixed point strength value of approximately 9·0 units for our specific desirable rule. Considering our new head-tax setting, we double the period of genetic algorithm activations according to equation (6.4.6.8).

Following the above discussion and analysis, we change our population size, mutation rate, head-tax, and genetic algorithm period as shown in table 6.8 below:

New Learning Parameters :	
Population Size	60
Mutation Rate	0·02
$C_{life-tax}$	0·0005
GA period	1000

Table 6.8 New learning parameters for simulation-3

Keeping the percentage of population replaced by the genetic algorithm fixed and, knowing that we have increased the population size, we will be inserting 3 rules per genetic activation.

We are now in a position to test our hypothesis. Three independent runs were performed, initiated with three different random seeds, using our new settings. The best of run (simulation-3) performance curve, up to and including generation 23000, is shown in figure 6.6. Here we note that, a steady time averaged performance of 0·973% has been achieved at iteration 23000 and this performance is still increasing. The 50 step time averaged score has reached a perfect 100·0% correct level at generation 2·1X10^4 and has retained its value up to and including generation 2·3X10^4. Therefore, the classifier system has performed correctly for 2000 cycles.

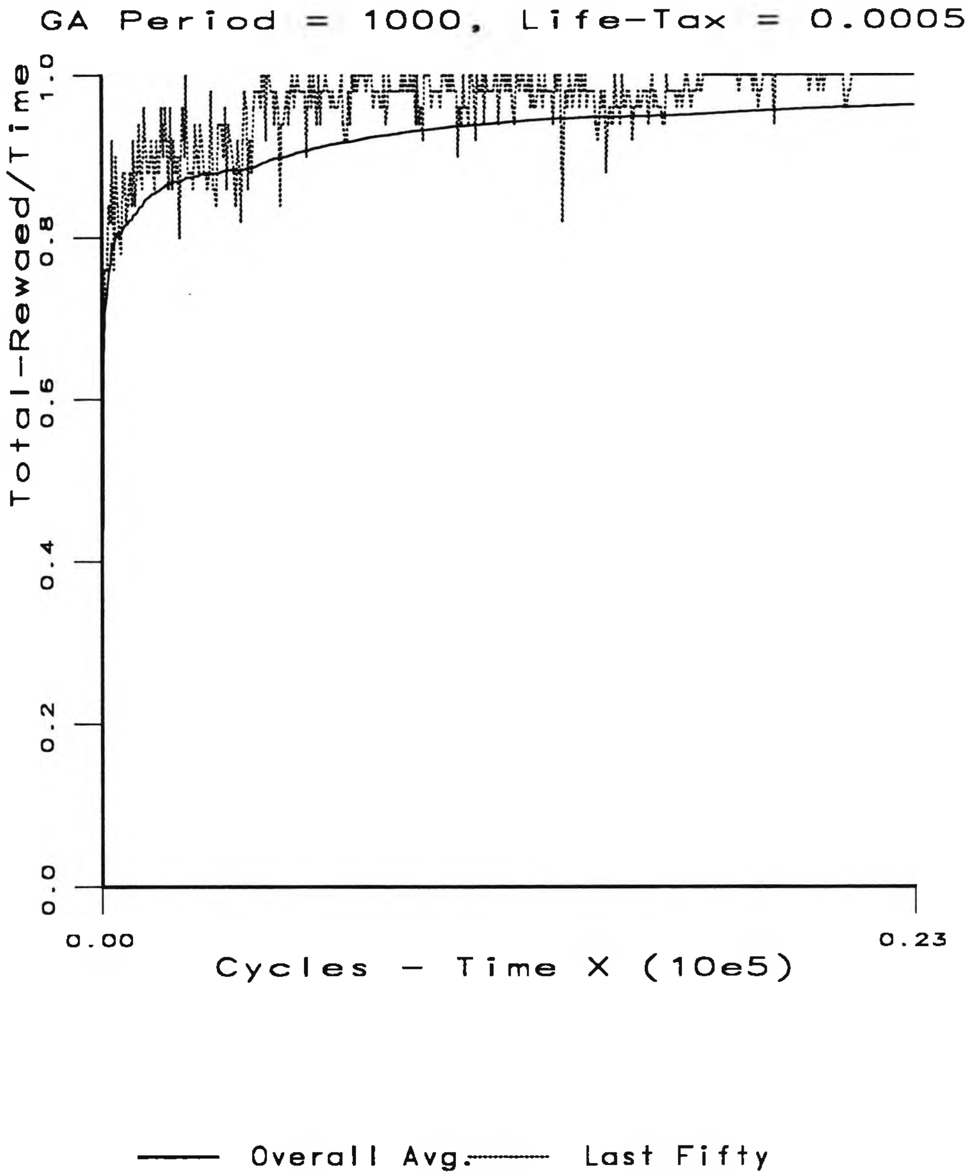


Figure 6.6 Rule learning, using the new learning parameters

By inspecting the rule population at generation 23,000, we observe that, the classifier system has created a set of above average co-operating rules achieving a perfect performance. These rules are shown in table 6.9.

small-influence design rules :		**strong-influence design rules :**	
Rule :	Strength :	Rule :	Strength :
C_{S1} : 1## 00# → 0	11·12	C_{B1} : ### 10# → 1	16·0
C_{S2} : 1## 01# → 0	13·26	C_{B2} : ### 11# → 1	16·0
C_{S3} : #00 00# → 0	10·54	C_{B3} : ### 01# → 1	9·6
C_{S4} : 01# 00# → 0	9·71		
C_{S5} : 01# 00# → 0	10·14		
C_{S6} : 01# 010 → 0	8·73		
Population Average		2·47	

table 6.9 Above average rules created during simulation-3

As seen from this table, the specific rule that we have been hoping to find, has been discovered by the classifier system. Rules C_{S2} and C_{S6} co-operatively cover all mistakes of the default rule C_{B3}. Therefore, the classifier system has created and learned effective design heuristics for perfect performance. These results confirm our expectation that, classifier systems are highly capable inductive systems for our purposes.

6.5 Conclusions and future work

In this chapter, we have investigated the application of classifiers to the design of instrument transducers. In our application, the implemented classifier system has been able to discover design heuristics successfully by direct interactions with the specified mathematical model representing its environment. In our particular empirical rule discovery and learning task, environmental messages relate to different diaphragm design problems. Therefore, different environmental messages, lead the classifier system to form different categories of design problems, for which different design actions are necessary.

In a number of simulations, the classifier system, starting from no appropriate knowledge of the specified design environment, has gradually created a set of useful design heuristics, postulated as if/then statements. The discovered design heuristics are in the form of default hierarchies. In the created default hierarchies, fairly general rules cover the most frequent design cases and more specific rules (that typically contradict the default rules) cover exceptions. However, as seen from table 6.9, rules C_{s1}, C_{s3}, C_{s4}, and C_{s5} together perform the same function as that of C_{s1} in table 6.7. That is, in this case the small influence design rules have a more specific nature. The collapse of the more general rule into a set of more specific rules is not desirable and has been observed in previous studies.

Holland (1986) has stated that default hierarchies should form in classifier systems if the specificity of a classifier is factored into its bid; this allows more specific exception classifiers to outbid, and "protect", general classifiers from making an error.

In most classifier systems, the bid has included specificity, yet default hierarchies have not appeared in abundance. Riolo (1987) showed, in experiments not using the genetic algorithm, that default hierarchies could be stable, once formed. The implication is that something about the genetic algorithm inhibits default hierarchy formation.

Wilson (1988) hypothesized that the bid competition not only protected defaults when they were wrong, it also tended to starve them when they were right, so that defaults, if generated, could not survive. The starvation resulted because the genetic algorithm could be expected to generate, besides the default, more specific versions of the default, and these latter classifiers would outbid the default and prevent it from being activated.

Under our standard bidding scheme, a classifier pays out a fraction b of its strength S, so that the bid itself approaches $b{\cdot}S = b(R/b) = R$, where R is the long-term average income (after taxes). Under this arrangement, the expected asymptotic strength values of classifiers are such that a general rule and a specific rule active in the same situations will come to bid the same amount (because the strength of a general classifier increases to the point that it can compensate for its smaller specificity dependent bid value). Consequently, a more specific rule, despite its greater specificity, can not stably beat the more general rule.

Holland (1987) has suggested that, stability might result under a bidding process that more closely reflects that of auctions in real markets. In real market economies, individuals examine an item they wish to purchase, decide how much they can afford, and bid up to the point where the buyer's bid meets or exceeds other bids and the seller agrees to sell. In a classifier system, a classifier matches a message, it pays out a fixed percentage of its bank balance, if that amount happens to be selected in the auction process. To make the standard bucket brigades more similar to real markets, we can have the winner classifiers pay out only the amount necessary to beat their competitors. In one simply implemented scheme, each classifier simply broadcasts its standard bid value, a winner is selected, but the winner only pays out an amount equal to the bid made by the second best rule. Whether these or other schemes can encourage superior default hierarchy formation and result in their persistence and stability are a subject for further research.

Our implemented classifier system, for the design heuristic extraction, has produced a set of useful design heuristics which relate the nature of the influence of the centre-boss size, the effective radius and relative corrugation depth to the overall characteristics of corrugated diaphragms. The induced design rules are more accurate than Andreeva's because, they are based on a more accurate representation of the design environment. The produced design heuristics can constitute a part of an overall design methodology for the design of corrugated diaphragms. In order to extend this work, we must modify the design environment and construct an appropriate syntax for the classifier rules enabling the classifier system to induce a sequence of design heuristics, constituting a design methodology. The design heuristics are based on empirical relationships derived from set(s) of dimensionless curves. The aim of this extension is to simulate a number of techniques for the inductive extraction of design methodologies (Mirza, 1983).

CHAPTER 7

Conclusions

7.1 Conclusions

Our aim in this thesis has been clear. We have directed our efforts into the utilisation of DAI techniques for the purpose of design automation of instrument transducers.

We have argued that design problems, by nature, are parallel, distributed processes and are best implemented using DAI techniques. DAI systems have a multitude of advantages as compared to conventional rule-based systems. These include: naturalness, reliability, efficiency, resource sharing, extendibility and cost-effectiveness. However, complex research issues must be resolved when a distributed approach is used. These issues are classified into four categories:

1- Formulation, decomposition and allocation of problems
2- The methods for achieving distributed control
3- Communication processes for coordination
4- Modelling other agents

Unfortunately, there has been relatively little research in problem formulation, decomposition and task allocation issues. In most DAI systems, decomposition of problems into sub-problems is known before the initiation of the problem solving process. This has influenced current approaches to the design of DAI systems. In these systems, almost always, there exist a predefined specification of a static conceptual hierarchy of the design problem and the way to decompose a design problem is provided by the developer. These systems support a routine type of problem solving. Non-routine type of design problem solving requires the generation of an extensive number of possible decompositions into subfunctions providing alternative synthesis plans for the overall design process.

In order to resolve these problems, we have proposed and implemented a Distributed Problem Solver framework for conceptual design of instruments. The implemented system consists of a community of knowledge-based agents with expertise on design of instrument sub-systems. The control structure of the system is in the form of a community of cooperative design organizations. Each design organization represents an energy domain. The energy domain organizations are hierarchical in their architecture in which functional-agents are the central coordinators which give commands to their sub-system agent organization members for design purposes.

On top of the above mentioned control structure, we have designed a communication protocol to coordinate agent interactions for problem-decomposition and sub-problem distribution. The designed protocol is based on the Contract-Net style of negotiation and supports a task-sharing form of cooperation among the agents.

The contribution of this work has been to consider the problem formulation and decomposition as major design problems which can be resolved via negotiations among cooperative agents. In other words, in our DPS system, the problem decomposition processes, themselves, have been distributed and are achieved by negotiations using a task-sharing form of cooperation. These negotiations result in dynamic configuration of design organizations which represent alternative conceptual design solutions (supporting alternative synthesis plans).

Our approach is in contrast to conventional Contract-Net style of negotiations in which the problem decomposition knowledge is embedded in Manager agents and negotiations are only directed towards the dynamic distribution of sub-problems.

In order to avoid excessive negotiations and support learning capabilities, each agent has been provided with self-model and env-model qualifiers. The env-model attribute allows agents to incorporate detailed models of the behaviour and capabilities of their acquaintances and reason about their actions.

Another contribution of our proposed DPS system is its utilisation of the complementary nature of both forms of cooperation (i.e., task-sharing and result-sharing) as opposed to a conventional CNET framework (Smith[10]) which only supports task-sharing or a Hearsay-II (Lesser [19]) type of framework in which only result-sharing is supported.

In order to study the techniques by which the adaptive capabilities of our DPS system can be enhanced, we directed our efforts into the investigation of machine learning techniques for single knowledge-based agents and their possible extension into our multi-agent system. The integration of learning in our DPS system is of immense importance. Machine learning techniques such as case-based reasoning, or inductive learning should be an integral part of distributed problem solving. These techniques help the multi-agent system performance through acquiring new knowledge, refining existing knowledge, using better coordination strategies, or memorizing cases previously proven to be unsuccessful. Therefore, incorporation of learning processes in our multi-agent system will enhance its problem-solving by enabling it to exploit the complementary role of group learning during the problem solving activity.

The result of our critical investigations of these issues has been to propose a theoretical DPS framework which can be considered as an extension of contractual form of cooperation into fine-grained level of processing (i.e. single agent arena). In this framework, an agent is considered as a Classifier System module. Classifier-systems (a fine-grained DAI system), span both syntactic and connectionist approaches. This computational framework (representing a cognitive system) incorporates the semantic aspects of induction by being environment-oriented in its problem-solving. The major inductive mechanism in classifier systems, for accumulation of experience, is the Bucket Brigade algorithm (which supports a task-sharing form of cooperation). Furthermore, classifier systems use genetic operators to recombine the genetic characteristics of well performing rules to produce plausible better rules.

The proposed system, also, uses a contractual form of cooperation (also used for our implemented DPS system) at its coarse-grained level of processing by using the Bucket-Brigade algorithm for accumulation of experience. The Bucket-Brigade algorithm introduces an element of competition into the cooperetive process, i.e., a knowledge-based agent that successfully bids for a task receives a reward which positively affects its ability to bid for future tasks. Agents not successful in the bidding process are eventually degraded. Crossover and mutation operators can also be used at this level of processing resulting in the generation of new agents which inherit the abilities of the successful agents. An advantage of this proposal is that, it is relatively easier to apply crossover and mutation operators to classifier systems at their coarse-level of processing, as compared to a syntactic or a connectionist knowledge-based agent.

In order to investigate the capabilities of classifier systems, we concentrated on the practical uses of Classifier Systems and Genetic Algorithms as applied to the design of instruments at the sub-component level.

Genetic algorithms are highly adaptive processes which can efficiently search environments characterized as discontinuous, vastly multimodal and noisy, without auxiliary information requirement. However, genetic algorithms occasionally suffer from "premature convergence". The primary cause of this behaviour is identified as "genetic drift" that characterizes the stochastic errors caused by reproductive strategies. A secondary cause of this behaviour might be caused by genetic algorithm hard problems. These issues were investigated in the context of the design optimization of two instrument sub-systems (i.e., corrugated diaphragms and LVDTs), using a three operator genetic algorithm. In our applications, we were able to find optimal values, for design parameters, satisfying the design requirements.

In our optimization problems, an stochastic remainder without replacement selection, in conjunction with elitisism and an scaling process (St-El-Wn), gave relatively superior, near optimal results as compared to stochastic remainder selection strategy alone. Although St-El-Wn can reduce the stochastic errors due to similarity of fitnesses near convergence, it is not effective in maintaining diversity across the gene pool, i.e., a chromosome with substantially higher fitness, will tend to dominate the population, causing premature convergence.

In order to investigate a different selection strategy, a ranking selection has been implemented. The result obtained from our ranking selection strategy, in conjunction with elitisism, are substantially better as compared to previous selection methods used. This comparison is based on a best of run performance metric. In both of our design optimization formulations, optimal designs have been identified by this selection strategy. The ranking selection, completely solves the scaling problem and provides a consistent means of controlling offspring allocation. In general, ranking methods provide an even, controllable pressure to push for the selection of better individuals.

In the context of our design optimization problems, we have also considered a number of sharing schemes for multi-modal design optimization of instruments. The sharing scheme, together with a proportionate selection strategy, is an effort to maintain a pressure to balance the sub-population

sizes around each peak; by making sure that strings are reproduced according to shared fitness values. We have proposed a normalized formulation in conjunction with sharing schemes. In this proposal, the formulated normalized phenotypic values become independent of a particular design search space. Our new normalized sharing function takes into account the similarity of all independent design variables evenly and in a straight-forward linear function of the degree of similarity among them. Alternative optimal designs have been found, for both of our design optimization problems (satisfying more relaxed user specified design criteria), by using the sharing schemes, together with the stochastic remainder without replacement selection strategy. This confirms that, for more relaxed design specifications, there will be alternative optimal designs in both of our design optimization problems.

In chapter 6, we detailed a number of important proposals for the application of classifier systems to the development of methodologies for the design automation of instruments. In particular, we worked on the process of design heuristic extraction for corrugated diaphragms, using a set of dimensionless curves. The dimensionless formulation is based on a lumped parameter mathematical model.

In our application, the implemented classifier system has been able to discover design heuristics successfully by direct interactions with the specified mathematical model representing its environment. In our particular empirical rule discovery and learning task, environmental messages relate to different diaphragm design problems. Therefore, different environmental messages, lead the classifier system to form different categories of design problems for which different design actions are necessary. This is achieved through the establishment of a set of design rules, covering the entire problem range, that relate the geometric features of a diaphragm type (i.e., the relative radius ratio of the Centre Boss, the diaphragm height and its thickness) to its performance.

In a number of simulations, using the standard Bucket Brigade Algorithm, the classifier system, starting from no appropriate knowledge of the specified design environment, has gradually created a set of useful design heuristics postulated as if/then statements. The discovered design heuristics are in the form of default hierarchies. In the created default hierarchies, fairly general rules cover the most frequent design cases and more specific rules (that typically contradict the default rules) cover exceptions.

7.2 Future Work

In this thesis our main efforts have been the identification and implementation of appropriate DAI techniques for the design automation of instruments. In this direction, we have proposed and implemented a Distributed Problem Solver for the conceptual design of instruments, and investigated the application of classifier systems, as inductive knowledge-based agents, for the automation of inductive reasoning in the design process. We, also, have identified a multitude of research problems which have to be resolved. These issues are listed below and are left for future investigations:

The most important issue to be resolved in future expansions of our DPS system for conceptual design of instruments is the identification and implementation of appropriate conflict resolution strategies during the sub-problem solution and answer synthesis phases of the problem solving process. The conflict resolution strategies are needed, because in the instrument design domain, the number of alternative conceptual decompositions is high and there are no easy, formalizable heuristics to choose among them. In the implemented DPS system, a functional-agent must be able to select the best candidate sub-system Agents, using its expert knowledge and local model of the design environment (i.e., award tasks to the most suitable contractor sub-system agents within its organization). This choice depends on conflicting aspects of the design environment such as cost, weight, functionality, robustness, appearance, topology, ergonomic characteristics, performance and so on. Furthermore, during the subproblem solution phase and answer synthesis phase of the problem solving process, design conflicts must be resolved, because the sub-system solutions are highly dependent.

In general, when different agents give incompatible specifications for a given design component, or one agent has a negative critique of specifications asserted by another agent, we can say that a conflict has occurred and it must be resolved for coherent problem solving. Therefore, conflict resolution strategies represent an essential part of the cooperative design problem solving. These strategies are mostly induced by generalization of specific conflict solutions to produce domain-independent expertise suitable for a wide range of conflicts. Conflict resolution strategies represent a mainly unexplored but important next step in providing truly effective support for cooperative design.

Our next step is to incorporate learning techniques such as explanation-based learning, case-based reasoninq and/or inductive learning techniques into our DPS system resulting in a truly intelligent and adaptive system. For example, the heuristic nature of conflict resolution expertise necessitates the exploration of analogy and learning to enhance the coverage and effectiveness of existing conflict expertise.

We have proposed the use of classifier system modules as inductive knowledge-based agents for this purpose. In future, other syntactic and connectionist learning approaches can and must be explored; providing alternative solutions to the same design problem and forming a concrete ground on which comparative studies can be carried out.

Our application of classifier systems for the simulation of sensitivity analysis as done by designers during the initial stages of the design process has produced a set of useful design heuristics which can constitute a part of an overall design methodology for the design of corrugated diaphragms. In order to extend this work, we must investigate appropriate modifications of the design environment, and classifier rule syntax that enables the classifier system to induce a sequence of design heuristics, each based on empirical relationships derived from set(s) of dimensionless curves. The process involves an inductive search within a set of possible design heuristics constituting a design methodology. The set of dimensionless curves representing the design environment, must invariably contain all the relevant design information regarding the class of the diaphragm. The allocation of reward to the classifier system must be based on the quality of the induced design methodology which incorporates the degree to which it can satisfy the design requirements.

Mirza (1983) has suggested and derived a number of techniques for the extraction of design methodologies for snap-action diaphragms. These techniques are based on a set of constructed dimensionless curves. His derived design methodology has been used for industrial design automation. However, he used a manual procedure for the derivation of his design methodology. Consequently the automation of this process has a high practical importance and can be extended to much more complex inductive design heuristic extraction processes which are impractical using conventional methods.

We also suggest the use of a community of classifier system modules that cooperate according to our proposed theoretical framework. We can also use syntactic or connectionist knowledge-based agents in this framework. In this way, we can also investigate group induction processes at the coarse-grained level of processing.

As elaborated in chapter 6, in our application of a classifier system, some undesirable behaviour related to the stability of default hierarchies has been observed. A number of schemes, for making the bidding competition in classifier systems more similar to auctions in real markets, have been suggested. To complement the above studies, we must also investigate alternative Bucket-Brigade schemes for encouraging superior default hierarchy formation and achieving better persistence and stability.

In the case of genetic algorithmic optimization, we must construct better theories of trial allocation, and carry out more studies of genetic algorithm hard problems. These studies are, also, vital to our understanding of inductive learning. In particular, we suggest more investigations for improving the sharing schemes used in multi-modal design optimization applications. For example, simple on-line and off-line performance measures are not directly suitable for judging the distribution pattern of the trials over the peaks in the case of multimodal function optimization. Therefore, experiments with better performance metric indicators, such as Deb & Goldberg's (1989b) chi-square-like criterion, for the analysis of the performance of our normalized sharing formulation and possible extensions of this formulation is also an important future work.

Finally, in our use of genetic algorithms for the design optimization of instrument sub-systems, we must strive for the development and use of more accurate mathematical models. These more accurate mathematical models are invariably in the form of distributed parameter mathematical models and are based on the solution of differential equations emerging from the application of the basic physical laws to a design concept. These mathematical models are expected to be more accurate as compared to analytical models, because they take into account all the geometric features and material properties. Therefore, the use of more accurate mathematical models will enable the genetic algorithm to exploit its more realistic image of its design environment to come up with much better optimal designs.

REFERENCES

Adeli, H. (1988). *Expert Systems in Construction and Structural Engineering.* Chapman and Hall Ltd.

Andreeva, L.E. (1966). *Elastic elements of instruments.* Israel Program for Scientific Translations Ltd. Jerusalem.

Bagley, J.D. (1967). *The behaviour of adaptive systems which employ genetic and correlation algorithms.* PHD Dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 68-7556.

Baker, J. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms and Their Applications,* Carnegie-Mellon University Press, Pittsburgh, PA.

Bazaraa, M.S. (1979). *Nonlinear programming.* Chichester: Wiley, New York.

Belew, R.K. & Booker, L.B. (Eds.) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Bethke, A.D. (1981). *Genetic Algorithms as Function Optimizers.* PHD. dissertation (C. C. S.). University of Michigan, Ann Arbor.

Bond, A.H. & Gasser, L. (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann.

Booch, G. (1991). *Object Oriented Design with Applications.* The Benjamin Cummings Publishing.

Booker, L.B. (1982). *Intelligent behaviour as an adaptation to the task environment.* PHD Dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 8214966.

Bozorth, H. (1964). *Ferromagnetism.* Van Nostrand.

Brown, D.C. & Chandrasekaran, B. (1983). An approach to expert systems for mechanical design. In *Trends and Applications, IEEE Computer Society,* pp 173-180.

Burton, P.J. (1990). *A Model of the Design Process.* PHD Thesis, Engineering Design Centre, Measurement and Instrumentation Centre, City Universsity.

Cammarata, S. (1983). Strategies of Cooperation in Distributed Problem Solving. in *Proceedings of the 8th. Int. Joint Conf. Artificial Intelligence*, pp 767-770.

Carbonell, J.G. (1983). Learning by Analogy: Formulating and Generalizing plans from past experience. In *Machine Learning: An Artificial Intelligence Approach - Vol. 1*, Tioga, Palo Alto, CA.

Carbonell, J.G. (1986). Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In *Machine Learning: An Artificial Intelligence Approach - Vol. 2*, Morgan Kaufmann, Los Altos, CA.

Cavicchio, D.J. (1970). *Adaptive search using simulated evolution.* PHD Dissertation, University of Michigan. Ann Arbor, MI.

Chandrasekaran, B. (1981). Natural and Social System Metaphors for Distributed Problem Solving: Introduction to the issue. *IEEE Transactions on Systems, Man and Cybernetics*, January, SMC-11(1):1-5.

Chandrasekaran B. & Brown, D.C. (1989). *Design Problem Solving.* Pitman Publishing.

Chandrasekaran, B. (1990). Design Problem Solving: A Task Analysis. *AI MAGAZINE*, Winter Issue.

Corkill, D. & Lesser, V. (1983). The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network. *Proceedings of the Eighth International Conference On Artificial INtelligence*, Karlsruhe, West Germany.

Davis, R. & Lenat, D.B. (1981). *Knowledge-Based Systems in Artificial Intelligence.* McGraw Hill, New York.

Davis, R. & Smith, R.G. (1983). Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, Vol. 20, pp 63-109.

Davis, L.D. (1985). Job shop scheduling with genetic algorithms. in: *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Carnegie-Mellon University Press, Pittsburgh, PA, 136-140.

Davis, L.D. (Ed.) (1987). *Genetic algorithms and simulated annealing.* Research Notes in Artificial Intelligence. Los Altos, CA: Morgan Kaufmann.

Deb, K. (1989a). *Genetic algorithms in multimodal function optimization.* TCGA Report No. 88002, University of Alabama, Tuscaloosa.

Deb, K. & Goldberg, D.E. (1989b). An Investigation of Niche and Species Formation in Genetic Function Optimization. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

DeJong, K. A. (1975). *An analysis of the behaviour of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 76-9381.

Doeblin, E.O. (1983). *Measurement Systems.* McGraw-Hill, Tokio.

Durfee, E.H., Lesser, V.R. & Corkill, D.D. (1985). Coherent cooperation among communicating problem solvers. *In Proceedings of the 1985 DAI Workshop*, Dec., pp 231-276.

Dufree, E.H., Lesser, V.R. & Corkill, D. (1987). Cooperation Through Communication in a Distributed Problem Solving Network. In *Distributed Artificial Intelligence*, Pitman Publishing.

Evans, J. (1982). *The psychology of deductive reasoning*. London: Routledge and Kegan Paul.

Fiacco, A.V. & McCormick G.P. (1968). *Nonlinear programming: Sequential Unconstrained Minimization Techniques*. Wiley, New York.

Finkelstein, L. & Watts, R.D. (1978). Mathematical models of instruments-fundamental principles. *J Phys E Sci Instrum*, 11, 841-855.

Finkelstein, L. & Finkelstein, A.C.W. (1983). Review of Design Methodology. *IEE Proceedings-A*, Vol. 130, No. 4.

Forrest, S. (1985). *Documentation for PRISONERS DILEMMA and NORMS programs that use genetic algorithm*. University of Michigan, Ann Arbor.

Forrest, S. (1990). Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Nature and Artificial Computing Networks. *The Proceedings of the 9th. Annual CNLS Conference*.

Fox, M.S. (1981). An organisational view of distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, January, SMC-11(1):70-80.

Frantz, D.R. (1973). *Non-linearities in genetic adaptive search*. Doctoral dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 73-11116.

Freeman, P. & Newell, A. (1971). A Model for Functional Reasoning in Design. In *Proceedings IJCAI-71*, Morgan Kaufmann, London.

French, M.J. (1988). *Invention and Evolution: Design in nature and engineering*. Cambridge University Press, Cambridge.

Gasser, L., Braganza C. & Herman N. (1987). MACE: A Flexible Testbed for Distributed AI Research. In Huhns, M.N. (Ed.) *Distributed Artificial Intelligence - Volume 1*, pp 119-152, Pitman Publishing.

Gasser, L. (1988). Distribution and Coordination of Tasks Among Intelligent Agents. *Proceedings of the First Scandinavian Conference on Artificial Intelligence*, March, pp 177-192.

Genesereth, M.K. & Ginsberg, M.L. (1986). Cooperation without Communication. In *Proceedings of the 5th. Nat. Conf. A.I.*, Philadelphia, PA, Aug., pp 51-57.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7, pp 155-170.

Gero, J.S. (Ed.). (1988). *Artificial Intelligence in Engineering Diagnosis and Learning*. Computational Mechanics Publications.

Gero, J.S. (Ed.). (1990). *Application of Artificial Intelligence in Engineering, Vol. 1 - Design*. Computational Mechancs Publications.

Gillies, A.M. (1985). *Machine learning procedures for generating image domain feature detectors*. Doctoral dissertation, University of Michigan, Ann Arbor.

Goldberg, D.E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 44(10), 3174B.

Goldberg, D.E. & Samtani, M.P. (1986). Engineering optimization via genetic algorithm. *Proceedings of the Ninth. Conference on Electronic Computation*, 471-482.

Goldberg, D.E. (1987a). Simple genetic algorithms and the minimal deceptive problem. In: L.D. Davis (Ed.), *Genetic algorithms and simulated annealing.* Research Notes in Artificial Intelligence. Los Altos, CA: Morgan Kaufmann.

Goldberg, D.E. & Richardson, J. (1987b). Genetic algorithms with sharing for multimodal function optimization. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 41-49.

Goldberg, D.E. (1989a). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison Wesley.

Goldberg, D.E. (1989b). Genetic algorithms and Walsh functions: Part II, Deception and its analysis. *Complex Systems*, 3, 153-171.

Goldberg, D.E. (1992). *Genetic algorithms as a computational theory of conceptual design.* Dept. of General Engineering, Univ. of Illinois, IL 61801.

Gomez F. & Chandrasekaran, B. (1981). Knowledge Organisation and Distribution for Medical Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, January, pp. 34-42.

Grefenstette, J.J., Gopal, R. (1985a). Genetic algorithms for the travelling salesman problem. in: *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Carnegie-Mellon University Press, Pittsburgh, PA, 136-140.

Grefenstette, J.J. (Ed.). (1985b). *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Carnegie-Mellon University Press, Pittsburgh, PA.

Grefenstette, J.J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16 (1), 122-128.

Hammon, S.Y. & Gage, D.V. (1984). Coordination of intelligent subsystems in complex robots. *Proceedings of the 1st. IEEE Conf. Artificial Intelligence Applications.*

Hayes-Roth, F. (1980). *Towards a Framework for Distributed A.I.* October, SIGART newslett.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, Vol. 26, pp 251-321.

Holland, J.H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Holland, J. (1986a). Escaping Brittleness: The possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In *Machine Learning: An Artificial Intelligence Approach - Vol. 2*, Morgan Kaufmann, Los Altos, CA.

Holland, J.H. , Holyoak, K.J., Nisbett, R.E. and Thagard, P.R. (1986b). *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, MA.

Holland, J.H. (1987). Genetic algorithms and classifier systems: Foundations and future directions. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 82-89.

Hollstien, R.B. (1971). *Artificial genetic adaptation in computer control systems.* P.H.D. Dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 71-23773.

Knuth, D.E. (1981). Seminumerical Algorithms. *The Art of Computer Programming*, Vol. 2, Reading Mass: Addison-Wesley, Second Edition.

Kornfeld, W.A. & Hewitt, C.E. (1981). The Scientific Community Metaphor. *IEEE Transactions on Systems, Man and Cybernetics*, January, SMC-11(1): 24-33.

Lander, S. & Lesser, V.R. (1991). *Conflict Resolution Strategies for Cooperating Expert Agents.* Department of Computer Science and Information Science, University of Massachusetts, Amherst, MA 01003.

Lenat, D. (1983). The role of heuristics in learning bydiscovery: three case studies. In *Machine Learning: An Artificial Intelligence Approach - Vol. 1*, Tioga, Palo Alto, CA.

Lesser, V.R. & Erman, L.D. (1980). Distributed Interpretation: A Model and Experiment. *IEEE Transactions on Computers*, Dec., C-29(12):1144-1163.

Lesser, V.R. & Corkill, D.D. (1987). Distributed Problem Solving. In *Encyclopedia of Artificial Intelligence.* Stuart C. Shapiro, pp 245-251, John Wiley and Sons, New York.

Maher, M.L. (1990). Process models of design synthesis. *AI MAGAZINE*, Winter Issue.

Massey, B.S. (1971). *Dimensional analysis and physical similarity.* Van Nostrand Reinhold Co., New York.

McArthur, D., Steeb R. & Cammarata, S. (1982). A Framework for Distributed Problem Solving. *Proceedings of the National Conference on Artificial Intelligence*, August, Pittsburgh, PA.

McClelland, J.L. & Rumelhart, D.E. and the PDP Research Group. (1987). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* (2 Volumes), MIT Press, Cambridge, MA.

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In *Machine Learning: An Artificial Intelligence Approach - Vol. 1*, Tioga, Palo Alto, CA.

Mirza, M.K. (1983). *Mathematical Modelling and Design of Snap-Action Diaphragms.* Doctoral dissertation, City University.

Mirza, M.K., Nevest, F.J.R. & Finkelstein, L. (1990a). A knowledge-based system for design-concept generation of instruments. *Measurment*, Jan-Mar, Vol 8, No. 1.

Mirza, M.K., Hill, W.J. & Finkelstein, L. (1990b). *Application of Knowledge Engineering in CAD of Instruments.* Presented at IMEKO Conference, Moscow.

Mirza, M.K. & Finkelstein, L. (1992). Mathematical Modelling of Instruments - Application and Design. *From Instrumentation to Nanotechnology.* Gardner, J.W. and Hingle, H.T. (Eds.), Gordon and Breach Science Publications.

Navinchandra, D. (1988). Case Based Reasoning in CYCLOPS, a Design Problem Solver. In *Proccedings of a Workshop on Case-Based Reasoning*, Kolonder, J. (Ed.), Holiday Inn, Clearwater Beach, Florida.

Neubert, H. (1975). *Instrument Transducers.* Second Edition, Claredon Press.

Newell, A. & Simon, H. (1963). *A program that simulates human thought.* In Computers and Thought, McGraw Hill, New York.

Newell, A. & Simon, H. (1972). *Human Problem Solving.* New Jersey: Prentice Hall.

Newell, A. (1979). Reasoning, Problem Solving and Decision Process: The Problem Space as a fundumental category. In *Attention and Performance*, Volume 8, 693-718.

Nillson, N.J. (1986). *Principles of Artificial Intelligence.* Springer-Verlag.

Osborn, A.F. (1953). *Applied Imagination.* Charles Scribner's Sons, New York.

Pahl, G. & Beitz, W. (1984). *Engineering Design.* The design council, London.

Press, W.H. (1988). *Numerical recipes in C, the art of scientific computing.* Cambridge University Press.

Rahman, M.M. (1979). *Mathematical modelling and computer aided design of field coupled electro-mechanical transducers.* PHD dissertation, City University.

Richardson, J.T. & Palmer, M.R. (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Riolo, R.L. (1987). Bucket brigade performance. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms.* Hillsdale, New Jersey: Lawrence Erlbaum Assoc.

Rosenberg, R.S. (1967). *Simulation of genetic algorithms with biochemical properties.* P.H.D. Dissertation, University of Michigan. Dissertation Abstracts International, University Microfilms No. 67-17836.

Rosenschein, J.S. & Genesereth, M.R. (1985). Deals Among Rational Agents. In *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*, August, pp 91-99.

Schaffer, J.D. (Ed.). (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Schank, R.C. (1980). Language and Memory. *Cognitive Science*, Vol. 4, No. 3, pp 243-284.

Shaw, M.J. & Whinston, A.B. (1989). Learning and Adaptation in Distributed Artificial Intelligence Systems. In Gasser, L. & Huhns, M.N. (Eds.), *Distributed Artificial Intelligence - Volume 2,* pp 413-429, Pitman Publishing.

Simon, H. A. (1957). *Models of Man.* Wiley, New York.

Simon, H.A. (1969). *The Sciences of the Artificial.* MIT Press.

Smith, R.G. (1980). The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, December, C-29(12):1104-1113.

Smith, R.G. & Davis R. (1981). Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(1): 61-70.

Sriram, D. & Adey, R. (Ed.s). (1986a). *Application of Artificial Intelligence in Engineering Problems.* Springer-Verlag.

Sriram, D. & Adey, R. (Ed.s). (1986b). *Knowledge Based Expert Systems for Engineering Problems.* Springer-Verlag.

Sriram, D. (1987). *Knowledge-Based Approaches for Structural Design.* Computational Mechanics Publications.

Sueyoshi, T. & Tokoro, M. (1990). *Dynamic Modelling of Agents for Coordination.* Dept. of Computer Science, Keio University.

Tanese, R. (1989). Distributed genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Thompson, D. (1961). *On Growth and Form.* Abridged Edition (J.T. Bonner, Ed.), Cambridge University Press.

Tomovic, R. & Vukobratovic, M. (1970). *General Sensitivity Theory.* American Elsevier Inc., New York.

Topping, B.H.V. (1989). *Artificial Intelligence Techniques and Applications for Civil and Structural Engineers.* Civil-Comp Press.

Verilli, R.J., Meunier K.L. & Dixon, J.R. (1988). *Iterative Respecification Management: a model for problem solving networks in mechanical design.* Mechanical Design Automation Laboratory, Department of Mechanical Engineering, University of Massachusetts, Amherst, Massachusetts.

Wesson, R.B. & Hayes-Roth, F.A. (1981). Network Structures for Distributed Situation Assessment. *IEEE Transactions on Systems, Man and Cybernetics*, Jan., SMC-11(1):5-23.

Whitley, D. (1989). The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Wilson, S.W. (1988). Bid competition and specificity reconsidered. *Complex systems*, 2(6), 705-723.

Winston, P. (1980). Learning and Reasoning by Analogy. Communications of the ACM, Vol. 23, No. 12, Dec., pp 683-703.

Winston, P. (1984). Learning physical descriptions from functional definitions, examples and precedents. *International Symposium on Robotics Research: Brady and Paul (Eds.),* MIT Press.

Yang, J. D., Huhns M.N. & Stephens, L.M. (1985). An Architecture for Control and Communications in Distributed Artificial Intelligence Systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15:316-326.

APPENDICES

APPENDIX-I

1 The software documentation of the Distributed Problem Solver for Conceptual Design of Instruments

The Object Oriented Design methodology, introduced by Booch (1991), was used to develop the software architecture. During the analysis and the early stages of design, the developer has two primary tasks:

1- Identify the classes and objects that form the vocabulary of the problem domain.
2- Invent the structures whereby sets of objects work together to provide the behaviours that satisfy the requirements of the problem.

This design methodology is based on separate independent kinds of design decisions. Among other things, a developer must consider the following fundamental issues in Object-Oriented design:

1- What classes exist and how are those classes related?
2- What mechanisms are used to regulate how objects collaborate?

Answers to these questions can be expressed in each of the following diagrams, respectively:

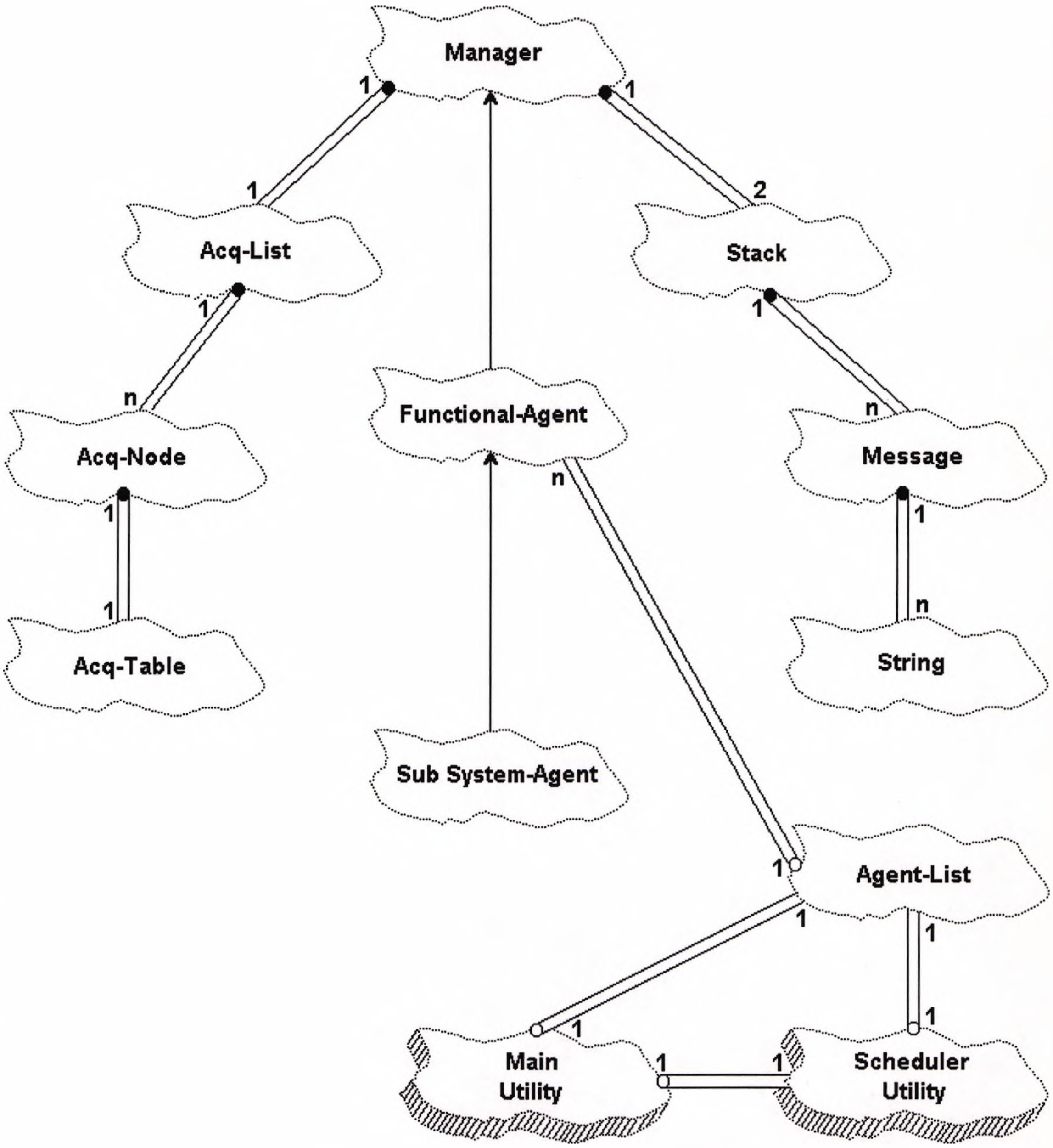
1-Class diagrams
2-Object diagrams

These diagrams form the basic notation of Object-Oriented design (Booch, 1991). Whereas class diagrams reflect the vocabulary of the problem domain, object diagrams represent mechanisms by which instances of particular classes collaborate to provide some behaviour that satisfies a requirement of the problem.Once a developer decides upon a particular mechanism, the work is distributed among many objects by defining suitable methods in the appropriate classes.

Using the informal strategy arrived at during the specifiction of the software requirements (Please refer to chapter 3, section 3.4), the following tangible objects are found as the key abstractions of the problem domain:

1-Manager (Interfacing) Agent
2-Functional Agents
3-Sub-System Agents
4-Env-Model or Acquaintances Model
5-Input-Queue and Output-Queue
6-Messages
7-Scheduler

Accordingly, The following Top-Level Class Diagram is derived:



The Top-Level Class Diagram

1.1 Defining The Semantics and Relationships of the Top-Level Classes

Given the design decisions expressed in the class structure (represented in the above figure), we can now reasonably establish the interfaces of some of these higher level classes:

1- Manager Class

Name: Manager
Cardinality: 1

Implementation:
Superclass: NIL
Uses: functional-agent
Fields: name, num-items, entries
Public Interface:
Operations:
1-Constructor
2-nm (accessor)
3-env-mod (accessor)
4-execute-message (virtual method inherited down the hierarchy, supports-polymorphism)
5-execute-f-message (virtual method inherited down the hierarchy, supports-polymorphism)
6-Link (modifier)
Concurrency: sequential
Persistence: dynamic

2- Functional-Agent Class

Name: (one of Mech-Mech, Mech-Elect, Elect-Mech, Elect-Elect, etc.)
Cardinality: 1
Hierarchy:
Super-Class: Manager Class
Public Interface:
Operations:
1- Constructor
2-Link (modifier)
3-nm (accessor)
4-env-mod (accessor)
5-execute-f-message
Above methods are all inherited from the Manager Agent.
6-execute-message (overloaded virtual)
Implementation:
Uses: Sub-System Agent, stack
Fields: name, entries (inherited)
Concurrency: sequential
Persistence: Dynamic

3-Sub-System Agent Class

Name: (name of a particular instrument sub-system that belongs to a specific energy domain organization)
Cardinality: 1
Hierarchy:
Super-Class: Functional Agent Class
Public Interface:
Operations:
1-Constructor
2-Link (modifier)
3-nm (accessor)
Above methods are all inherited from the Manager Agent.
4-execute-message (overloaded virtual)

Implementation:
Uses: Functional-Agent Class
Fields: name, entries (inherited)
Concurrency: sequential
Persistence: Dynamic

4-String Class

Name: Identifier
Cardinality: 1
Hierarchy:
Super-Class: NIL
Public Interface:
Operations:
1-Constructor
2-assign (modifier)
3-length (accessor)
4-st() (accessor)
5-print() (accessor)
6-Operator == (accessor)
7-Operator = (modifier)
Implementation:
Uses: Pointers to string characters
Fields: String instances, Integer length instances
Concurrency: sequential
Persistence: Dynamic

5- Acq-List Class

Name: env-mod
Cardinality: 1
Hierarchy:
Super-Class: NIL
Public Interface:
Operations:
1-Constructor
2-Destructor
3-add
4-Update
5-Is-In
6-Display
Implementation:
Uses: Acq-node
Fields: List of Acq-node instances
Concurrency: sequential
Persistence: Dynamic

6-Acq-node Class

Name: Identifier
Cardinality: n

Hierarchy:
  Super-Class: NIL
Public Interface:
  Operations:
    1-Constructor
Implementation:
  Uses: Acq-table
  Fields: Acq-table instance
Concurrency: sequential
Persistence: Dynamic

7- Acq-table Class

Name: Identifier
Cardinality: n
Public Interface:
  Operations:
    1-Constructor
    2-isnt-in-connect
    3-isnt-out-connect
    4-is-in-inconnect
Implementation:
  Uses: in-out class
      string class
  Fields:
      name string class
      in-out skill instance
      in-connect string instance
      out-connect string instance
      integer mark
Concurrency: sequential
Persistence: Dynamic

8-Stack Class

Name: Identifier
Cardinality: 2
Hierarchy:
  Super-Classes: NIL
Public Interface:
  Operations:
    1-Constructor
    2-Destructor
    3-reset
    4-push
    5-pop
    6-top-of (accessor)
    7-pri (accessor)
    8-ptpri (accessor)
    9-empty (accessor)
    10-full (accessor)

Implementation:
  Uses: Message class
  Fields:
    Array of message instances
    Integer instance variable "top"
Concurrency: sequential
Persistence: Dynamic

9-Message Class

  Name: Identifier
  Cardinality: n
Hierarchy:
  Super-classes: NIL
Public Interface:
  Operations:
    1-Constructor
    2-printm (accessor)
    3-fn (accessor)
    4-tn (accessor)
    5-sone (accessor)
    6-stwo (accessor)
    7-mt (accessor)
    8-mb (accessor)
Implementation:
  Uses: String class
    Mess-Bod Class
  Fields:
    1-string instance from-name
    2-string instance to-name
    3-Mess-Bod instance message-body
    4-string instance sub-one
    5-string instance sub-two
    6-string instance message-type
Concurrency: sequential
Persistence: Dynamic

10-Agent-List Class

  Name: Identifier
  Cardinality: 1
Hierarchy:
  Super-class: NIL
Public Interface:
  Operations:
    1-Constructor
    2-next() (accessor)
    3-add_after (accessor)
    4-is_in (accessor)
    5-find_in
Private Interface:
    1-is_not_marked

2-find_in
Implementation:
Uses: functional-agent class
Fields: list of functional-agent instances
Concurrency: sequential
Persistence: Dynamic

11-Scheduler Class Utility

Name: Scheduler
Documentation:
Schedules functional-agents, with out-queue messages, for processing. It sends the out-queue messages of a functional agent to addressee destination agents and invokes the execute-f-message method of each destination agent for processing of the received message. It terminates when there are no functional agents with out-queue messages to be processed.
Parameters: Agent-List
Implementation:
Uses: Agent-List Class
Fields: Message object instances

12-Main Utility

Name: Main
Documentation:
Sets up the agents of the DPS system. Sets up the Agent-List container class of the energy domain functional agents. Supports the interface of the software by sending the top-level design requirements to the manager agent. After the task-sharing phase is completed, it sends appropriate messages to the created agent-list instance for the initiation of the result-sharing phase.
Implementation:
Uses: Manager class and all of its sub-ordinate energy domain functional agents, Agent-List class and scheduler utility.
Fields: String object instances, Message object instances.

2 Objects and Object Relationships

A single object diagram represents all or part of the object structure of a system. Typically, the design of a system requires a set of object diagrams.

The purpose of each object diagram is to illustrate the semantics of key mechanisms in the logical design. Classes are largely static in the design of a system, whereas objects are much more transitory, in that many of them may be created and destroyed during the execution of a single program. Therefore, we use object diagrams to capture the dynamic semantics of operations. A single object diagram represent a snap-shot in time of an otherwise transitory event. In this sense, object diagrams represent the interactions that may occur among a collection of objects, no matter what specifically named instances participate in the mechanism.

In what follows, we give the important object diagrams of the implemented DPS system.

2.1 Object Mechanisms for Task-Sharing

During the task-sharing phase of the problem solving (as described in chapter 3) the following object diagram mechanisms are invoked sequentially:

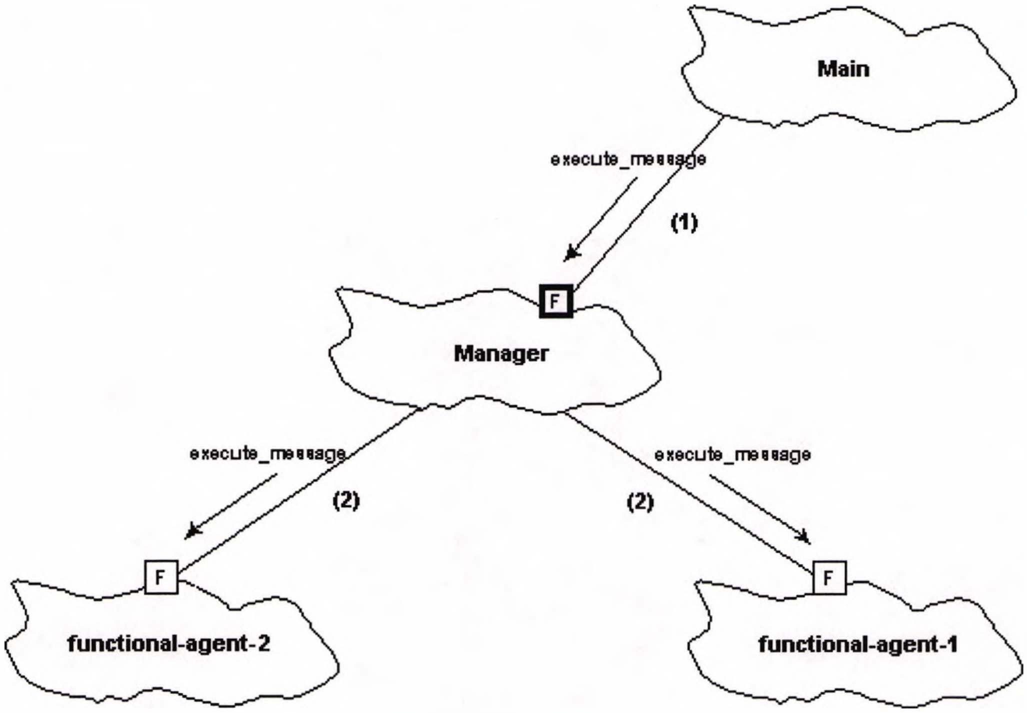


Figure-1 The Manager Object Diagram

The object templates for the above object diagram are shown below:

1-1 Manager object template:

Name: Identifier
Documentation: Is the interfacing agent which sends design goals, contained within the HAVE? message-type to all of its sub-ordinate functional agents.
Class: Manager
Persistence: dynamic

1-1-1 Message Operation:

Name: execute_message
Formal Parameters:
Message object instances

Action:
This message is initiated by the main() class utility. The execute_message method is a virtual method that is inherited by all of the agents sub-ordinate to the manager object instance. The manager object instance, by using this method, sends the initial overall design goal to all of the functional agents across the problem solving net. Each functional agent responds according to its particular execute_message method (i.e. this process supports polymorphism).
Exception: NIL
Concurrency: sequential

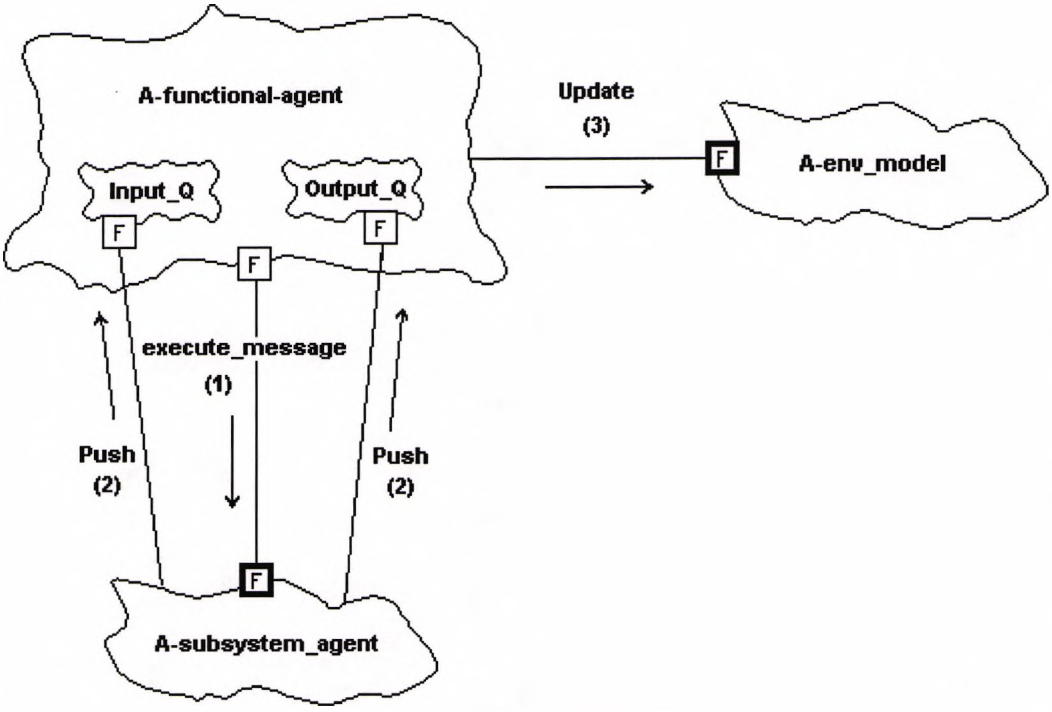


Figure-2 Functional-Agent object disgram

2-1 Functional-Agent object template:

Name: The name of the functional agent object instance
Documentation: Each functional-agent object instance is instantiated during the setting up stage of the network. A functional-agent object, with an allocated design requirement, by using this method, sends the received messages to its sub-system organization members. In turn, the addressee sub-system agents might return information providing messages to the input-queue of their functional agents. The input-queue messages are next poped (by the functional agent) and used to update the env-model attribute of the functional agent.
Class: Functional agent's class (one of functional agent classes, i.e., Mech-Mech, Mech-Elect, Elect-Elect, etc.).
Persistence: dynamic

2-1-1 Message Operation:

Name: execute_message

Formal parameters: Message object instances

Action:

This method is invoked by the manager object instance (during the allocation of the top-level design goals, as shown in figure 1). It is overloaded by each functional agent class across the DPS system. Each functional agent, by using this method, will invoke the execute_message methods of its appropriate sub-system organization members. In this way, it sends the allocated top-level design goal to its sub-system organization members. Next, during negotiations, according to the received messages from its sub-system agents (as shown in figure-2), it updates its env_model object instance.

Exception: If no information providing messages are received from the sub-system agents, the env_model instance will not be updated.

Concurrency: sequential

2-2 Sub-System object template:

Name: The name of the sub-system object instance (i.e., the name of a particular sub-system instrument that belongs to a specific energy domain organization)

Documentation: A sub-system object instance belongs to a particular energy domain organization (i.e., a private field of its energy domain functional agent). Sub-system object instances relate input power,effort or flow variables, represented by a particular energy domain, to the output power, effort or flow variables, represented by the same or a different energy domain. Each sub-system object instance might have domain knowledge, concerned with other compatible energy domain organizations, across which compatible sub-system elements might be found that can be connected to them for a particular overall design goal. Sub-system agents, during negotiations, will send information-providing and/or information-requiring messages to their functional agent queues.

Class: The class of the sub-system

Persistence: dynamic

2-2-1 Message operation:

Name: execute_message

Formal Parameters: Message object instances

Action:

This is a virtual method that is inherited from the manager class down the class hierarchy. It is overloaded, by each sub-system agent instance, to support polymorphism. During the negotiations, this method is invoked by each functional agent (as shown in figure-2). It is used, by each sub-system agent instance, to respond to the design requirement messages allocated by functional agents. These messages are processed according to their types.

Exception: If the capability of a sub-system agent is not relevant to an allocated design requirement message, the addressee sub-system agent does not participate in the negotiations.

Concurrency: sequential

2-3 env_model object template:

Name: env_model identifier for a particular functional agent object instance.
Documentation: Each env_model object instance is a private field which belongs to a particular energy domain functional agent (as shown in figure 2). It supports the overall structure and methods of the env_model qualifier for a specific functional agent object instance.
Class: Acq_list
Persistence: dynamic

2-3-1 Message Operation:

Name: update
Formal Parameters: Information providing message object instances
Action:
A functional agent, during negotiations with sub-ordinate sub-system agents and other energy domain organizations, invokes this method to update its env_model instance of the design environment. This method modifies the env_model attribute of a functional agent according to the message type received.
Exception: If a message type is not understood, no action takes place.
Concurrency: sequential

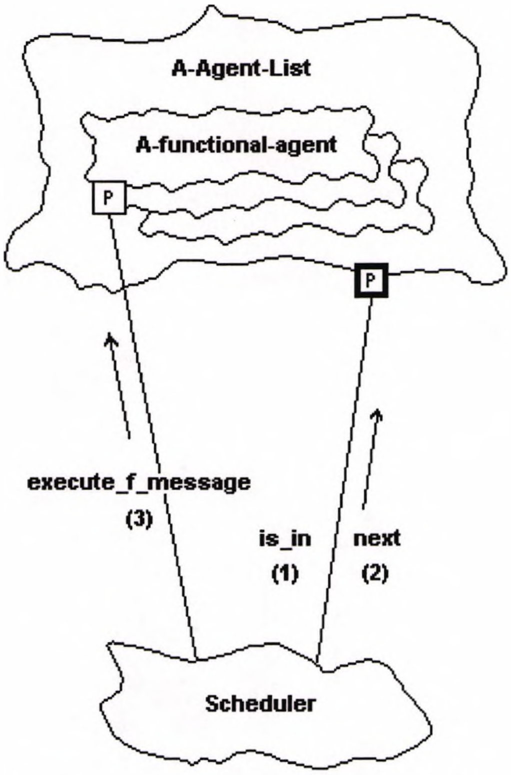


Figure-3 The Object Diagram of the Scheduler Utility

3-1 Agent_List object template:

Name: Identifier (the name of one of the contained functional agents is sufficient)
Documentation: The Agent_List is the container class of the functional agents that belong to the problem solving net. It is implemented in the form of a circular linked-list data structure.
Class: Agent_List class
Persistence: dynamic

3-1-1 Message Operation:

Name: is_in
Formal Parameters: NIL
Action:
This method is invoked by the Scheduler utility to check the existence of functional agents with output-queue messages across the problem solving net.
Exception: NIL
Concurrency: sequential

3-1-2 Message Operation:

Name: next()
Formal Parameters: NIL

Action:
This method is invoked, by the scheduler utility, to find the addressee functional agent of an intra-organizational message. It returns a pointer to the next functional agent within the Agent_List object instance.
Exception: NIL
Concurrency: sequential

3-1-3 Message Operation:

Name: execute_f_message (is a method that belongs to the manager class and is inherited down the problem solving hierarchy)
Formal Parameters: message object instances
Action:
This method is invoked, by the scheduler utility, to enable a functional agent to start processing its input-queue messages. It deals with the intra-organizational messages received by a functional agent. The execute_f_message can have s_have or s_need message-type instances as its formal parameters. The invoked functional agent uses this method to respond to the received message by either updating its env_model, invoking negotiations across its organization or checking its env_model instance. Each action depends on the type of received message.
Exception: If the received message instance is not understood, no action is invoked.
Concurrency: sequential

2.2 Result Sharing

The result-sharing occurs during the answer-synthesis phase of the problem solving. During answer-synthesis, the energy domain functional agents will share their partial solutions to arrive at complete conceptual design solutions. During this phase, the main utility invokes the "find_in" method of the Agent_List container class which contains a circular linked-list of the energy domain functional agents (as shown in figure 4).

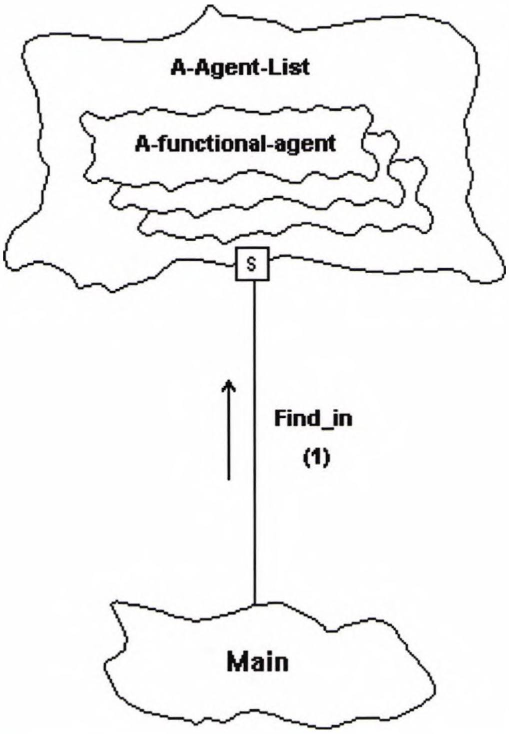


Figure-4 The Object diagram mechanism for the Answer-Synthesis phase of the problem solving

4-1 Message Operation:

Name: Find_in (belongs to the Agent_List class)
Formal Parameters: String object instances
Action:
Finds functional agent instances which possess, inside their env_model attributes, models of sub-ordinate sub-system agents satisfying the overall input of the design goal. If they are complete concepts, they are displayed to the user, otherwise a recursive method is initiated to synthesize the partial solutions existing across the network of functional agents. During this process, alternative conceptual designs are derived. This method supports the result-sharing concept.
Exception: If functional agents have no partial solutions, no overall solutions are synthesized.
Concurrency: sequential

APPENDIX-II

Genetic Algorithm Program Listing

MAIN PROGRAM

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define maxpop 100
#define maxstring 40
#define max_best 34
#define N_parms 4
#define Cmax 1000.0
#define tourneysize 3
#include "Trnd.h"


main(int argc, char** argv)
{

 numfiles = argc - 1;

 if(numfiles == 0)
   {
    infp = stdin;
    outfp = stdout;
   }

 if(numfiles == 1)
 {
     if((infp = fopen(argv[1],"r")) == NULL)
     {
       fprintf(stderr,"Cannot open input file %s\n",argv[1]);
       exit(-1);
     }
     outfp = stdout;
     grp_fp_max = stdout;
     grp_fp_avg = stdout;
 }

 if(numfiles == 3)
 {
   if((infp = fopen(argv[1],"r")) == NULL)
     {
       fprintf(stderr,"Cannot open input file %s\n",argv[1]);
       exit(-1);
     }

    outfp = stdout;

   if((grp_fp_max = fopen(argv[2],"w")) == NULL)
     {
       fprintf(stderr,"Cannot open output file %s\n",argv[3]);
       exit(-1);
     }
   if((grp_fp_avg = fopen(argv[3],"w")) == NULL)
     {
       fprintf(stderr,"Cannot open output file %s\n",argv[3]);
       exit(-1);
     }
}

 if(numfiles == 4)
 {
   if((infp = fopen(argv[1],"r")) == NULL)
     {
       fprintf(stderr,"Cannot open input file %s\n",argv[1]);
       exit(-1);
     }
```

```
  if((outfp = fopen(argv[2],"w")) == NULL)
    {
     fprintf(stderr,"Cannot open output file %s\n",argv[2]);
     exit(-1);
    }
  if((grp_fp_max = fopen(argv[3],"w")) == NULL)
    {
     fprintf(stderr,"Cannot open output file %s\n",argv[3]);
     exit(-1);
    }
  if((grp_fp_avg = fopen(argv[4],"w")) == NULL)
    {
     fprintf(stderr,"Cannot open output file %s\n",argv[4]);
     exit(-1);
    }

 }


 while(1)
 {
 int k,j,flag,GEN;
 flag = GEN = 0;
 initialize();
 do {
    GEN++;
    flag++;
    generation(GEN);
    statistics(newpop);
    if((flag%10)==0)
    report(GEN);
    short_report(GEN);
    fprintf(outfp,"After report in main \n");
    fprintf(outfp,"====================================================\n");
    Assign(oldpop, newpop); /* assign old population to new population */
    }   while(GEN <= MAXGEN);

  Disp_GLBST(GLB_BST);    /* Display Global Best Design */

   printf("Do you wish to start a new design ? \n");
   scanf("%s", &a);
   if (strcmp(a, "yes"));
   break;
  }
 }
```

GLOBAL VARIABLES

```
int numfiles;

typedef struct indiv {
        int chromosome[maxstring]; /* Genotype = bit position */
        double x1;   /* Phenotype = unsigned integer (for n) */
        double x2;   /* Phenotype = unsigned integer (for Teta) */
        double x3;   /* Phenotype = unsigned integer (for h) */
        double x4;    /* Phenotype = unsigned integer (for Hh) */
        /* Note: don't have double x,i.e. Phenotype = unsigned integer */
        double obj;  /* Objective function value, equivalent to rw_fit */
        double d_rw_fit; /* Parameter to account for environmental niche */
        int Needs_evaluation; /* Used for ranking selection */
        double HH;  /* Corrugation depth */
        double RR;  /* Radious of diaphragn */
        double ll;  /* Linear term */
        double cc;  /* Cubic term */
        double error1, error2; /* percentage nonlinearity error */
        double Terror; /*Total error */
        int itr_n;  /* Iteration Number */
        double  Partition; /* Partition coef., only used by the Best_List data structure */
        int parent1, parent2, xsite; /* parents and cross point */
               } individual;
```

```
typedef struct parms {
        int lparm; /* length of parameter */
        double parameter, maxparm, minparm; /* Parameter and range */
              } parameter;


parameter parm_arr[N_parms];
individual oldpop[maxpop];
individual newpop[maxpop]; /* oldpop & newpop represent two nonoverlapping populations */
individual GLB_BST;       /* Represents global best design */
int choices[maxpop], nremain; /* Parameters used with the stochastic remainder selection */
float fraction[maxpop];
individual Best_List[max_best];

int POPSIZE, LCHROM, GEN, MAXGEN; /* integer global variables */
double PCROSS, PMUTATION; /* double global variables */
double RW_SUMFIT; /* Raw sum_fitness, based on obj */
double D_RW_SUMFIT; /* d_rw_sumfit based on d_rw_fit, for sharing purposes*/
int NMUTATION, NCROSS; /* integer statistics */
double AVG,D_AVG;
double W_AVG, WDOW_SUM_FIT; /* average and sumfitness for scaling */
double RW_MAX, RW_MIN; /* statistics, based on obj  */
double D_RW_MAX, D_RW_MIN; /* statistics, based on d_rw_fit, for sharing purposes  */
double Pmax,Wmax,E; /* Design parameters */
double perc; /* percentage of elite for ranking selection */
double (*p) (individual&); /* pointer to chosen fitness function */
double Mu; /* Penalty coefficient */
```

INTERFACE ROUTINES

```
double objfunc_Force_to_Disp (individual& CR)
{
/* Objective function, returns cost */
 fprintf(outfp,"cr.x3 = %f\n", CR.x3);
 return(-1.0);
}


double objfunc_Press_to_Disp (individual& ind)
{
/* Objective function, returns cost */

double k, k1, k2, alph, exp1, exp2,H;
double exp3,exp4,exp5;
double R,a,b;
double ER1,ER2,TER; /* Nonlinearity error */
double OBJ1,OBJ2;
double L,C,output1,output2,output3,output4;

 k = cos(ind.x2);
 k1 = 1.0/k;
 H = (ind.x4*ind.x3);
 ind.HH = H;
 k2 = (((ind.x4*ind.x4)*k1)+k);
 alph = sqrt(k1*k2);
 exp1 = (2.0*(3.0 + alph)*(1.0 + alph));
 exp2 = (3.0*k1*(1-(0.09/(alph*alph))));
 a = (exp1/exp2);
 exp3 = (32.0*k1)/((alph*alph)-9.0);
 exp4 = (1.0/6.0)-(2.7/((alph-0.3)*(alph+3.0)));
 b = exp3 * exp4;
 R = (H*(2.0*ind.x1))/(tan(ind.x2));
 ind.RR = R;
 L = (E/(R*R*R*R))*(ind.x3*ind.x3*ind.x3)*a;
 ind.ll = L;
 C = (E/(R*R*R*R))*ind.x3*b;
 ind.cc = C;

 output1 = (L*(Wmax))+(C*(pow(Wmax,3.0)));
 output2 = (Pmax);
 ER1 = fabs(output1 - output2);
 ind.error1 = ER1;

 output3 = (L*(Wmax/2.0))+(C*(pow(Wmax/2.0,3.0)));
 output4 = (Pmax/2.0);
 ER2 = fabs(output3 - output4);
 ind.error2 = ER2;

 if (R < REQS) /* Required constraint on diaphragm's radious */
  ER3 = 0.0;
 else
  ER3 = fabs(R - REQS);

 TER = ER1 + ER2 + (Mu*((ER3*ER3)));
 ind.Terror = TER;
 return ( 1.0/(1.0+(TER)));
 }


double decode (int* chromosome, int lbits)
{
/* Decode string as unsigned binary integer - true=1, false=0 */
int j;
double accum, powerof2;

accum = 0.0; powerof2 = 1.0;

for(j=0; (j<=lbits-1) ; j++)
  {
  if (check(chromosome[j]))
   {
    accum += powerof2;
   }
    powerof2 *= 2;
  }
return accum;
}


double map_parm(double x,double max_parm,double min_parm,double full_scale)
{
 double value;
 value = min_parm+(((max_parm-min_parm)/full_scale)*x);
 return value;
}


void extract_parm(const int*& chromfrom,int *chromto,int& jposition,int lchrom,int lparm)
{
/* Extracts a sub_string from a full string */
 int j, jtarget;
 j=0;
 jtarget = jposition+lparm-1;

 if(jtarget>lchrom-1)
   jtarget=lchrom-1; /* clamp if excessive */
 while (jposition <= jtarget)
 {
  chromto[j]=chromfrom[jposition];
  j++; jposition++;
  }
}


void map_parms(int N_Parms,int lchrom,const int *chrom,parameter *par_ar)
{
 int j,jposition;
 int chr_temp[maxstring]; /* Temp chrom buffer */
 j=0; /* Parameter counter */
 jposition = 0; /* string position counter */

 for (int k=0; k<N_Parms; k++)
  {
  if(par_ar[k].lparm > 0)
  {
   extract_parm(chrom,chr_temp,jposition,lchrom,(par_ar[k].lparm));
   par_ar[k].parameter=map_parm((decode(chr_temp,par_ar[k].lparm)),
                 par_ar[k].maxparm,par_ar[k].minparm,(pow(2.0,(par_ar[k].lparm))-1.0));
  }
  else
  {
   par_ar[k].parameter=0.0;
  }
 } /* Termination of loop */
}
```

Initialization Procedures

```
void choose_fitness_function(void)
{
 char b[12];

while(1)
 {
  fprintf(outfp,"Is Diaphragm being used for sensing 'Force' or 'Pressure' ?\n");
  fscanf(infp,"%s",&b);

 if (!strcmp("force", b))
     {
     fprintf(outfp,"You have chosen force \n");
     p = objfunc_Force_to_Disp;
     break;
     }
  else if (!strcmp("pressure", b))
     {
      fprintf(outfp,"You have chosen pressure \n");
      p = objfunc_Press_to_Disp;
      break;
     }
   }
}



void init_parms ( parameter *parm)
/*------------------------------
   Purpose     : Initializes parameters and ranges
   called by   : initdata
   calls       : none
*/
{
 for(int j=0; j<N_parms; j++)
 {
  fprintf(outfp,"Enter length of parm number %d \n",j);
  fscanf(infp,"%d", &parm[j].lparm);
  fprintf(outfp,"Enter its min_range \n");
  fscanf(infp,"%lf",&parm[j].minparm);
  fprintf(outfp,"Enter its max_range \n");
  fscanf(infp,"%lf",&parm[j].maxparm);
  fprintf(outfp,"##########################\n");
 }
}


void initdata(void)
/* -----------------
       Purpose        : interactive data enguiry and setup
       called by      : initialize(void)
       calls          : init_parms,choose_fitness_function,randomize
*/
{
fprintf(outfp,"Enter population size ----------> \n"); fscanf(infp,"%d", &POPSIZE);
fprintf(outfp,"Enter chromosome length --------> \n"); fscanf(infp,"%d", &LCHROM);
fprintf(outfp,"Enter max. generations ---------> \n"); fscanf(infp,"%d", &MAXGEN);
fprintf(outfp,"Enter crossover prrobability ----> \n"); fscanf(infp,"%lf", &PCROSS);
fprintf(outfp,"Enter mutation probability -----> \n"); fscanf(infp,"%lf", &PMUTATION);
fprintf(outfp,"**************************************\n");
fprintf(outfp,"** Need Diaphragm design parameters ** \n");
fprintf(outfp,"**************************************\n");
init_parms (parm_arr);
choose_fitness_function();
fprintf(outfp,"Enter Max. Pressure --->\n");
fscanf(infp,"%lf", &Pmax);
fprintf(outfp,"Enter Max. Displacement  --->\n");
fscanf(infp,"%lf", &Wmax);
fprintf(outfp,"Enter  the Young's modulus of the material --->\n");
fscanf(infp,"%lf", &E);
fprintf(outfp,"Enter Penalty Coefficient \n");
fscanf(infp,"%lf",&Mu);

/* Initialize random number generator */
randomize();
/* Initialize counters */
NMUTATION = 0;
NCROSS = 0;
fprintf(outfp,"END of initdata \n");
}


void Disp_GLBST(individual& bst)
/* ----------------------------
  Purpose      : Displays best design after required number of iterations
  called by    : initreport, report
  calls        : none
*/
{
 fprintf(outfp,"Genotype = "); Twritechrom(bst.chromosome, LCHROM);
 fprintf(outfp,"\n");
 fprintf(outfp,"Phenotype1 (n) = %f \n",bst.x1);
 fprintf(outfp,"Phenotype2 (Teta) = %f \n",bst.x2);
 fprintf(outfp,"Phenotype3 (h) = %f \n",bst.x3);
 fprintf(outfp,"Phenotype4 (Hh) = %f \n",bst.x4);
 fprintf(outfp,"obj = %f \n",bst.obj);
 fprintf(outfp,"d_rw_fit = %f \n",bst.d_rw_fit);
 fprintf(outfp,"H = %f \n",bst.HH);
 fprintf(outfp,"Radious = %f \n",bst.RR);
 fprintf(outfp,"ll = %f \n",bst.ll);
 fprintf(outfp,"cc = %f \n",bst.cc);
 fprintf(outfp,"error1 = %f \n",bst.error1);
 fprintf(outfp,"error2 = %f \n",bst.error2);
 fprintf(outfp,"Terror = %f \n",bst.Terror);
 fprintf(outfp,"Iteration number = %d \n",bst.itr_n);
 fprintf(outfp,"#####################################\n\n");
}


void Disp_Best (individual *B_L)
/*
  Purpose      : Displays relative best designs in each partition
  called by    : initreport, report
  calls        : none
*/

{
 const double coef = 7.0;

 for(int j=0; j<max_best; j++)
 {
  fprintf(outfp,"Best individual number %d \n",j);
  fprintf(outfp,"Genotype = "); Twritechrom(B_L[j].chromosome, LCHROM);
  fprintf(outfp,"\n");
  fprintf(outfp,"Phenotype1 (n) = %f \n",B_L[j].x1);
  fprintf(outfp,"Phenotype1 (Teta) = %f \n",B_L[j].x2);
  fprintf(outfp,"Phenotype1 (h) = %f \n",B_L[j].x3);
  fprintf(outfp,"Phenotype1 (Hh) = %f \n",B_L[j].x4);
  fprintf(outfp,"obj = %f \n",B_L[j].obj);
  fprintf(outfp,"d_rw_fit = %f \n",B_L[j].d_rw_fit);
  fprintf(outfp,"H = %f \n",B_L[j].HH);
  fprintf(outfp,"Radious = %f \n",B_L[j].RR);
  fprintf(outfp,"ll = %f \n",B_L[j].ll);
  fprintf(outfp,"cc = %f \n",B_L[j].cc);
  fprintf(outfp,"error1 = %f \n",B_L[j].error1);
  fprintf(outfp,"error2 = %f \n",B_L[j].error2);
  fprintf(outfp,"Terror = %f \n",B_L[j].Terror);
  fprintf(outfp,"Partition = %f \n",B_L[j].Partition);
  fprintf(outfp,"Iteration number = %d \n",B_L[j].itr_n);
   fprintf(outfp,"***********************************\n\n");
 }
}
```

```
void initreport(void)
/* -----------------
      purpose        : generate Initial report
      called by      : initialize
      calls          : Disp_Best
*/
{
/* Initial report */
int gen = 0;
fprintf(outfp,"------------------------------------------------------------------------\n");
fprintf(outfp,"--GENETIC ALGORITHM FOR DIAPH. SIMULATIONS  --\n");
fprintf(outfp,"------------------------------------------------------------------------\n");
fprintf(outfp,"$$$$$$$$$$$$$$$$$$$$$ GEN ALG PARMS $$$$$$$$$$$$$$$$$$$$$\n");
fprintf(outfp,"Population size (POPSIZE)           = %d \n", POPSIZE);
fprintf(outfp,"Chromosome length (LCHROM)          = %d \n", LCHROM);
fprintf(outfp,"Maximum number of generation (MAXGEN) = %d \n", MAXGEN);
fprintf(outfp,"Crossover probability (PCROSS)       = %f \n", PCROSS);
fprintf(outfp,"Mutation probability (PMUTATION)     = %f \n", PMUTATION);
fprintf(outfp,"############### Initial Generation Statistics ##########################\n");
fprintf(outfp,"------------------------------------------------------------------------\n");
fprintf(outfp,"   Initial population maximum fitness = %f \n", RW_MAX);
fprintf(grp_fp_max,"%d   %f \n",gen,RW_MAX); /* init. file pointer for GRAPHICS */
fprintf(grp_fp_avg,"%d   %f \n",gen,AVG);    /* init. file pointer for GRAPHICS */
fprintf(outfp,"   Initial population average fitness = %f \n", AVG);
fprintf(outfp,"   Initial population minimum fitness = %f \n", RW_MIN);
fprintf(outfp,"   Initial population sum of fitness  = %f \n", RW_SUMFIT);
fprintf(outfp," ********* USER SPECIFIED DESIGN VARIABLES ********* \n");
fprintf(outfp," Pmax = %f Wmax = %f \n",Pmax,Wmax);
fprintf(outfp," Penalty coefficient = %f \n",Mu);
fprintf(outfp,"  Young's Modulus = %f \n",E);
Disp_Best(Best_List);
}


void init_Best_List(individual *B_List)
/*
      purpose        : Initialize the Best_List data structure
      called by      : initpop
      calls          : none
*/
{
 double Part_coef = 0.5; /* Using partitions 0.5 apart for the H/h parameter */
 for(int j=0; j<max_best; j++)
  {
   B_List[j].x3 = 0.0;
   B_List[j].Partition = ((j+1) * Part_coef); /* Set Partition coef. for each Best_List element */
   B_List[j].obj = 0.0;
  }
}


void initpop(void)
/*
      purpose        : Initialize a population at random
      called by      : initialize
      calls          : map_parms, objfunc_press_to_disp, init_Best_List
                       check, flip, speciation   */
{
int j, j1;

 init_Best_List(Best_List);
 for(j=0; j<POPSIZE; j++)
  {
   for(j1=0; j1<LCHROM ; j1++)
    {
    oldpop[j].chromosome[j1] = check(flip(0.5)); /* A fair coin toss */
    }
    map_parms(N_parms,LCHROM,oldpop[j].chromosome, parm_arr);
    oldpop[j].x1=parm_arr[0].parameter;
    oldpop[j].x2=parm_arr[1].parameter;
    oldpop[j].x3=parm_arr[2].parameter;
    oldpop[j].x4=parm_arr[3].parameter;
    oldpop[j].obj = (*p) (oldpop[j]);
    oldpop[j].parent1 = oldpop[j].parent2 = oldpop[j].xsite = 0;
    oldpop[j].itr_n = 0;
  }
 speciation(oldpop);
}
```

```
void initialize(void)
{
/* Initialization coordinator */
initdata();
initpop();
statistics(oldpop);
initreport();
}
```

Reporting Procedures

```
void Twritechrom(const int chromosome[] , int lchrom)
{
/* write a chromosome as a string of 1's (true's) and 0's (false's) */
int j;

 for(j=lchrom-1; j  >= 0; j--)
  if (check(chromosome[j]))
      fprintf(outfp,"1");
    else
      fprintf(outfp,"0");
}


void report(int gen)
/* ----------------------
       Purpose       : write detailed population report
       called by     : main
       calls         : writechrom,Disp_GLBST
*/
{
int j;

/* write the population report */
fprintf(outfp,"Population Report \n");
fprintf(outfp,"Generation %2d \n", gen-1);
fprintf(outfp,"Generation %2d \n", gen);
fprintf(outfp,"           String                X         fitness");
fprintf(outfp,"           # parents  xsite");
fprintf(outfp,"             string            X        fitness\n");
fprintf(outfp,"\n");

for (j=0; j<POPSIZE; j++)
 {
  fprintf(outfp,"%2d) " ,j);
 /* old string */
  writechrom(oldpop[j].chromosome, LCHROM);
  fprintf(outfp," %10.6f %10.6f %6.6f    |", oldpop[j].x3,oldpop[j].x4, oldpop[j].obj);
  /* new string */
 fprintf(outfp,"  %2d ) (%2d , %2d )%2d",j,newpop[j].parent1,newpop[j].parent2,newpop[j].xsite);
  writechrom(newpop[j].chromosome, LCHROM);
  fprintf(outfp," %10.6f %10.6f %6.6f\n", newpop[j].x3, newpop[j].x4, newpop[j].obj);
 }
fprintf(outfp,"\n");
/* Generation statistics and accumulated values */
fprintf(outfp,"Note: Generation %d & accumulated statistics\n",gen);
fprintf(outfp," max=%6.6f min=%6.6f avg=%6.6f d_sumf=%6.6f nmutation=%d ncross=%d\n",RW_MAX, RW_MIN,
AVG,RW_SUMFIT,NMUTATION, NCROSS);
Disp_Best(Best_List);
fprintf(grp_fp_max,"%d   %f \n",gen,RW_MAX);
fprintf(grp_fp_avg,"%d   %f \n\n",gen,AVG);
fprintf(outfp," ************************************ \n ");
fprintf(outfp,"Best design was : \n");  Disp_GLBST(GLB_BST);
fprintf(outfp," ************************************ \n\n");
}


void short_report(int gen)
/* ---------------------
       Purpose       : write short population report
       called by     : main
       calls         : none
*/
{
 fprintf(outfp," ******************** Short Report **********************\n");
 fprintf(outfp,"Note: Generation %d & accumulated statistics\n",gen);
 fprintf(outfp," max=%6.6f min=%6.6f avg=%6.6f sumfitness=%6.6f nmutation=%d ncross=%d\n",RW_MAX, RW_MIN,
AVG,RW_SUMFIT, NMUTATION, NCROSS);
 fprintf(outfp,"********************************************************\n\n");
}
```

Pseudo Random Number Generator

```
static double oldrand[55]; /* array of 55 random numbers */
static int jrand;    /* current random */
static int rndcalcflag;
static double rndx2;

void advance_random (void) /* create next batch of 55 random numbers */
{
 int j1;
 double new_random;

  for(j1=0; j1 < 24 ; j1++)
  {
   new_random = oldrand[j1] - oldrand[j1+31];
   if (new_random < 0.0)
      new_random = new_random + 1.0;
      oldrand[j1]=new_random;
  }
  for(j1=24; j1 < 55; j1++)
   {
    new_random = oldrand[j1] - oldrand[j1-24];
    if (new_random < 0.0)
      new_random = new_random + 1.0;
      oldrand[j1]=new_random;
   }
}


void warmup_random (double random_seed)
/* Get random off and running */
{
 int j1,ii;
 double new_random, prev_random;

 oldrand[54] = random_seed;
 new_random = 0.000000001;
 prev_random = random_seed;
 for ( j1 = 1; j1 <= 54 ; j1++)
 {
  ii = (21*j1) % 54;
  oldrand[ii] = new_random;
  new_random = prev_random - new_random;
  if (new_random < 0.0)
    new_random = new_random + 1.0;
  prev_random = oldrand[ii];
  }
  advance_random(); advance_random(); advance_random();
  jrand = 0;
}


double trandom(void)
{
 ++jrand;
 if (jrand >= 55)
 {
  jrand = 1;
  advance_random();
 }
 return oldrand[jrand];
}
```

```
int flip(double probability)
{
 if(trandom() <= probability)
    return(1);
 else
    return(0);
}


int rnd(int low,int high)
/* pick a random integer between low and high */
{
 int i;

 if (low >= high)
   i = low;
 else
  {
   i =(int) ((trandom() * (high-low + 1)) + low);
   if (i > high)
     i = high;
  }
 return(i);
}


void randomize(void)
{
double randomseed;
 do
  {
    fprintf(outfp,"Enter seed random number (0.0 .. 1.0) > \n");
    fscanf(infp,"%lf", &randomseed);
    fprintf(outfp,"randomseed is %f \n",randomseed);
  }
  while ((randomseed <= 0.0) || (randomseed >= 1.0));
  warmup_random(randomseed);
 }
```

Utility Functions

```
int round (double x)
{
 int integer;
 double fraction;
 integer=(int)(x/1);
 fraction = (x-integer);
 if(fraction <= 0.5)
   return(x/1);
 else
   return((x/1)+1);
}


int trunc (double x)
{
 int integer;
 integer=(int)(x/1);
 return (integer);
}


int check(int x)
{
 if((x>-1)&&(x<2))
  return(x);
 else {
  fprintf(outfp,"EROR!!! -  x is %f\n",x);
  return(x);
       }
}
```

STATISTICAL ROUTINES

```
double ASSGN_W_FIT(individual *pop, double RW_MIN)
{
/* Assign scaled fitness values */
double wdow_sm_fit = 0;
static double r_min = 0.0;

 if (RW_MIN > r_min)
   r_min = RW_MIN;

 for (int j=0; j<POPSIZE; j++)
 {
  pop[j].W_Fit = (pop[j].obj) - r_min;
  wdow_sm_fit += pop[j].W_Fit;
 }
 return(wdow_sm_fit);
}


void statistics (individual * indiv)
{
/* Calculates population statistics */
int j,WEAK;
static int GLB = 0;
int found_best = 0;

 /* Initialize */

/* Note: have used obj, which is equivalent to rw_fit */
 RW_SUMFIT = RW_MIN = RW_MAX = indiv[0].obj;
 D_RW_MIN = D_RW_MAX = D_RW_SUMFIT = indiv[0].d_rw_fit;

 if(GLB == 0)
  {
   Assign_Global_Best(indiv[0]);
   GLB = 1;
  }

 if(indiv[0].obj > GLB_BST.obj)
  {
  Assign_Global_Best(indiv[0]);
  found_best = 1;
  }


 if(indiv[0].obj == GLB_BST.obj)
  found_best = 1;


/* loop for max, min, sumfitness */
 for (j=1; j<POPSIZE; j++)
  {
   RW_SUMFIT += indiv[j].obj; /* accumulate rw_fit sum */
   D_RW_SUMFIT += indiv[j].d_rw_fit;

   if(indiv[j].obj > GLB_BST.obj)
     {
     Assign_Global_Best(indiv[j]);
     found_best = 1;
     }
   if(indiv[j].obj == GLB_BST.obj)
         found_best = 1;
   if (indiv[j].obj > RW_MAX)
        RW_MAX = indiv[j].obj; /* new rw_max */
   if (indiv[j].obj < RW_MIN)
        {
        RW_MIN = indiv[j].obj; /* new rw_min */
        WEAK = j;
        }
   if (indiv[j].d_rw_fit > D_RW_MAX)
        D_RW_MAX = indiv[j].d_rw_fit; /* new d_max */
   if (indiv[j].d_rw_fit < D_RW_MIN)
        D_RW_MIN = indiv[j].d_rw_fit; /* new d_min */
  }
/* Calculate average */
AVG = RW_SUMFIT/POPSIZE; /* Population average fitness */
WDOW_SUM_FIT =  ASSGN_W_FIT(indiv, RW_MIN);
W_AVG = WDOW_SUM_FIT / POPSIZE;   /* Population degraded average fitness, using scaling
     window */
D_AVG = D_RW_SUMFIT/POPSIZE; /* Population degraded average fitness, using sharing */

if(found_best == 0)
 {
  fprintf(outfp," Best individual has been lost\n");
  Assign_All(GLB_BST, indiv[WEAK]); /* replace weak individual with the best individual */
 }
Put_Best(indiv,Best_List);   /* put best individuals into the Best_List data structure */
}
```

GENETIC ALGORITHMIC OPERATORS

```
int mutation(int alleleval, double pmutation)
{
/* Mutate an allele with probability pmutation , count number of mutations */
int mutate;

 mutate = flip(pmutation); /* flip the biased coin */
 if (mutate)
   {
    NMUTATION += 1;
    if (alleleval)
       return (0);
    else
       return (1);
   }
 return (alleleval);
}


int crossover(int *parent1, int *parent2, int *child1,
           int *child2)
{

/* Cross 2 parent strings, place in 2 child strings */
int j,jcross;

if (flip(PCROSS))  /* Do crossover with prob. pcross */
  {
   jcross = rnd(1, LCHROM-2); /* Cross between 1 and l-1 */
   ++NCROSS;
  }
  else
   jcross = LCHROM-1;
 /* First exchange, 1 to 1 and 2 to 2 */
 for(j=0; j <= jcross; j++)
  {
   child1[j]=mutation(parent1[j], PMUTATION);
   child2[j]=mutation(parent2[j], PMUTATION);
  }
/* exchange, 1 to 2 and 2 to 1 */
 for(j=jcross+1; j <= LCHROM-1 ; j++)
  {
   child1[j]=mutation(parent2[j], PMUTATION);
   child2[j]=mutation(parent1[j], PMUTATION);
  }
 /* Note must return jcross here */
 return(jcross);
}


/* ************************************************************************** */
                                             /* Roulete Wheel Selection */
/* ************************************************************************** */


int select()
/* roulette-wheel selection */
{
   float sum, pick;
   int i;

   pick = trandom();
   sum = 0;

   if(D_RW_SUMFIT != 0)
   {
      for(i = 0; (sum < pick) && (i < POPSIZE); i++)
         sum += oldpop[i].d_rw_fit/D_RW_SUMFIT;
   }
   else
      i = rnd(0,POPSIZE-1);

   return(i-1);
}


/* ************************************************************************** */
                                        /* Tournament Selection */
/* ************************************************************************** */


void reset(void)
/* Shuffles the tourneylist at random */
{
   int i, rand1, rand2, temp;

   for(i=0; i<POPSIZE; i++) tourneylist[i] = i;

   for(i=0; i < POPSIZE; i++)
   {
      rand1=rnd(i,POPSIZE-1);
      rand2=rnd(i,POPSIZE-1);
      temp = tourneylist[rand1];
      tourneylist[rand1]=tourneylist[rand2];
      tourneylist[rand2]=temp;
   }
}


void preselect(void)
{
   reset();
   tourneypos = 0;
}


int select()
{
   int pick, winner, i;

   /* If remaining members not enough for a tournament, then reset list */
   if((POPSIZE - tourneypos) < tourneysize)
   {
      reset();
      tourneypos = 0;
   }


   /* Select tourneysize structures at random and conduct a tournament */
   winner=tourneylist[tourneypos];
   for(i=1; i<tourneysize; i++)
   {
      pick=tourneylist[i+tourneypos];
      if(oldpop[pick].obj > oldpop[winner].obj) winner=pick;
   }

   /* Update tourneypos */
   tourneypos += tourneysize;
   return(winner);
}
```

```
/* ************************************************************************* */
                                  /* Schotastic Remainder Selection */
/* ************************************************************************* */


void preselect()
/* preselection for stochastic remainder method */
{
   int j, jassign, k;
   float expected;

   if(AVG == 0)
   {
      for(j = 0; j < POPSIZE; j++) choices[j] = j;
   }
   else
   {
      j = 0;
      k = 0;

      /* Assign whole numbers */
      do
      {
         expected = ((oldpop[j].obj)/AVG);
         jassign = expected;
         /* note that expected is automatically truncated */
         fraction[j] = expected - jassign;
         while(jassign > 0)
         {
            jassign--;
            choices[k] = j;
            k++;
         }
         j++;
      }
      while(j < POPSIZE);

      j = 0;
      /* Assign fractional parts */
      while(k < POPSIZE)
      {
         if(j >= POPSIZE) j = 0;
         if(fraction[j] > 0.0)
         {
            /* A winner if true */
            if(flip(fraction[j]))
            {
               choices[k] = j;
               fraction[j] = fraction[j] - 1.0;
               k++;
            }
         }
         j++;
      }
   }
   nremain = POPSIZE - 1;
}


int select()
/* selection using remainder method */
{
   int jpick, slect;

   jpick = rnd(0, nremain);
   slect = choices[jpick];
   choices[jpick] = choices[nremain];
   nremain--;
   return(slect);
}
```

```
/* ************************************************************************ */
                                   /* RANKING SELECTION */
/* ************************************************************************ */


void RANK_POP()
{

double perf;        /* next best perf (for ranking)       */
double rank_max;    /* max number of offspring under ranking */
int best;           /* index of next best structure       */
int i,j;


        /* Assign each structure its rank within the population. */
        /* rank = Popsize-1 for best, rank = 0 for worst         */
        /* Use the Needs_evaluation field to store the rank      */

        /* clear the rank fields */
        for (i=0; i<POPSIZE; i++)
            oldpop[i].Needs_evaluation = 0;

        for (i=0; i < POPSIZE-1; i++)
        {
            /* find the ith best structure */
            best = -1;
            perf = 0.0;
            for (j=0; j<POPSIZE; j++)
            {
                if (oldpop[j].Needs_evaluation == 0 &&
                    (best == -1 || (oldpop[j].obj > perf)))
                {
                    perf = oldpop[j].obj;
                    best = j;
                }
            }
            /* mark best structure with its rank */
            oldpop[best].Needs_evaluation = POPSIZE -1 - i;
        }
        /* normalizer for ranking selection */
        for (int k=0; k<POPSIZE; k++)
        {
         if(oldpop[k].Needs_evaluation >= round(POPSIZE - (perc*POPSIZE)))
           {
           oldpop[k].Needs_evaluation = 2.0;
           }
         else if(oldpop[k].Needs_evaluation <= round(perc*POPSIZE))
           {
           oldpop[k].Needs_evaluation = 0.0;
           }
         else
           {
           oldpop[k].Needs_evaluation = 1.0;
           }
        }
}


void preselect()
/* preselection for ranking method */
{
  int j, jassign, k;
  float expected;

  if(AVG == 0)
  {
    printf(" AVG is zero in ppreselect, ************** \n ");
    for(j = 0; j < POPSIZE; j++) choices[j] = j;
  }
  else
  {
    j = 0;
    k = 0;

    /* Assign whole numbers */
    do
    {
      expected = oldpop[j].Needs_evaluation;
      jassign = expected;
      /* note that expected is automatically truncated */
      fraction[j] = expected - jassign;
      while(jassign > 0)
      {
        jassign--;
        choices[k] = j;
        k++;
      }
      j++;
    }
    while(j < POPSIZE);

    j = 0;
    /* Assign fractional parts */
    while(k < POPSIZE)
    {
      if(j >= POPSIZE) j = 0;
      if(fraction[j] > 0.0)
      {
        /* A winner if true */
        if(flip(fraction[j]))
        {
          choices[k] = j;
          fraction[j] = fraction[j] - 1.0;
          k++;
        }
      }
      j++;
    }
  }
  nremain = POPSIZE - 1;
}


int select()
/* selection using ranking method */
{
  int pick, select;

  pick = rnd(0, nremain);
  slect = choices[pick];
  choices[pick] = choices[nremain];
  nremain--;
  return(select);
}
```

Generation Coordinator

```
void generation(const int iter)
{
/* Create a new generation through select, crossover, and mutation */

/* First we call preselect */
preselect();

int j, mate1, mate2, jcross;

j=0;
do {   /* selection, crossover, and mutation until newpop is filled */

  mate1 = select();
  mate2 = select();

/* Crossover and mutation - mutation is called by the crossover operator */

jcross = crossover(oldpop[mate1].chromosome,oldpop[mate2].chromosome,
     newpop[j].chromosome,newpop[j+1].chromosome);

 /* decode string, evaluate fitness & record parantage for both children */
 map_parms(N_parms,LCHROM,newpop[j].chromosome, parm_arr);
  newpop[j].x1=parm_arr[0].parameter; /* n */
  newpop[j].x2=parm_arr[1].parameter; /* Teta */
 newpop[j].x3=parm_arr[2].parameter; /* h */
 newpop[j].x4=parm_arr[3].parameter; /* Hh */
 newpop[j].obj = (*p) (newpop[j]);
 newpop[j].parent1 = mate1;
 newpop[j].parent2 = mate2;
 newpop[j].xsite = jcross;
 newpop[j].itr_n = iter;

 map_parms(N_parms,LCHROM,newpop[j+1].chromosome, parm_arr);
 newpop[j+1].x1=parm_arr[0].parameter; /* n */
 newpop[j+1].x2=parm_arr[1].parameter; /* Teta */
 newpop[j+1].x3=parm_arr[2].parameter; /* h */
 newpop[j+1].x4=parm_arr[3].parameter; /* Hh */
 newpop[j+1].obj = (*p) (newpop[j+1]);
 newpop[j+1].parent1 = mate1;
 newpop[j+1].parent2 = mate2;
 newpop[j+1].xsite = jcross;
 newpop[j+1].itr_n = iter;
j+=2;
}
 while(j<=POPSIZE);
 speciation(newpop);
}
```

Global variables for LVDT simulations

```
#define maxstring 60
#define max_best 40


int numfiles;

typedef struct indiv {
        int chromosome[maxstring]; /* Genotype = bit position */
        double B;   /* Phenotype(1) -> Length of primary coil */
        double M;   /* Phenotype(2) -> Length of secondary coil */
        double La;  /* Phenotype(3) -> Armature Length */
        double Ri;  /* Phenotype(4) -> Inner radious of coils */
        double Ro;  /* Phenotype(5) -> Outer radious of coils */
        double Fs;  /* Phenotype(6) -> Supply frequency */
        double Bx;   /* Normalized Phenotype(1) -> Normalized Length of primary coil */
        double Mx;   /* Normalized Phenotype(2) -> Normalized Length of secondary coil */
        double Lax;  /* Normalized Phenotype(3) -> Normalized Armature Length */
        double Rix;  /* Normalized Phenotype(4) -> Normalized Inner radious of coils */
        double Rox;  /* Normalized Phenotype(5) -> Normalized Outer radious of coils */
        double Fsx;  /* Normalized Phenotype(6) -> Normalized Supply frequency */
         int Needs_evaluation; /* Used for ranking selection */
        double obj;  /* Objective function value, equivalent to rw_fit */
        double d_rw_fit; /* Parameter to account for environmental niche */
        double I_P, N_P, N_S, AR, RS, IMPD, percent_nlt; /* Dependent Variables */
        double K_1, K_2, ERR1, ERR2;
        double OBJECTIVE;
        int itr_n;  /* Iteration Number */
        double  Partition; /* Partition coef., only used by the Best_List data struc. */
        int parent1, parent2, xsite; /* parents and cross point */
              } individual; /* Represents a design candidate */


typedef struct parms {
        int lparm; /* length of parameter */
        double Ph_typ; /* Phenotype used for sharing */
        double parameter, maxparm, minparm; /* Parameter and range */
              } parameter;


double Vs; /* Supply voltage */
double Nv; /* Number of turns per unit volume */
double Xm; /* Maximum displacement range */
double Lm; /* Maximum possible length of the transducer */
double Rho; /* Resistivity of the coil wire */
double PER; /* Relative permeability of the core */
double AREA; /* cross sectional area of coil wire */
double RES, IMPID; /* D.C. resistance and Impedance of primary coil */
double (*p) (individual&); /* pointer to chosen fitness function */
double Mu; /* Penalty coefficient */
double NLT, SENST; /* user required minimum % non-linearity & maximum sensitivity */
```

Interface Routines for LVDT simulations

```
double Get_Primary_Ip(double L_P,double Rin,double Rout,double Fs,double Nv)
{
\*
Purpose        : Calculates primary excitation current
called by      :objfunc_LVDT
calls          :None
*\

 double Lav, Rdc, N;
 double Ar, I;
 double EXP1, EXP2;
 double Rm,Lc,IMP;


 N = Nv * L_P * 3.1415926 * ( pow(Rout,2.0) - pow(Rin,2.0));
 Lav = 3.1415926 * ( Rin + Rout );
 Ar = ((3.1415926 * (pow(Rout,2.0) - pow(Rin,2.0)) * L_P)/(Lav * N)) * 0.92;
 AREA = Ar;
 Rdc = Rho * 3.1415926 * (N/Ar ) * (Rin + Rout);

 Rm = 0.5 * (Rin + Rout);
 EXP1 = 4.0*(pow(3.1415926,2.0))*(0.0000001)*(pow(N,2.0))*(pow(Rm,2.0));
 EXP2 = sqrt(pow(L_P,2.0)+(4.0*pow(Rm,2.0)));
 Lc = EXP1/EXP2;

 IMP = sqrt(pow(Rdc,2.0)+pow((2.0*3.1415927*Fs*Lc),2.0));
 IMPID = IMP; /* GLOBAL VAR IMPID */

 I = Vs/IMP;
 return(I);
 }


double objfunc_LVDT (individual& ind)
{
/* Objective function, returns cost */
double Ip, K1,K2,Np,Ns,Xo,PER_NLT;
double EXP1, EXP2, EXP3, EXP4,EXP5,EXP6,EXP7,EXP8;
double OP1,OP2,ER1, ER2, ERR,TER1,TER2,TER,OBJ;
double C1,C2,C3,C4;
 EXP7 = C1 = C3 = 0.0;

 if (ind.Ro <= ind.Ri)
   return(0.0);

 Ip = Get_Primary_Ip(ind.B,ind.Ri,ind.Ro,ind.Fs,Nv);
 ind.I_P = Ip; ind.AR = AREA;  ind.IMPD = IMPID;

 Np = Nv * (ind.B) * 3.1415926 * ( pow(ind.Ro,2.0) - pow(ind.Ri,2.0));
 ind.N_P = Np;
 Ns = Nv * (ind.M) * 3.1415926 * ( pow(ind.Ro,2.0) - pow(ind.Ri,2.0));
 ind.N_S = Ns;
 Xo = 0.5 * (ind.La - ind.B );
 if (Xo <0.0) return(0.0);

 EXP1 = 16.0 * pow(3.1415926,3.0) * ind.Fs * Ip * Np * Ns * (ind.B + Xo) * Xo;
 EXP2 = 10000000.0 * log(ind.Ro/ind.Ri) * (ind.M) * ind.La;

 K1 = EXP1/EXP2; ind.K_1 = K1;
 K2 = 1.0/((ind.B + Xo) * Xo); ind.K_2 = K2;

 ERR = K1*Xm*(1.0 - (K2*Xm*Xm));
  if (ERR <0.0)  return(0.0);
 ER1 =(K1*Xm) - ERR;  /* Nonlinearity measure at max displacement w.r.t zero displacement */
 ind.ERR1 = ER1;                          /* This must be minimized */
 PER_NLT = (ER1/(K1*Xm))*100.0; ind.percent_nlt = PER_NLT;
 ER2 = K1 * (1.0 - (K2*(Xm*Xm))); /* Sensitivity measure at max. displacement. This must be maximized */
 ind.ERR2 = ER2;
EXP3 =  Lm - (ind.B + (2.0 * ind.M) );
     if(EXP3 >= 0.0) C1 = 0.0;
      else
     C1 = Mu*pow(EXP3,2.0);
EXP4 = ind.La - ind.B;
              if(EXP4 >= 0.0) C2 = 0.0;
      else
              C2 = Mu*pow(EXP4,2.0);
EXP5 = (ind.B + (2.0 * ind.M) - (2.0 * Xm)) - ind.La;
     if(EXP5 >= 0.0) C3 = 0.0;
      else
     C3 = Mu*pow(EXP5,2.0);
EXP6 = (ind.Ro - ind.Ri);
     if(EXP6 <= 0.001)
      EXP7 = Mu*(0.001-(EXP6))*(0.001-(EXP6));

if(PER_NLT <= NLT)
  OP1 = (PER_NLT / NLT);
else
  OP1 = (NLT / PER_NLT);

if(ER2 <= SENST)
  OP2 = (ER2 / SENST);
else
  OP2 = (SENST / ER2);

OBJ = ((OP1 + OP2)/2.0)-EXP7-C1-C2-C3;
   return (OBJ);
}


double decode (int* chromosome, int lbits)
{
/* Decode string as unsigned binary integer - true=1, false=0 */
int j;
double accum, powerof2;

accum = 0.0; powerof2 = 1.0;

for(j=0; (j<=lbits-1) ; j++)
  {
   if (check(chromosome[j]))
    {
     accum += powerof2;
    }
     powerof2 *= 2;
  }
return accum;
}


double map_parm(double x,double max_parm,double min_parm,double full_scale)
{
 double value;

 value = min_parm+(((max_parm-min_parm)/full_scale)*x);
 return value;
}


void extract_parm(const int*& chromfrom,int *chromto,int& jposition,int lchrom,int lparm)
{
 /* Extracts a sub_string from a full string */
 int j, jtarget;
 j=0;
 jtarget = jposition+lparm-1;

 if(jtarget>lchrom-1)
    jtarget=lchrom-1; /* clamp if excessive */
 while (jposition <= jtarget)
 {
  chromto[j]=chromfrom[jposition];
  j++; jposition++;
 }
}
```

```
void map_parms(int N_Parms,int lchrom,const int *chrom,parameter *par_ar)
{
 int j,jposition;
 double decde, full_sc;
 int chr_temp[maxstring]; /* Temp chrom buffer */
 j=0; /* Parameter counter */
 jposition = 0; /* string position counter */

 for (int k=0; k<N_Parms; k++)
 {
  if(par_ar[k].lparm > 0)
  {
   extract_parm(chrom,chr_temp,jposition,lchrom,(par_ar[k].lparm));
   decde = decode(chr_temp,par_ar[k].lparm);
   full_sc = pow(2.0,(par_ar[k].lparm))-1.0;
   par_ar[k].Ph_typ = decde/full_sc;
   par_ar[k].parameter=map_parm(decde,par_ar[k].maxparm,par_ar[k].minparm,full_sc);
  }
  else
  {
   par_ar[k].parameter=0.0;
  }
 } /* Termination of loop */
}
```

Sharing procedures in conjunction with the LVDT simulations

```
void init_Best_List(individual *B_List)
/*
       purpose       : Initialize the Best_List data structure
       called by     : initpop
       calls         : none
*/
{
  double Part_coef1 = 0.061237244;
  double Part_coef2 = 0.12247449; /*Using partitions  0.12247 apart based on the normalized distance*/
  B_List[0].M = 0.0;
  B_List[0].Partition = Part_coef1;
  B_List[0].obj = 0.0;
  for(int j=1; j<max_best; j++)
  {
   B_List[j].M = 0.0;
   B_List[j].Partition = ((j) * Part_coef2) + Part_coef1; /* Set Partition coef. for each Best_List element */
   B_List[j].obj = 0.0;
  }
}


void Put_Best(const individual* indiv, individual *B_L)
/* ----------------------------------------------------
   Purpose      : Puts a relatively better candidate design in its related partition
   called by    : Statistics
   calls        : Assign_All
*/

{
double d,Dmin; /* relative dist.(d) and minimum rel. dist. (Dmin) */
int index;     /* Partition index to be replaced */

 for(int j=0; j<POPSIZE; j++)
 {
  d = fabs((indiv[j].Lx) - (B_L[0].Partition)); /* init. d and Dmin */
  Dmin = d;
  index = 0;
  for(int x=1; x < max_best; x++) /* Loops through the best list */
  {
   d = fabs((indiv[j].Lx) - (B_L[x].Partition)); /* Find closest partition for indiv[j] */
    if( Dmin > d)
     {
      Dmin = d;
      index = x;
     }
   }
   if((indiv[j].obj)>(B_L[index].obj)) /* If this candidate has higher fitness, -*/
    {                                 /* replace it with the previous candidate. */
     Assign_All(indiv[j], B_L[index]);
    }
  }
}


double share (individual& ind1, individual& ind2)
/* -------------------------------------------
   Purpose      : Compares relative similarity of two individuals
   called by    : Niche
   calls        : none
*/
{
 const double sigm_share = 0.7433466;
 double exxp;
 double dist,sh;

 /* Calculate Normalized distance metric */
 exxp = pow(ind1.Bx-ind2.Bx,2.0)+pow(ind1.Mx-ind2.Mx,2.0)+pow(ind1.Lax-ind2.Lax,2.0)
      +pow(ind1.Rix-ind2.Rix,2.0)+pow(ind1.Rox-ind2.Rox,2.0)+pow(ind1.Fsx-ind2.Fsx,2.0);

 dist = sqrt(exxp);

 if ((dist < sigm_share))
 {
  sh = (1.0 - (dist/sigm_share)); /* Triangular sharing function */
  return (sh);
 }
 else
  return(0.0);
}


double Niche(individual& ind,individual *indivl)
/*
   Purpose      : calculates Niche count for each candidate design
   called by    : speciation
   calls        : share
*/
{
 double count;

 count = 0.0;
 for ( int j=0; j<POPSIZE; j++)
 {
  count += share(ind, (indivl[j]));
 }
 return (count);
}


void speciation( individual *indivl)
/*--------------------------------
   Purpose      : loops and assigns shared fitness (i.e d_rw_fit) to each individual
   called by    : initpop, generation
   calls        : Niche
*/
{
 double niche_count;

 for (int k=0; k< POPSIZE ; k++)
 {
  niche_count = Niche((indivl[k]), indivl);
  indivl[k].d_rw_fit = indivl[k].obj/niche_count;

/* Calculate distance from the origin of the hyperspace */
  indivl[k].Lx = sqrt(pow(indivl[k].Bx,2.0)+pow(indivl[k].Mx,2.0)+pow(indivl[k].Lax,2.0)+
                 pow(indivl[k].Rix,2.0)+pow(indivl[k].Rox,2.0)+pow(indivl[k].Fsx,2.0));
 }
}
```

APPENDIX-III

Result of the Sharing Experiment for multimodal design optimization of a Diaphragm

```
##############################################################################
   SIGMA_SHARE = 6.40,     POP_SIZE = 170,     MAX_GEN = 100,     Pm = 0.001,     Pc = 0.85
##############################################################################


****************************************
Best individual number 1
Genotype = 1111101110111000101000000001000000000011
Phenotype1 (H/h) = 1.062546
Phenotype2 (R) = 56.159495
Fitness = 0.999995
Degraded Fitness = 0.097795
Linear Term of the Characteristic = 0.105995
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  7.583523
Profile Coefficient b  =  1.658391
H = 0.498049
Thickness = 0.468732
error1 = 0.000005
error2 = 0.000000
Terror = 0.000005
Partition = 1.000000
****************************************
Best individual number 2
Genotype = 1001110001111000110000010011111010110001
Phenotype1 (H/h) = 2.244889
Phenotype2 (R) = 37.448157
Fitness = 0.998133
Degraded Fitness = 0.091170
Linear Term of the Characteristic = 0.104129
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  12.762260
Profile Coefficient b  =  0.673436
H = 0.512318
Thickness = 0.228215
error1 = 0.001871
error2 = 0.000000
Terror = 0.001871
Partition = 2.000000
****************************************
Best individual number 3
Genotype = 0111011011111110100100100001010101011011
Phenotype1 (H/h) = 3.083422
Phenotype2 (R) = 30.085911
Fitness = 0.996673
Degraded Fitness = 0.101821
Linear Term of the Characteristic = 0.109338
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  17.832677
Profile Coefficient b  =  0.413188
H = 0.477812
Thickness = 0.154962
error1 = 0.003338
error2 = 0.000000
Terror = 0.003338
Partition = 3.000000
****************************************
Best individual number 4
Genotype = 0101011101001100000100101011110110001011
Phenotype1 (H/h) = 3.740405
Phenotype2 (R) = 23.859126
Fitness = 0.952653
Degraded Fitness = 0.088003
Linear Term of the Characteristic = 0.056300
Cubic Term of the Characteristic = 0.105500
```

```
Profile Coefficient a  =  22.521593
Profile Coefficient b  =  0.300242
H = 0.315489
Thickness = 0.084346
error1 = 0.049700
error2 = 0.000000
Terror = 0.049700
Partition = 4.000000
****************************************
Best individual number 5
Genotype = 0011111001101100110101000111001000010011
Phenotype1 (H/h) = 5.445607
Phenotype2 (R) = 18.973120
Fitness = 0.998230
Degraded Fitness = 0.091936
Linear Term of the Characteristic = 0.107773
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  37.504689
Profile Coefficient b  =  0.155570
H = 0.354482
Thickness = 0.065095
error1 = 0.001773
error2 = 0.000000
Terror = 0.001773
Partition = 5.000000
****************************************
Best individual number 6
Genotype = 0011001010101100100101010101100100000011
Phenotype1 (H/h) = 6.347707
Phenotype2 (R) = 16.664693
Fitness = 0.995330
Degraded Fitness = 0.078679
Linear Term of the Characteristic = 0.110692
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  47.045605
Profile Coefficient b  =  0.117659
H = 0.325163
Thickness = 0.051225
error1 = 0.004692
error2 = 0.000000
Terror = 0.004692
Partition = 6.000000
****************************************
Best individual number 7
Genotype = 0010101000110001010001100001101110111011
Phenotype1 (H/h) = 7.108328
Phenotype2 (R) = 14.998504
Fitness = 0.999746
Degraded Fitness = 0.110519
Linear Term of the Characteristic = 0.106254
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  55.951251
Profile Coefficient b  =  0.095421
H = 0.294600
Thickness = 0.041444
error1 = 0.000254
error2 = 0.000000
Terror = 0.000254
Partition = 7.000000
****************************************
Best individual number 8
Genotype = 0010001000111000100001101100111111011111
Phenotype1 (H/h) = 7.812003
Phenotype2 (R) = 13.432503
Fitness = 0.980996
Degraded Fitness = 0.086767
Linear Term of the Characteristic = 0.086628
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  64.889727
Profile Coefficient b  =  0.079975
H = 0.248516
Thickness = 0.031812
error1 = 0.019372
error2 = 0.000000
Terror = 0.019372
Partition = 8.000000
```

```
****************************************
Best individual number 9
Genotype = 00011010101110101011011000000011000000010011
Phenotype1 (H/h) = 9.094048
Phenotype2 (R) = 11.997770
Fitness = 0.999830
Degraded Fitness = 0.111440
Linear Term of the Characteristic = 0.106170
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  82.900033
Profile Coefficient b  =  0.060000
H = 0.245431
Thickness = 0.026988
error1 = 0.000170
error2 = 0.000000
Terror = 0.000170
Partition = 9.000000
****************************************
Best individual number 10
Genotype = 00011000100010111000100010010000000111111
Phenotype1 (H/h) = 9.563469
Phenotype2 (R) = 11.531739
Fitness = 0.994020
Degraded Fitness = 0.113332
Linear Term of the Characteristic = 0.112016
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  90.050988
Profile Coefficient b  =  0.054511
H = 0.242453
Thickness = 0.025352
error1 = 0.006016
error2 = 0.000000
Terror = 0.006016
Partition = 10.000000
****************************************
Best individual number 11
Genotype = 00001111110000010110100111111101101000000
Phenotype1 (H/h) = 10.981455
Phenotype2 (R) = 9.805072
Fitness = 0.979845
Degraded Fitness = 0.066861
Linear Term of the Characteristic = 0.085430
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  113.461525
Profile Coefficient b  =  0.041813
H = 0.189702
Thickness = 0.017275
error1 = 0.020570
error2 = 0.000000
Terror = 0.020570
Partition = 11.000000
****************************************
Best individual number 12
Genotype = 00000110011010000011101011111110111111001
Phenotype1 (H/h) = 11.995997
Phenotype2 (R) = 7.968623
Fitness = 0.930691
Degraded Fitness = 0.077387
Linear Term of the Characteristic = 0.031530
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  131.879044
Profile Coefficient b  =  0.035254
H = 0.107222
Thickness = 0.008938
error1 = 0.074470
error2 = 0.000000
Terror = 0.074470
Partition = 12.000000
****************************************
Best individual number 13
Genotype = 00001000101010111000110001011001100111111
Phenotype1 (H/h) = 13.350094
Phenotype2 (R) = 8.413167
Fitness = 0.996876
Degraded Fitness = 0.081596
Linear Term of the Characteristic = 0.109134
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  158.626942
Profile Coefficient b  =  0.028646
H = 0.182465
Thickness = 0.013668
error1 = 0.003134
error2 = 0.000000
Terror = 0.003134
Partition = 13.000000
***************************************
Best individual number 14
Genotype = 000001100010101001101100111111000111110
Phenotype1 (H/h) = 13.986310
Phenotype2 (R) = 7.921190
Fitness = 0.990042
Degraded Fitness = 0.085911
Linear Term of the Characteristic = 0.095942
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  172.049315
Profile Coefficient b  =  0.026163
H = 0.164474
Thickness = 0.011760
error1 = 0.010058
error2 = 0.000000
Terror = 0.010058
Partition = 14.000000
***************************************
Best individual number 15
Genotype = 0000001011101010000011100111100100101011
Phenotype1 (H/h) = 15.473326
Phenotype2 (R) = 7.282454
Fitness = 0.999757
Degraded Fitness = 0.105213
Linear Term of the Characteristic = 0.105757
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  205.552542
Profile Coefficient b  =  0.021477
H = 0.158357
Thickness = 0.010234
error1 = 0.000243
error2 = 0.000000
Terror = 0.000243
Partition = 15.000000
***************************************
Best individual number 16
Genotype = 0000001011101111100011101001011010110011
Phenotype1 (H/h) = 15.588683
Phenotype2 (R) = 7.286675
Fitness = 0.993587
Degraded Fitness = 0.097181
Linear Term of the Characteristic = 0.112455
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  208.276350
Profile Coefficient b  =  0.021167
H = 0.162249
Thickness = 0.010408
error1 = 0.006455
error2 = 0.000000
Terror = 0.006455
Partition = 16.000000
***************************************
Best individual number 17
Genotype = 0000001000100110000011111110100100101101
Phenotype1 (H/h) = 16.910858
Phenotype2 (R) = 7.132051
Fitness = 0.933929
Degraded Fitness = 0.126134
Linear Term of the Characteristic = 0.176745
Cubic Term of the Characteristic = 0.105500
Profile Coefficient a  =  240.778414
Profile Coefficient b  =  0.018045
H = 0.189489
Thickness = 0.011205
error1 = 0.070745
error2 = 0.000000
Terror = 0.070745
Partition = 17.000000
```

APPENDIX-IV

Result of the Sharing Experiment for multimodal design optimization of an LVDT

```
Is LVDT being used for sensing DISPLACEMENT ?
You have chosen DISPLACEMENT
Enter Number of turns per unit volume ---> 162.929e+07 turns/m*m*m
Enter Supply voltage ---> 2.7 VOLTS
Enter Maximum displacement range ---> 1.27 mm
Enter Maximum possible length of the transducer ---> 89.0 mm
Enter resistivity of the coil wire ---> 1.8e-08 ohm-meter
Enter seed random number (0.0 .. 1.0) >
randomseed is 0.630591
END of initdata

$$$$$$$$$$$$$$$$$$$$$$ GENETIC ALGORITHM PARAMETERS $$$$$$$$$$$$$$$$$$$$$$

Population size (POPSIZE)          = 400
Chromosome length (LCHROM)         = 60
Maximum number of generation (MAXGEN) = 140
Crossover probability (PCROSS)     = 0.950000
Mutation probability (PMUTATION)   = 0.003000

################ Initial Generation Statistics ################
----------------------------------------------------------------------
  Initial population maximum fitness = 0.635062
  Initial population average fitness = 0.109754
  Initial population minimum fitness = 0.000000
  Initial population sum of fitness  = 43.901594

===============================================================
Results at generation 140 :
===============================================================

Best individual number 1
Genotype = 000000000000000000000000000000000000000000000000000000000000
Phenotype1 (B) = 0.000000
Phenotype2 (M) = 0.000000
Phenotype3 (La) = 0.000000
Phenotype4 (Ri) = 0.000000
Phenotype5 (Ro) = 0.000000
Phenotype6 (Fs) = 0.000000
Percentage range (Bx) = 0.000000
Percentage range (Mx) = 0.000000
Percentage range (Lax) = 0.000000
Percentage range (Rix) = 0.000000
Percentage range (Rox) = 0.000000
Percentage range (Fsx) = 0.000000
(I_P) = 0.000000
(N_P) = 0.000000
(N_S) = 0.000000
(AR) = 0.0000000000
(IMEDANCE) = 0.0000000000
(K_1) = 0.000000
(K_2) = 0.000000
(ERR1) = 0.000000
(ERR2) = 0.000000
objective = 0.000000
DIST = 0.000000
Degraded Fitness = 0.000000
Partition = 0.061237
Iteration number = 0
```

```
************************************
Best individual number 2
Genotype = 000000000000000000000000000000000000000000000000000000000000
Phenotype1 (B) = 0.000000
Phenotype2 (M) = 0.000000
Phenotype3 (La) = 0.000000
Phenotype4 (Ri) = 0.000000
Phenotype5 (Ro) = 0.000000
Phenotype6 (Fs) = 0.000000
Percentage range (Bx) = 0.000000
Percentage range (Mx) = 0.000000
Percentage range (Lax) = 0.000000
Percentage range (Rix) = 0.000000
Percentage range (Rox) = 0.000000
Percentage range (Fsx) = 0.000000
(I_P) = 0.000000
(N_P) = 0.000000
(N_S) = 0.000000
(AR) = 0.0000000000
(IMEDANCE) = 0.0000000000
(K_1) = 0.000000
(K_2) = 0.000000
(ERR1) = 0.000000
(ERR2) = 0.000000
objective = 0.000000
DIST = 0.000000
Degraded Fitness = 0.000000
Partition = 0.183712
Iteration number = 0
************************************
Best individual number 3
Genotype = 0000011001000010001100000000100100011110001001000000000100000
Phenotype1 (B) = 0.001454
Phenotype2 (M) = 0.005082
Phenotype3 (La) = 0.008269
Phenotype4 (Ri) = 0.001010
Phenotype5 (Ro) = 0.002274
Phenotype6 (Fs) = 19.530792
Percentage range (Bx) = 0.015640
Percentage range (Mx) = 0.140762
Percentage range (Lax) = 0.279570
Percentage range (Rix) = 0.001955
Percentage range (Rox) = 0.034213
Percentage range (Fsx) = 0.024438
(I_P) = 25.779174
(N_P) = 30.877409
(N_S) = 107.956725
(AR) = 0.0000000547
(IMEDANCE) = 0.1047357081
(K_1) = 40.436131
(K_2) = 60367.810618
(ERR1) = 0.005000
(ERR2) = 36.498977
objective = 0.535838
DIST = 0.316211
Degraded Fitness = 0.075982
Partition = 0.306186
Iteration number = 135
************************************
Best individual number 4
Genotype = 0010000011100000111110000000000100110101110001100101000000000100
Phenotype1 (B) = 0.001113
Phenotype2 (M) = 0.006726
Phenotype3 (La) = 0.011929
Phenotype4 (Ri) = 0.001010
Phenotype5 (Ro) = 0.002469
Phenotype6 (Fs) = 61.466276
Percentage range (Bx) = 0.003910
Percentage range (Mx) = 0.197458
Percentage range (Lax) = 0.420332
Percentage range (Rix) = 0.001955
Percentage range (Rox) = 0.058651
Percentage range (Fsx) = 0.131965
(I_P) = 24.503232
(N_P) = 28.935626
(N_S) = 174.807761
(AR) = 0.0000000517
(IMEDANCE) = 0.1101895436
(K_1) = 185.762639
(K_2) = 28358.166975
(ERR1) = 0.010791
(ERR2) = 177.266063
objective = 0.457898
DIST = 0.486357
Degraded Fitness = 0.052897
Partition = 0.428661
Iteration number = 22
************************************
Best individual number 5
Genotype = 011000101000000000110001000110010111111000110110010001000001
Phenotype1 (B) = 0.002843
Phenotype2 (M) = 0.007152
Phenotype3 (La) = 0.010709
Phenotype4 (Ri) = 0.001342
Phenotype5 (Ro) = 0.002023
Phenotype6 (Fs) = 160.205279
Percentage range (Bx) = 0.063539
Percentage range (Mx) = 0.212121
Percentage range (Lax) = 0.373412
Percentage range (Rix) = 0.068426
Percentage range (Rox) = 0.002933
Percentage range (Fsx) = 0.385142
(I_P) = 22.701182
(N_P) = 33.364572
(N_S) = 83.939205
(AR) = 0.0000000534
(IMEDANCE) = 0.1189365387
(K_1) = 428.276132
(K_2) = 37524.942392
(ERR1) = 0.032920
(ERR2) = 402.355156
objective = 0.574535
DIST = 0.584375
Degraded Fitness = 0.067548
Partition = 0.551135
Iteration number = 90
************************************
Best individual number 6
Genotype = 001010100100111110100001000100100100111101010101100000011110
Phenotype1 (B) = 0.001850
Phenotype2 (M) = 0.010695
Phenotype3 (La) = 0.016021
Phenotype4 (Ri) = 0.001332
Phenotype5 (Ro) = 0.003955
Phenotype6 (Fs) = 74.428152
Percentage range (Bx) = 0.029326
Percentage range (Mx) = 0.334311
Percentage range (Lax) = 0.577713
Percentage range (Rix) = 0.066471
Percentage range (Rox) = 0.244379
Percentage range (Fsx) = 0.165200
(I_P) = 2.335780
(N_P) = 131.343814
(N_S) = 759.129779
(AR) = 0.0000000340
(IMEDANCE) = 1.1559309114
(K_1) = 292.026108
(K_2) = 15795.718077
(ERR1) = 0.009449
(ERR2) = 284.586184
objective = 0.625611
DIST = 0.733353
```

```
Degraded Fitness = 0.044262
Partition = 0.673610
Iteration number = 8
************************************
Best individual number 7
Genotype = 001100000000111001110000111101101100111101010100100000010101
Phenotype1 (B) = 0.001595
Phenotype2 (M) = 0.010582
Phenotype3 (La) = 0.019274
Phenotype4 (Ri) = 0.001298
Phenotype5 (Ro) = 0.003806
Phenotype6 (Fs) = 83.196481
Percentage range (Bx) = 0.020528
Percentage range (Mx) = 0.330401
Percentage range (Lax) = 0.702835
Percentage range (Rix) = 0.059629
Percentage range (Rox) = 0.225806
Percentage range (Fsx) = 0.187683
(I_P) = 3.148476
(N_P) = 104.552646
(N_S) = 693.494116
(AR) = 0.0000000352
(IMEDANCE) = 0.8575577023
(K_1) = 396.098804
(K_2) = 10842.144933
(ERR1) = 0.008797
(ERR2) = 389.172107
objective = 0.776340
DIST = 0.832666
Degraded Fitness = 0.041817
Partition = 0.796084
Iteration number = 20
************************************
Best individual number 8
Genotype = 001000101100001010110000010010110111101001101100000000010011
Phenotype1 (B) = 0.001539
Phenotype2 (M) = 0.013246
Phenotype3 (La) = 0.023620
Phenotype4 (Ri) = 0.001088
Phenotype5 (Ro) = 0.002336
Phenotype6 (Fs) = 62.991202
Percentage range (Bx) = 0.018573
Percentage range (Mx) = 0.422287
Percentage range (Lax) = 0.869990
Percentage range (Rix) = 0.017595
Percentage range (Rox) = 0.042033
Percentage range (Fsx) = 0.135875
(I_P) = 21.740420
(N_P) = 33.663333
(N_S) = 289.816920
(AR) = 0.0000000525
(IMEDANCE) = 0.1241926350
(K_1) = 384.985586
(K_2) = 7200.395151
(ERR1) = 0.005678
(ERR2) = 380.514549
objective = 0.924728
DIST = 0.977800
Degraded Fitness = 0.064428
Partition = 0.918559
Iteration number = 116
************************************
Best individual number 9
Genotype = 001000111001001110110001101010111100001101101110010000110000
Phenotype1 (B) = 0.002361
Phenotype2 (M) = 0.013501
Phenotype3 (La) = 0.025475
Phenotype4 (Ri) = 0.001518
Phenotype5 (Ro) = 0.004463
Phenotype6 (Fs) = 64.134897
Percentage range (Bx) = 0.046921
Percentage range (Mx) = 0.431085
Percentage range (Lax) = 0.941349
Percentage range (Rix) = 0.103617
Percentage range (Rox) = 0.307918
Percentage range (Fsx) = 0.138807
(I_P) = 1.125831
(N_P) = 212.871772
(N_S) = 1217.467876
(AR) = 0.0000000300
(IMEDANCE) = 2.3982279151
(K_1) = 402.570826
(K_2) = 6216.910128
(ERR1) = 0.005127
(ERR2) = 398.534146
objective = 0.995575
DIST = 1.094985
Degraded Fitness = 0.048145
Partition = 1.041033
Iteration number = 129
************************************
Best individual number 10
Genotype = 001000100000000000000000011110111110101001110010100011011011
Phenotype1 (B) = 0.007208
Phenotype2 (M) = 0.013983
Phenotype3 (La) = 0.026466
Phenotype4 (Ri) = 0.001147
Phenotype5 (Ro) = 0.002000
Phenotype6 (Fs) = 61.847507
Percentage range (Bx) = 0.214076
Percentage range (Mx) = 0.447703
Percentage range (Lax) = 0.979472
Percentage range (Rix) = 0.029326
Percentage range (Rox) = 0.000000
Percentage range (Fsx) = 0.132942
(I_P) = 8.747408
(N_P) = 99.074003
(N_S) = 192.196042
(AR) = 0.0000000571
(IMEDANCE) = 0.3086628729
(K_1) = 402.436681
(K_2) = 6168.028450
(ERR1) = 0.005085
(ERR2) = 398.433075
objective = 0.996530
DIST = 1.106420
Degraded Fitness = 0.055047
Partition = 1.163508
Iteration number = 64
************************************
Best individual number 11
Genotype = 001000111101111111100100011100111110000001110000000010111101
Phenotype1 (B) = 0.006358
Phenotype2 (M) = 0.013700
Phenotype3 (La) = 0.026212
Phenotype4 (Ri) = 0.002388
Phenotype5 (Ro) = 0.005988
Phenotype6 (Fs) = 64.516129
Percentage range (Bx) = 0.184751
Percentage range (Mx) = 0.437928
Percentage range (Lax) = 0.969697
Percentage range (Rix) = 0.277615
Percentage range (Rox) = 0.498534
Percentage range (Fsx) = 0.139785
(I_P) = 0.123766
(N_P) = 981.372198
(N_S) = 2114.688093
(AR) = 0.0000000215
(IMEDANCE) = 21.8153598111
(K_1) = 402.575421
(K_2) = 6185.687762
(ERR1) = 0.005101
```

```
(ERR2) = 398.558968
objective = 0.998111
DIST = 1.229378
Degraded Fitness = 0.042053
Partition = 1.285982
Iteration number = 133
***********************************
Best individual number 12
Genotype = 001000111100001011100000000110111111001011101001000011110110
Phenotype1 (B) = 0.007974
Phenotype2 (M) = 0.027420
Phenotype3 (La) = 0.026670
Phenotype4 (Ri) = 0.001029
Phenotype5 (Ro) = 0.002360
Phenotype6 (Fs) = 64.516129
Percentage range (Bx) = 0.240469
Percentage range (Mx) = 0.911046
Percentage range (Lax) = 0.987292
Percentage range (Rix) = 0.005865
Percentage range (Rox) = 0.044966
Percentage range (Fsx) = 0.139785
(I_P) = 4.058844
(N_P) = 184.019474
(N_S) = 632.822149
(AR) = 0.0000000530
(IMEDANCE) = 0.6652139571
(K_1) = 403.749947
(K_2) = 6175.802431
(ERR1) = 0.005108
(ERR2) = 399.728213
objective = 0.998916
DIST = 1.372652
Degraded Fitness = 0.062120
Partition = 1.408457
Iteration number = 62
***********************************
Best individual number 13
Genotype = 001000101111100110111000000010111100001101110110100000010001
Phenotype1 (B) = 0.001482
Phenotype2 (M) = 0.014437
Phenotype3 (La) = 0.025475
Phenotype4 (Ri) = 0.003512
Phenotype5 (Ro) = 0.009218
Phenotype6 (Fs) = 62.991202
Percentage range (Bx) = 0.016618
Percentage range (Mx) = 0.463343
Percentage range (Lax) = 0.941349
Percentage range (Rix) = 0.502444
Percentage range (Rox) = 0.902248
Percentage range (Fsx) = 0.135875
(I_P) = 0.095978
(N_P) = 550.961549
(N_S) = 5367.513933
(AR) = 0.0000000141
(IMEDANCE) = 28.1314851006
(K_1) = 404.144598
(K_2) = 6184.451716
(ERR1) = 0.005120
(ERR2) = 400.113295
objective = 0.997929
DIST = 1.478535
Degraded Fitness = 0.057894
Partition = 1.530931
Iteration number = 120
***********************************
Best individual number 14
Genotype = 001000000111001111101110111100111100010001110110010000010100
Phenotype1 (B) = 0.001567
Phenotype2 (M) = 0.014409
Phenotype3 (La) = 0.025500
Phenotype4 (Ri) = 0.005673
```

```
Phenotype5 (Ro) = 0.008491
Phenotype6 (Fs) = 59.178886
Percentage range (Bx) = 0.019550
Percentage range (Mx) = 0.462366
Percentage range (Lax) = 0.942326
Percentage range (Rix) = 0.934506
Percentage range (Rox) = 0.811339
Percentage range (Fsx) = 0.126100
(I_P) = 0.133588
(N_P) = 320.138520
(N_S) = 2943.756575
(AR) = 0.0000000127
(IMEDANCE) = 20.2113519195
(K_1) = 403.924030
(K_2) = 6174.558820
(ERR1) = 0.005109
(ERR2) = 399.901373
objective = 0.999018
DIST = 1.627765
Degraded Fitness = 0.056089
Partition = 1.653406
Iteration number = 30
***********************************
Best individual number 15
Genotype = 0010000110111111111111110111001111100111101101100010001101011
Phenotype1 (B) = 0.004033
Phenotype2 (M) = 0.013275
Phenotype3 (La) = 0.025780
Phenotype4 (Ri) = 0.005658
Phenotype5 (Ro) = 0.010000
Phenotype6 (Fs) = 61.085044
Percentage range (Bx) = 0.104594
Percentage range (Mx) = 0.423265
Percentage range (Lax) = 0.953079
Percentage range (Rix) = 0.931574
Percentage range (Rox) = 1.000000
Percentage range (Fsx) = 0.130987
(I_P) = 0.024804
(N_P) = 1403.580815
(N_S) = 4619.638261
(AR) = 0.0000000115
(IMEDANCE) = 108.8540875974
(K_1) = 405.307007
(K_2) = 6169.560168
(ERR1) = 0.005122
(ERR2) = 401.273844
objective = 0.996254
DIST = 1.727264
Degraded Fitness = 0.064866
Partition = 1.775880
Iteration number = 18
***********************************
Best individual number 16
Genotype = 0010001111111111011101110100010111111101011111000010100000111
Phenotype1 (B) = 0.008456
Phenotype2 (M) = 0.029150
Phenotype3 (La) = 0.026873
Phenotype4 (Ri) = 0.005545
Phenotype5 (Ro) = 0.009867
Phenotype6 (Fs) = 64.516129
Percentage range (Bx) = 0.257087
Percentage range (Mx) = 0.970674
Percentage range (Lax) = 0.995112
Percentage range (Rix) = 0.909091
Percentage range (Rox) = 0.983382
Percentage range (Fsx) = 0.139785
(I_P) = 0.012268
(N_P) = 2882.750434
(N_S) = 9937.990512
(AR) = 0.0000000117
(IMEDANCE) = 220.0794909954
```

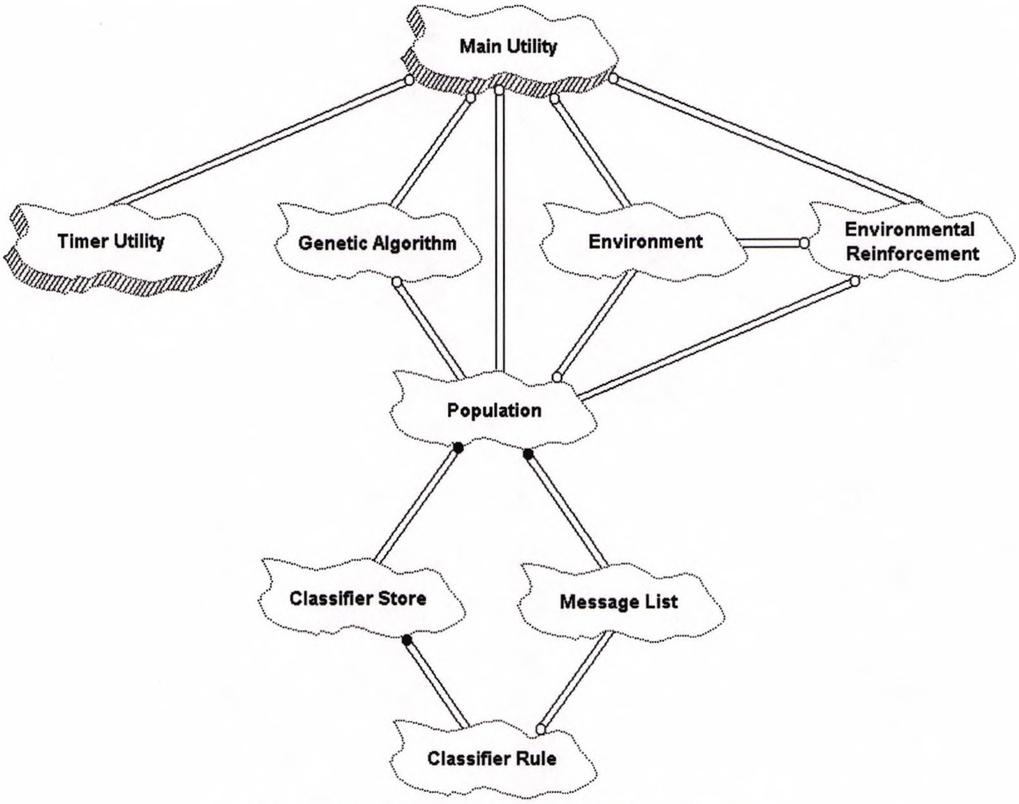
```
(K_1) = 405.399277
(K_2) = 6147.621830
(ERR1) = 0.005105
(ERR2) = 401.379541
objective = 0.997785
DIST = 1.952326
Degraded Fitness = 0.050804
Partition = 1.898355
Iteration number = 75
************************************
Best individual number 17
Genotype = 001000100011111111111111001110111110101011110110010011000100
Phenotype1 (B) = 0.006556
Phenotype2 (M) = 0.028923
Phenotype3 (La) = 0.026466
Phenotype4 (Ri) = 0.005761
Phenotype5 (Ro) = 0.010000
Phenotype6 (Fs) = 61.847507
Percentage range (Bx) = 0.191593
Percentage range (Mx) = 0.962854
Percentage range (Lax) = 0.979472
Percentage range (Rix) = 0.952102
Percentage range (Rox) = 1.000000
Percentage range (Fsx) = 0.132942
(I_P) = 0.015215
(N_P) = 2242.254677
(N_S) = 9891.729743
(AR) = 0.0000000114
(IMEDANCE) = 177.4622886428
(K_1) = 403.091883
(K_2) = 6083.836230
(ERR1) = 0.005023
(ERR2) = 399.136496
objective = 0.991405
DIST = 1.961462
Degraded Fitness = 0.073864
Partition = 2.020829
Iteration number = 52
************************************
Best individual number 18
Genotype = 111100110111111110111001000111111110100011111010111010000110
Phenotype1 (B) = 0.019313
Phenotype2 (M) = 0.029433
Phenotype3 (La) = 0.026415
Phenotype4 (Ri) = 0.003849
Phenotype5 (Ro) = 0.009969
Phenotype6 (Fs) = 380.938416
Percentage range (Bx) = 0.631476
Percentage range (Mx) = 0.980450
Percentage range (Lax) = 0.977517
Percentage range (Rix) = 0.569892
Percentage range (Rox) = 0.996090
Percentage range (Fsx) = 0.951124
(I_P) = 0.001903
(N_P) = 8358.786366
(N_S) = 12738.930885
(AR) = 0.0000000130
(IMEDANCE) = 1418.9449911848
(K_1) = 420.274233
(K_2) = 12315.602064
(ERR1) = 0.010602
(ERR2) = 411.925975
objective = 0.726038
DIST = 2.130069
Degraded Fitness = 0.069649
Partition = 2.143304
Iteration number = 81
************************************
Best individual number 19
Genotype = 111000101111111110111110010011111111111111110100011011010100
Phenotype1 (B) = 0.021524
```

```
Phenotype2 (M) = 0.028696
Phenotype3 (La) = 0.027000
Phenotype4 (Ri) = 0.005472
Phenotype5 (Ro) = 0.009969
Phenotype6 (Fs) = 355.777126
Percentage range (Bx) = 0.707722
Percentage range (Mx) = 0.955034
Percentage range (Lax) = 1.000000
Percentage range (Rix) = 0.894428
Percentage range (Rox) = 0.996090
Percentage range (Fsx) = 0.886608
(I_P) = 0.002084
(N_P) = 7649.338932
(N_S) = 10198.192183
(AR) = 0.0000000116
(IMEDANCE) = 1295.8645595125
(K_1) = 410.094422
(K_2) = 15053.461682
(ERR1) = 0.012645
(ERR2) = 400.137441
objective = 0.701483
DIST = 2.234107
Degraded Fitness = 0.078705
Partition = 2.265778
Iteration number = 139
************************************
Best individual number 20
Genotype = 1111011010111111110001111110010111111101011110011101101000000
Phenotype1 (B) = 0.024586
Phenotype2 (M) = 0.028611
Phenotype3 (La) = 0.026873
Phenotype4 (Ri) = 0.005936
Phenotype5 (Ro) = 0.009945
Phenotype6 (Fs) = 385.894428
Percentage range (Bx) = 0.813294
Percentage range (Mx) = 0.952102
Percentage range (Lax) = 0.995112
Percentage range (Rix) = 0.987292
Percentage range (Rox) = 0.993157
Percentage range (Fsx) = 0.963832
(I_P) = 0.001839
(N_P) = 8011.974587
(N_S) = 9323.783714
(AR) = 0.0000000113
(IMEDANCE) = 1468.5407858425
(K_1) = 195.030373
(K_2) = 33983.102115
(ERR1) = 0.013576
(ERR2) = 184.340495
objective = 0.418255
DIST = 2.334154
Degraded Fitness = 0.043909
Partition = 2.388253
Iteration number = 114
************************************
```

APPENDIX-V

The Software System Architecture of the Classifier System

The implemented class diagram of the classifier system is shown below:



Class Diagram of the Classifier System

Please note that, detector, effector and creditor classes have been included as methods of the Classifier System (represented by the population class). For more sophisticated applications, it is prefered to represent these modules as separate class objects. This supports further modularity and ease of future expansions of the software.

The most important object templates and their message operations are documented below:

1- Population Object Template:

Name: Population Object

Documentation: Represents the Classifier System instance with its appropriate set of parameters and data. It has appropriate methods for:

1-Its initialization.
2-Initial report and reports of its status at iteration instances specified by the user.
3-Detection of the design environment using its detectors method.

4-Rule and Environmental Message-List matching (using Match_Classifiers method).
5-Invocation of the Apportionment of Credit Algorithm (via invocation of its auction method).
6-Effecting its design environment by calling its effector method.

Class: Population

Persistence: Dynamic

2- Genetic Algorithm Object Template:

Name: Genetic Algorithm object instance name.

Documentation: Encapsulates an instance of the Genetic Algorithm parameters and methods. It is instantiated via the Main utility and contains methods for the initial report of its instantiated parameters and field classes. It, also, reports, at specific intervals (as set by the user), the statistical and parametric information resulting from the application of the genetic operations.

The genetic algorithm is applied to the classifier system at specific intervals (as set by the user). It is supported by the galg(population & pop) method of the genetic algorithm object instance and is invoked via the main utility. The overall functionality of this method (which uses other private methods including select, crowding, crossover, and mutation), using pseudo code, is:

```
BEGIN
  1-Statistics (population & pop);
   {Calculates population statistics:
    MAX, AVG, MIN, and sum_of_strength for roulete wheel (Monte Carlo) selection.
   }
   REPEAT
      2-Select Mate1
      3-Select Mate2
      4-Crossover(Mate1, Mate2);
        {Uses mutation methods during the transfer of strings}
      5-Crowding(child1, population & pop);
        {Uses DeJong's crowding method to maintain a diverse population of rules.
         The classifier-rule index for replacement by child1 is
         returned by using this method.}
      6-Crowding(child2, population & pop);
      7-Insert child1 and child2 in place of classifier-rule members, as specified by
         Crowding method invocations.
      8-Update population statistics.
   UNTIL (enough mates are piked according to CF parameter)
END;
```

Class: GA class

Persistence: Dynamic

3-Reinforcement Object Template:

Name: The name of the reinforcement object instance.

Documentation: A particular reinforcement object instance monitors the performance of the classifier system and pays instantiated reward for correct answers. This functionality is provided by the invocation of the reinf(environment & env, population & pop) method within the main utility.

The overall functionality of this method, using pseudocode, is:

```
BEGIN
    if (Criterion (environment &))
    { Criterion is a private method invoked within the reinf(environment &,
     population &), which compares the classifier system's answer to a particular
     design problem with the correct answer, as contained and simulated within the
     environment object instance (in the form of a mathematical model). It returns
     TRUE or FALSE based on correct or wrong classifier system responses.
    }
    payreward (population & pop);
    { Pay instantiated reward to the winner classifier rule}
END;
```

The reinforcement object instance, also, has appropriate report methods which are invoked at simulation intervals (as set by the user). These reports (including graphical and statistical information) provide the user with information regarding the rate of improvement of the classifier system behaviour, and are vital for analysis of the classifier system performance.

Class: Reinforcement class

Persistence: Dynamic

4- Environment Object Template:

Name: The name of the environment object instance.

Documentation: An environmental object instance represents the design environment to the classifier system. This functionality is supported by the Coordinate(); method. The overall functionality of this method, using pseudocode, is:

```
BEGIN
    1-generate_signal();
    {In our application, at this stage, this private method produces two random sub-
     strings. The overall concatenated mapped fixed point string represents the
     design problem encountered by the classifier system.
    }
    2-Correct_Action();
    {The environmental object instance, using its knowledge of the Diaphragm-
     Mathematical Model, and its generated design signal, works out the correct
     design response by using the error_range(H/h, rr) routine. This information is
     used by the reinforcement object instance for the evaluation of classifier system
    performance.
   }
END;
```

The environment object instance, also, possesses methods for the initialization of its atributes and class fields. It, also, has methods that report the current status of the environmental object instance in terms of its design signal, decoded design parameters, correct design action and further statistical information.

The program listing of the classifier system is given in the following pages.

```
//                          Main Program


main(int argc, char** argv)
{

 int numfiles = argc - 1;

  if(numfiles == 0)
   {
    infp = stdin;
    outfp = stdout;
    prop_rew = stdout;
    prop_rew_50 = stdout;
   }

 if(numfiles == 4)
  {
   if((infp = fopen(argv[1],"r")) == NULL)
     {
       fprintf(stderr,"Cannot open input file %s\n",argv[1]);
       exit(-1);
     }
   if((outfp = fopen(argv[2],"w")) == NULL)
     {
       fprintf(stderr,"Cannot open output file %s\n",argv[2]);
       exit(-1);
     }
   if((prop_rew = fopen(argv[3],"w")) == NULL)
     {
       fprintf(stderr,"Cannot open output file %s\n",argv[3]);
       exit(-1);
     }
   if((prop_rew_50 = fopen(argv[4],"w")) == NULL)
     {
       fprintf(stderr,"Cannot open output file %s\n",argv[4]);
       exit(-1);
     }
  }


 int i=0;
 initrnd();
 population pop(NCLASS,NPOS);
 environment env;
 reinforcement reinf;
 GA gag(MARSIZE);
 timer tme;

 pop.init_population();
 pop.init_report_classifiers();

 env.initenvironment();
 env.initrepenvironment();
 pop.repmatchlist();

 reinf.initreinforcement();
 reinf.initrepreinforcement();

 gag.initga(pop);
 gag.initrepga();

 tme.inittimer();
 tme.initreptimer();

 pop.detectors(env);

 fprintf(outfp," Console Report \n");
 fprintf(outfp,"-----------------\n");
 env.reportenvironment();
 pop.reportdetectors();
 pop.report_classifiers();
 pop.repmatchlist();
 pop.reportaoc();
 reinf.reportreinforcement();
 reinf.plot_reportreinforcement();
while(i<MAX_ITER)
{
 tme.time();
 env.coordinate();
 pop.detectors(env);
 pop.matchclassifiers();
 pop.aoc();
 pop.effector(env);
 reinf.reinf(env,pop);

 if(tme.get_gaflag())
 {
  gag.galg(pop);
 }
 if(tme.get_plotrepflag())
 {
  reinf.plot_reportreinforcement();
 }


 if(tme.get_reportflag()) {
     fprintf(outfp, "**********************\n");
     fprintf(outfp, "Iteration number is :\n");
     tme.reporttime();
     fprintf(outfp, "**********************\n");
     fprintf(outfp, "Console Report \n");
     fprintf(outfp, "------------------\n");
      gag.reportga(pop);
     env.reportenvironment();
     pop.reportdetectors();
     pop.report_classifiers();
     pop.repmatchlist();
     pop.reportaoc();
     reinf.reportreinforcement();
 }

 pop.advance();
 i++;
}
fprintf(outfp,"Console Report \n");
fprintf(outfp, "------------------\n");
env.reportenvironment();
pop.reportdetectors();
pop.report_classifiers();
pop.repmatchlist();
pop.reportaoc();
reinf.reportreinforcement();
pop.advance();
}
```

// Classifier System Classes and Methods

```
class classifier_rule {
 IntArray cond;
 int action;
 double strength, bid, ebid;
 int matchflag;
 int specificity;
 friend class population;
public:
 classifier_rule ()
              { }
 void init_classifier(int n_position);
 void count_specificity();
 double& Bid()
     {return bid;}
 double& Ebid()
     {return ebid;}
 int& get_matchflag()
    { return matchflag;}
 int& get_action()
    { return action;}
 int get_int_act()
    { return action;} //See matchcount
 int get_int_cond(int indx)
    { return (cond[indx]);} //See matchcount
 int get_cond_size()
    { return (cond.getSize());}
 int& get_cond(int indx)
    { return (cond[indx]);}
 double getstr()
    { return strength;}
 double& get_strength()
    { return strength;}
};


void classifier_rule::init_classifier(int n_position)
{
 cond=n_position;
     bid=0.0;
     ebid=0.0;
     matchflag=0;
     strength = 1.97;
     action = 679;
}


void classifier_rule::count_specificity()
{
 int npos = cond.getSize();
 int temp=0;
 for (int i=0; i<npos; i++)
 {
  if (cond[i]!=2)
     ++temp;
 }
 specificity = temp;
}


class classifier_store {
 int size;
 classifier_rule *rules;
public:
 classifier_store (int sz,int npos);
 ~classifier_store() { delete rules; }
 void rangecheck(int ix);
 classifier_rule& operator[] (int);
 int getSize() { return size; }
};


classifier_store::classifier_store(int sz,int npos) {
 size = sz;
 rules = new classifier_rule[size];
 for(int i =0; i < size; i++)
 rules[i].init_classifier(npos);
}


void classifier_store::rangecheck(int ix)
{
 if ((ix<0) || (ix >= size))
    {
    cerr << "Index out of bounds for classifier_store :"
         << "\n\t size: " << size
         << "\t index : " << ix << "\n";
          exit(17);
    }
}


classifier_rule& classifier_store::operator[] (int index)
{
 rangecheck(index);
 return rules[index];
}
```

```
class population {
 classifier_store store;
 int nclassifiers;
 IntArray matchlist;
 IntArray envmessage;
 int n_position;
 double pgeneral, cbid, bidsigma;
 double bidtax, lifetax;
 double bid1, bid2, ebid1, ebid2;
 double sumstrength, maxstrength;
 double avgstrength, minstrength;
 int winner, oldwinner;
 int bucketbrigadeflag;
 friend class reinforcement;
 friend class GA;
public:
 population (int sz,int nposition)
        :store(sz,nposition),matchlist(0)
 {
  n_position=nposition;
  nclassifiers=sz;
  winner=0;
  oldwinner=0;
  sumstrength=maxstrength=0.0;
  avgstrength=minstrength=0.0;
 }
void repmatchlist();
void init_population();
void read_classifiers();
void init_report_classifiers();
void report_classifiers();
void initrepaoc();
void reportaoc();
void detectors(environment& env);
void reportdetectors();
void writemessage();
void effector(environment& env);
void matchclassifiers();
int match(IntArray& c, int npos);
int auction();
void taxcollector();
void clearinghouse();
void aoc();
void advance();
};


void population::repmatchlist()
{
 fprintf(outfp, "Matchlist = ");
  for(int j=0; j<matchlist.getSize(); j++)
     fprintf(outfp,"%d-", matchlist[j]);
     fprintf(outfp,"\n");
}


void population::init_population()
{
 fprintf(outfp, "inside init_classifiers \n");
 fprintf(outfp, "Enter pgeneral\n");
 fscanf(infp,"%lf",&pgeneral);
 fprintf(outfp, "Enter cbid\n");
 fscanf(infp,"%lf", &cbid);
 fprintf(outfp,"Enter bidsigma\n");
 fscanf(infp,"%lf", &bidsigma);
 fprintf(outfp,"Enter bidtax\n");
 fscanf(infp,"%lf", &bidtax);
 fprintf(outfp,"Enter lifetax\n");
 fscanf(infp,"%lf", &lifetax);
 fprintf(outfp, "Enter bid1\n");
 fscanf(infp,"%lf", &bid1);
 fprintf(outfp, "Enter bid2\n");
 fscanf(infp,"%lf", &bid2);
 fprintf(outfp,"Enter ebid1\n");
 fscanf(infp,"%lf", &ebid1);
 fprintf(outfp,"Enter ebid2\n");
 fscanf(infp,"%lf", &ebid2);
 fprintf(outfp, "Enter bucketbrigadeflag\n");
 fscanf(infp,"%lf", &bucketbrigadeflag);
 fprintf(outfp,"Enter classifiers \n");
 read_classifiers();
 }


void population::read_classifiers()
{
 char ch;
 double s;
 fprintf(outfp,"Enter Strength \n");
 fscanf(infp,"%lf", &s);
 for(int j=0; j< nclassifiers; j++)
  {
   store[j].strength=s;
   for (int i=0; i<store[j].cond.getSize(); i++)
    {
    fscanf(infp,"%1s", &ch);
    switch(ch) {
                case '0':
                     store[j].cond[i]=0;
                       fprintf(outfp, "in zero \n");
                       break;
              case '1':
                     store[j].cond[i]=1;
                       fprintf(outfp, " In one \n");
                       break;
              case '#':
                     store[j].cond[i]=2;
                       fprintf(outfp, "In two \n");
                       break;
              case 'R':
                     store[j].cond[i]=randomchar(pgeneral);
                       fprintf(outfp, "In R \n");
                       break;
                default:
                       store[j].cond[i]=6;
                       fprintf(outfp, " inside default \n");
                       break;
                };
     }
    fscanf(infp,"%1s", &ch);
        if (ch == '1')
        store[j].action=1;
        else if (ch == '0')
        store[j].action=0;
        else
        store[j].action=6;
        store[j].count_specificity();
   }
 }


void population::init_report_classifiers()
{
 //Initial report on population parameters
 fprintf(outfp, "\n");
 fprintf(outfp, "Population Parameters \n");
 fprintf(outfp, "----------------------\n");
 fprintf(outfp, "Number of Classifiers = %d \n",nclassifiers);
 fprintf(outfp,"Number of Positions = %d \n", n_position);
 fprintf(outfp, "Generality Probability = %f \n",pgeneral);
 fprintf(outfp,  "Bid Coefficient = %f \n",cbid);
 fprintf(outfp, "Bid Spread = %f \n",bidsigma);
 fprintf(outfp, "Bidding tax = %f \n",bidtax);
 fprintf(outfp, "Existence Tax = %f \n",lifetax);
 fprintf(outfp, "Bid Specificity (M1) = %f \n",bid1);
 fprintf(outfp, "Bid Specificity Multiplier (M2) = %f \n",bid2);
 fprintf(outfp, "Effective Bid Specificity  = %f \n",ebid1);
 fprintf(outfp, "Effective Bid Specificity Multiplier = %f \n",ebid2);;
 initrepaoc();
 }
```

```
void population::initrepaoc()
{
 fprintf(outfp, "Approportion of Credit Parameters\n");
 fprintf(outfp, "--------------------------------\n");
 fprintf(outfp, "Bucket Brigade Flag   = ");
   if(bucketbrigadeflag)
    fprintf(outfp, "TRUE\n");
   else
    fprintf(outfp, "FALSE\n");
}


void population::reportaoc()
{
 fprintf(outfp, "New winner [%d] : oldwinner [%d] \n",winner,oldwinner);
}


void population::report_classifiers()
{
fprintf(outfp, "\n");
fprintf(outfp, "No     strength     bid      ebid     M    Classifier \n");
fprintf(outfp, "---------------------------------------------------------- \n");
 for(int j=0; j<nclassifiers; j++)
 {
 fprintf(outfp,"%d    %f    %f     %f ",j,store[j].strength,store[j].bid,store[j].ebid);
 if (store[j].matchflag)
   fprintf(outfp,"X  ");
 else
   fprintf(outfp, "   ");
  {
   for(int i=0; i<store[j].cond.getSize() ; i++)
   fprintf(outfp,"%d", store[j].cond[i]);
   fprintf(outfp,"->%d \n", store[j].action);
   fprintf(outfp,"Specificity for this classifier is: %d\n",store[j].specificity);
  }
 }
}


int population::auction()
{
 double bidmaximum=0.0;
 winner=oldwinner; //if no match old winner wins again
 if (matchlist.getSize() > 0)
  {
   int k;
   for(int j=0;j<matchlist.getSize();j++)
    {
     k=matchlist[j];
     store[k].bid = cbid*(bid1+bid2*(store[k].specificity))*(store[k].strength);
     store[k].ebid=cbid*(ebid1+ebid2*(store[k].specificity))*(store[k].strength)
    +noise(0.0,bidsigma);

     if ((store[k].ebid)>bidmaximum)
       {
        winner=k;
        bidmaximum=store[k].ebid;
       }
    }
  }
//  matchlist=0;
 return (winner);
}


void population::taxcollector()
{
 //Collects head and bidding taxes from population members
 double bidtaxswitch;
 //Take  head tax from all rules and bidding taxes from active rules
 if((lifetax != 0.0)||(bidtax != 0.0))
  {
   for(int j=0; j<nclassifiers; j++)
    {
     if (store[j].matchflag)
        bidtaxswitch=1.0;
     else
```

```
        bidtaxswitch=0.0;
     store[j].strength=store[j].strength-(lifetax*store[j].strength)-
                    (bidtax*bidtaxswitch*store[j].strength);
    }
  }
}


void population::clearinghouse()
{
 //Distribute payment from recent winner to old winner
 double payment;
 payment = store[winner].bid;
 store[winner].strength-=payment;
 if (bucketbrigadeflag) //Pay oldwinner receipt if BB is on
    store[oldwinner].strength+=payment;
}


void population::aoc()
{
 //apportionment of credit coordinator
 winner = auction();
 taxcollector();
 clearinghouse();
}


void population::detectors(environment& env)
{
 //converts environmental state to env. message
 envmessage=env.signal;
}


void population::effector(environment& env)
{
 env.classifieroutput=store[winner].action;
}


void population::reportdetectors()
{
 fprintf(outfp, "\n");
 fprintf(outfp,"Environmental message = ");
    writemessage();
 fprintf(outfp,"\n");
}


void population::writemessage()
{
 for (int j=0; j<n_position; j++)
    fprintf(outfp,"%d",envmessage[j]);
}


void population::matchclassifiers()
{
 matchlist = 0;
 for(int j=0; j<nclassifiers; j++)
  {
   store[j].get_matchflag()=match(store[j].cond, store[j].cond.getSize());
   if (store[j].matchflag)
    matchlist.add(j);
  }
}


int population::match(IntArray& c, int npos)
{
 int matchtemp = 1;
 npos-=1;
 while (matchtemp && (npos>-1))
  {
   matchtemp = ((c[npos] == 2)||(c[npos]==envmessage[npos]));
   npos-=1;
  }
```

```
return(matchtemp);
}


void population::advance()
{
 //register oldwinner for the next cycle
 oldwinner = winner;
}


//                  The  environment class, methods, and mathematical model calculations


double ap (double Hh)
{
 const double k1 = 1.011775539;
 const double kinv = 0.98836151;
 double k2, alph, a;
 double exp1,exp2;

 k2 = (((Hh*Hh)*k1)+kinv);
 alph = sqrt(k1*k2);
 exp1 = (2.0*(3.0 + alph)*(1.0 + alph));
 exp2 = (3.0*k1*(1-(0.09/(alph*alph))));
 a = (exp1/exp2);
 return(a);
}


double bp(double Hh)
{
 const double k1 = 1.011775539;
 const double kinv = 0.98836151;
 double k2, alph, b;
 double exp3,exp4;

 k2 = (((Hh*Hh)*k1)+kinv);
 alph = sqrt(k1*k2);
 exp3 = (32*k1)/((alph*alph)-9.0);
 exp4 = (1.0/6.0)-(2.7/((alph-0.3)*(alph+3.0)));
 b = exp3 * exp4;
 return(b);
}


double np (double Hh, double rr)
{
 const double k1=1.011775539;
 const double kinv = 0.98836151;
 double k2, alph, res;
 double exp1,exp2,exp3;
 double exp4,exp5;

 k2=(((Hh*Hh)*k1)+kinv);
 alph = sqrt(k1*k2);

 exp1=(3.0-alph)*(1.0-alph);
 exp2=(3.0+pow(alph,2.0))*(1.0-pow(rr,4.0));
 exp3=(4.0*alph)/(1-pow(rr, 2.0*alph));
 exp4=2.0*pow(rr, alph+1.0)*(1.0+pow(rr,2.0));
 exp5=(1.0+pow(rr, 2.0*alph))*(1.0+pow(rr,4.0));

 res = exp1/(exp2+(exp3*(exp4-exp5)));
 return(res);
}


double fp (double Hh, double rr)
{
 const double k1=1.011775539;
 const double kinv = 0.98836151;
 double k2, alph, res;
 double exp1, exp2, exp3, expp1;
 double expp2, expp3, exp4, exp5;
 double expp4, exp6, exp7,expp5;
 double exppp1, exppp2;

 k2=(((Hh*Hh)*k1)+kinv);
 alph = sqrt(k1*k2);
 exp1 = pow((1.0-pow(rr,2.0)),4.0);
 exp2 = exp1*(1.0+pow(rr,2.0));
 exp3 = (1.0/6.0)-(2.7/((alph-0.3)*(alph+3.0)));
 expp1 = 1.0/(exp2*exp3);

 expp2 = (1.0-pow(rr, 6.0))/6.0;

 expp3 = (2.7/(1.0-pow(rr, 2.0*alph)));

 exp4 = pow((1-pow(rr, alph+3.0)), 2.0);
 exp5 = (alph-0.3)*(3.0+alph);
 expp4 = exp4/exp5;

 exp6 = pow((pow(rr,alph)-pow(rr, 3.0)), 2.0);
 exp7 = (alph + 0.3)*(3.0-alph);
 expp5=exp6/exp7;

 exppp1 = expp4 + expp5;
 exppp2 = expp3*exppp1;
 res = expp1*(expp2-exppp2);
 return (res);
 }


int error_range (double Hh, double rr)
{
 double error;
 double AP,NP,BP,FP;
 const double k1=1.011775539;
 const double kinv = 0.98836151;
 double k2, alph;
 const double wh = 6.0;
 double output1,output2;

 k2=(((Hh*Hh)*k1)+kinv);
 alph = sqrt(k1*k2);
 AP=ap(Hh);
 NP=np(Hh,rr);
 BP=bp(Hh);
 FP=fp(Hh,rr);
 output1 = (AP*NP*wh)+(BP*FP*wh*wh*wh);
 output2 = (AP*wh)+(BP*wh*wh*wh);
 error = (output1-output2)/(output1);

 if (error <= ACCER)
   return(0);
 else
   return(1);
 }



class environment {
 int lparam1, lparam2, lsignal;
 int output, classifieroutput;
 int x1,x2; // decoded value of param.s in unsigned int
 double Hh, rr; //Corresponding values of design parm.s.
 double ACCER; // Acceptable range of error
 int one, zero;
 IntArray signal;
 friend class population;
 friend class reinforcement;
public:
 void initenvironment();
 void initrepenvironment();
 void writesignal();
 void reportenvironment();
 void generatesignal();
 void Correct_Action();
 int decode(int parm, int len);
 void coordinate();
 };
```

```
void environment::initenvironment()
{
 fprintf(outfp, "Enter acceptable range of error \n");
 fscanf(infp,"%lf", &ACCER);
 fprintf(outfp, "Enter lparam1 \n");
 fscanf(infp,"%d", &lparam1);
 fprintf(outfp, "Enter lparam2 \n");
 fscanf(infp,"%d", &lparam2);
 lsignal = lparam1+lparam2;
 fprintf(outfp,"Inside env.h, lsignal is %d \n",lsignal);
 // generatesignal1()
 signal = lsignal;
 Hh=rr=0.0;
 x1=x2=output=0;
 zero=one=0;
 classifieroutput=0;
}


void environment::initrepenvironment()
{
 fprintf(outfp,"\n");
 fprintf(outfp,"Environmental Parameters \n");
 fprintf(outfp, "--------------------------------------\n");
 fprintf(outfp,"lparam1 (Hh) %d \n",lparam1);
 fprintf(outfp,"lparam2 (RR) %d \n",lparam2);
 fprintf(outfp, "Total length of signal %d \n",lsignal);
}


void environment::writesignal()
{
 for(int j=0; j<lsignal; j++)
   fprintf(outfp,"%d", signal[j]);
}


void environment::reportenvironment()
{
 fprintf(outfp,"\n");
 fprintf(outfp,"Current Environmental Status \n");
 fprintf(outfp,"---------------------------\n");
 fprintf(outfp, "Signal  = ");
   writesignal();
 fprintf(outfp, "\n");
 fprintf(outfp,"Decoded param1 = %d, Hh = %f\n",x1,Hh);
 fprintf(outfp,"Decoded param2 = %d, RR = %f\n",x2,rr);
 fprintf(outfp, "Correct output = %d \n",output);
 fprintf(outfp,"Number of ones = %d, Number of zeros = %d\n",one,zero);
 fprintf(outfp,"Classifier output = %d \n",classifieroutput);
}


void environment::generatesignal()
{
 for(int j=0; j<lsignal; j++)
 {
  if (flip(0.5))
   signal[j]=1;
  else
   signal[j]=0;
 }
}


void environment::Correct_Action()
{
 x1=decode(lparam1,3);
 Hh=((16.0/7.0)*x1)+1.0;
 x2=decode(lparam1+lparam2,3);
 rr=(1.0/7.0)*x2;

 if (error_range(Hh,rr))
  {
  output=1;
  ++one;
  }
 else
```

```
  {
  output=0;
  ++zero;
  }
}


int environment::decode(int parm,int len)
{
 int acc, pw2, n;
 acc=0.0; pw2=1.0; n=0;

 for(int j=parm-1;j!=-1; j--)
 {
  ++n;
  acc+=(pw2*signal[j]);
  pw2*=2;
  if(n==len)
   break;
 }
 return(acc);
}


void environment::coordinate()
{
 generatesignal();
 Correct_Action();
}



//                              The Genetic Algorithm Classes and Methods


const int maxmating=10;

class marriage_record {
 int mate1, mate2;
 int mort1, mort2, sitecross;
 friend class GA;
public:
 marriage_record()
                {}
 void initialize()
  {
   mate1=mate2=0;
   mort1=mort2=sitecross=0;
  }
};


class marry_store {
 int sze;
 marriage_record *marry_rec;
public:
 marry_store(int size);
 ~marry_store() {delete marry_rec;}
 void range_check(int idx);
 marriage_record& operator[] (int index);
};


marry_store::marry_store(int size)
{
 sze = size;
 marry_rec = new marriage_record[sze];
 for(int i =0; i < sze; i++)
  marry_rec[i].initialize();

}


void marry_store::range_check(int idx)
{
 if ((idx<0) || (idx >= sze))
    {
```

```
    cerr << "Index out of bounds for marry_store :"
         << "\n\t size: " << sze
         << "\t index : " << idx << "\n";
          exit(17);
   }
}


marriage_record& marry_store::operator[] (int index)
{
 range_check(index);
 return marry_rec[index];
}


class GA
{
 marry_store marr_str; //mating record for genetic algorithm report
 double proportionselect, pmutation, pcrossover;
 int ncrossover, nmutation, crowdingfactor;
 int crowdingsubpop, nselect;
public:
 GA(int size)
   : marr_str(size)
   { } //Modify later for children
 void initga(population& pop);
 void initrepga();
 void reportga(population& pop);
 void galg(population& pop);
 void statistics(population& pop);
 int select(population& pop);
 int crossover(classifier_rule& parent1,classifier_rule& parent2,
             classifier_rule& c1_rule,classifier_rule& c2_rule);
 int bmutation(const int& ac);
 int mutation (const int& trit);
 int crowding(classifier_rule& c_rule,population& pop);
 int worst(population& pop);
 int matchcount(classifier_rule& c_rule,classifier_rule& member);
};


void GA::initga(population& pop)
{
 //Initialize genetic algorithm parameters
 fprintf(outfp,"Enter proportionselect \n");
 fscanf(infp,"%lf",&proportionselect);
 fprintf(outfp,"Enter pmutation \n");
 fscanf(infp,"%lf",&pmutation);
 fprintf(outfp,"Enter pcrossover \n");
 fscanf(infp,"%lf",&pcrossover);
 fprintf(outfp,"Enter crowdingfactor \n");
 fscanf(infp,"%d",&crowdingfactor);
 fprintf(outfp,"Enter crowdingsubpop \n");
 fscanf(infp,"%d",&crowdingsubpop);

 //Number of mate pairs to select
 nselect = round((proportionselect*(pop.nclassifiers)*0.5));
 nmutation = 0;
 ncrossover = 0;
}


void GA::initrepga()
{
 //Initial report
 fprintf(outfp,"GA parameters \n");
 fprintf(outfp,"-------------\n");
 fprintf(outfp,"Proportion select = %f \n",proportionselect);
 fprintf(outfp,"Number of selections = %d \n",nselect);
 fprintf(outfp,"Probability of mutation = %f \n",pmutation);
 fprintf(outfp,"Probability of crossover = %f \n",pcrossover);
 fprintf(outfp,"Crowding factor = %d \n",crowdingfactor);
 fprintf(outfp,"Crowding subpop = %d \n",crowdingsubpop);
}
```

```
void GA::reportga(population& pop)
{
 //Reports mating, crossover and replacement
 fprintf(outfp,"\n");
 fprintf(outfp,"Genetic Algorithm Report\n");
 fprintf(outfp,"-----------------------\n");
 fprintf(outfp,"\n");
 fprintf(outfp,"Pair  Mate1  Mate2  Cross_Site  Dead1  Dead2 \n");
 fprintf(outfp,"----------------------------------------\n");
 for (int j=0; j<nselect; j++) //******
 {
  fprintf(outfp,"%d  %d  %d  %d  %d  %d
\n",j,marr_str[j].mate1,marr_str[j].mate2,marr_str[j].sitecross,marr_str[j].mort1,marr_str[j].mort2);
 }
 fprintf(outfp,"Statistics Report \n");
 fprintf(outfp,"---------------\n");
 fprintf(outfp,"Average Strength = %f \n",pop.avgstrength);
 fprintf(outfp, "Maximum Strength = %f \n",pop.maxstrength);
 fprintf(outfp,"Minimum Strength = %f \n", pop.minstrength);
 fprintf(outfp, "Sum of Strength = %f \n",pop.sumstrength);
 fprintf(outfp,"Number of crossovers = %d \n",ncrossover);
 fprintf(outfp,"Number of mutations = %d \n",nmutation);
}


void GA::galg(population& pop)
{
 //Coordinates genetic operations
 classifier_rule child1,child2;

 child1.init_classifier(pop.n_position);
 child2.init_classifier(pop.n_position);

 statistics(pop);

 for(int j=0; j<nselect; j++)
 {
  marr_str[j].mate1=select(pop);
  marr_str[j].mate2=select(pop);
  int x = marr_str[j].mate1;
  int y = marr_str[j].mate2;

  marr_str[j].sitecross=crossover(pop.store[x],pop.store[y],child1,child2);

  marr_str[j].mort1=crowding(child1,pop);
  //Update sumstrength
  int w=marr_str[j].mort1;
  pop.sumstrength=pop.sumstrength-pop.store[w].getstr()+child1.getstr();
  pop.store[w]=child1; //Insert child in Dead1's place

  marr_str[j].mort2=crowding(child2,pop);
  //Update sumstrength
  int z=marr_str[j].mort2;
  pop.sumstrength=pop.sumstrength-pop.store[z].getstr()+child2.getstr();
  pop.store[z]=child2; //Insert child in Dead2's place
 }
}


int GA::select(population& pop)
{
 //Select a rule using roulete wheel selection

 double rand,partsum;
 int j;
 partsum =0.0; j=-1;
 rand = trandom()*pop.sumstrength;
 do
  {
   ++j;
   partsum+=pop.store[j].get_strength();
  } while((partsum < rand)&&(j != pop.nclassifiers-1));
  //Return selected member's index
  return(j);
}
```

```
int GA::crossover(classifier_rule& parent1,classifier_rule& parent2,
              classifier_rule& c1_rule,classifier_rule& c2_rule)
{
double inheritance;
int sitecrss;

 if (flip(pcrossover))
 {
  sitecrss = rnd(1,parent1.get_cond_size());
  ++ncrossover;
 }
 else sitecrss = (parent1.get_cond_size())+1; // i.e. transfer but no cross
 //transfer action part
 c1_rule.get_action()=bmutation(parent1.get_action());
 c2_rule.get_action()=bmutation(parent2.get_action());
//Transfer and cross above crossover site
 for(int j=sitecrss-1;j<parent1.get_cond_size();j++)
 {
  c2_rule.get_cond(j)=mutation(parent1.get_cond(j));
  c1_rule.get_cond(j)=mutation(parent2.get_cond(j));
 }
//Transfer only below crossover site
 for(int k=0;k<sitecrss-1;k++)
 {
  c1_rule.get_cond(k)=mutation(parent1.get_cond(k));
  c2_rule.get_cond(k)=mutation(parent2.get_cond(k));
 }
 //Children inherit average of  their parents strength values
 inheritance=avg(parent1.getstr(),parent2.getstr());
 c1_rule.get_strength()=inheritance;
 c1_rule.get_matchflag()=0;
 c1_rule.Bid()=0.0;
 c1_rule.Ebid()=0.0;
 c1_rule.count_specificity();

 c2_rule.get_strength()=inheritance;
 c2_rule.get_matchflag()=0;
 c2_rule.Bid()=0.0;
 c2_rule.Ebid()=0.0;
 c2_rule.count_specificity();

 return(sitecrss);
}


int GA::bmutation(const int& ac)
{
 //Mutate a single action bit with specified probability
 int tmpmut;

 if (flip(pmutation))
    {
     tmpmut=(ac+1)%2;
     ++nmutation;
    }
 else
   tmpmut = ac;
  return(tmpmut);
}


int GA::mutation(const int& trit)
{
 //Mutate a single condition-bit with specified probability
 int tempm;

 if (flip(pmutation))
  {
   tempm = (trit+rnd(1,2))%3;
   ++nmutation;
  }
  else
   tempm=trit;
  return(tempm);
}
```

```
int GA::crowding(classifier_rule& c_rule,population& pop)
{
// Replacement using DeJong's crowding method
 int popmember,match,matchmax,mostsimilar;
 matchmax=-1;
 mostsimilar=0;

 if(crowdingfactor<1)
   crowdingfactor=1;

 for(int j=1; j<=crowdingfactor;j++)
 {
  popmember=worst(pop); //Pick worst of n
  match = matchcount(c_rule, pop.store[popmember]);

  if(match > matchmax)
   {
    matchmax=match;
    mostsimilar = popmember;
   }
 }
return(mostsimilar);
}


int GA::worst(population& pop)
{
 //Select worst individual from random subpopulation
 int wrst, candidate;
 double worststrength;

 wrst=rnd(0,pop.nclassifiers-1);
 worststrength=pop.store[wrst].getstr();

 if(crowdingsubpop > 1)
 {
  for(int j=2; j<=crowdingsubpop; j++)
   {
    candidate=rnd(0,pop.nclassifiers-1);
    if(worststrength>pop.store[candidate].getstr())
     {
      wrst=candidate;
      worststrength=pop.store[wrst].getstr();
     }
   }
 }
 return(wrst);
}


int GA::matchcount(classifier_rule& c_rule,classifier_rule& member)
{
 //Count number of  similar positions
 int tmpcnt;

  if ((c_rule.get_int_act())==(member.get_int_act()))
   {
    tmpcnt=1;
   }
  else
    tmpcnt=0;

   for(int j=0;j<c_rule.get_cond_size();j++) //*****!!!!!!!
    if(c_rule.get_int_cond(j)==member.get_int_cond(j))
      ++tmpcnt;
   return (tmpcnt);
}


void GA::statistics(population& pop)
{
 //Population statistics
 pop.maxstrength=pop.store[0].get_strength();
 pop.minstrength=pop.store[0].get_strength();
 pop.sumstrength=pop.store[0].get_strength();

 for(int j=1; j<pop.nclassifiers;j++)
 {
  pop.maxstrength=max(pop.maxstrength,pop.store[j].getstr());
  pop.minstrength=min(pop.minstrength,pop.store[j].getstr());
  pop.sumstrength+=pop.store[j].getstr();
 }
 pop.avgstrength=pop.sumstrength/pop.nclassifiers;
}
```

// **The Reinforcement Class and its methods**

```
class reinforcement
{
 double reward, rewardcount, totalcount, count50;
 double rewardcount50, proportionreward, proportionreward50;
 int lastwinner;
public:
 reinforcement();
 void initreinforcement();
 void initrepreinforcement();
 void reportreinforcement();
 void plot_reportreinforcement();
 int criterion(environment& env);
 void payreward(population& pop);
 void reinf(environment& env, population& pop);
};


reinforcement::reinforcement()
{
 reward = 0.0;
 rewardcount = 0.0;
 totalcount = 0.0;
 count50 = 0.0;
 rewardcount50 = 0.0;
 proportionreward = 0.0;
 proportionreward50 = 0.0;
 lastwinner = 0;
}


void reinforcement::initreinforcement()
{
 fprintf(outfp,"Enter Reward >\n");
 fscanf(infp,"%lf",&reward);
}


void reinforcement::initrepreinforcement()
{
 fprintf(outfp,"\n");
 fprintf(outfp,"Reinforcement Parameters \n");
 fprintf(outfp,"------------------------\n");
 fprintf(outfp," Reward  = %f \n",reward);
}


void reinforcement::plot_reportreinforcement()
{
 fprintf(prop_rew,"%f %f \n",totalcount,proportionreward);
 fprintf(prop_rew_50,"%f %f \n",totalcount,proportionreward50);
}


void reinforcement::reportreinforcement()
{
 //Report reward
  fprintf(outfp,"\n");
  fprintf(outfp,"Reinforcement Report\n");
  fprintf(outfp, "--------------------\n");
  fprintf(outfp, "Proportion correct (from start) = %f \n",proportionreward);
  fprintf(outfp, "Proportion correct (from last fifty) = %f \n",proportionreward50);
  fprintf(outfp,  "Last winning classifier = %d \n",lastwinner);
}
```

```
int reinforcement::criterion(environment& env)
{
 int flag;
 if(env.output == env.classifieroutput)
 flag = 1;
 else
 flag=0;
 totalcount+=1;
 count50+=1;
 if (flag)
  {
   rewardcount+=1.0;
   rewardcount50+=1.0;
  }
 //Calculate reward proportions: running and last 50
 proportionreward = rewardcount/totalcount;
 if((roundd(count50-50.0))==0)
   {
    proportionreward50=rewardcount50/50.0;
    rewardcount50=0.0; count50=0.0; //reset
   }
 return(flag);
}


void reinforcement::payreward(population& pop)
{
 //pay reward to appropriate individual
 int k = pop.winner;
 pop.store[k].get_strength()+=reward;
 lastwinner = pop.winner;
}


void reinforcement::reinf(environment& env, population& pop)
{
 if(criterion(env))
  payreward(pop);
}



//                                  The Timer Class utility


const int iterationsperblock = 10000;


class timer{
 int initialiteration, initialblock, iteration, block;
 int reportperiod, gaperiod, plotrepperiod;
 int nextreport, nextga, nextplotreport;
 int reportflag, gaflag, plotrepflag;
public:
 timer()
      { }
 void inittimer();
 void initreptimer();
 void time();
 void reporttime();
 int addtime(int t, int dt, int& carryflag);
 int get_reportflag()
     { return(reportflag);}
 int get_gaflag()
     { return(gaflag);}
 int get_plotrepflag()
     { return(plotrepflag); }
};


void timer::inittimer()
{
 //initializes timer
 int dummy;
 iteration=block=0;
 fprintf(outfp,"Enter initialiteration \n");
 fscanf(infp,"%d",&initialiteration);
 fprintf(outfp,"Enter initialblock \n");
 fscanf(infp,"%d",&initialblock);
 fprintf(outfp,"Enter reportperiod \n");
 fscanf(infp,"%d",&reportperiod);
 fprintf(outfp,"Enter gaperiod \n");
 fscanf(infp,"%d",&gaperiod);
 fprintf(outfp,"Enter plotrepperiod \n");
 fscanf(infp,"%d",&plotrepperiod);
 iteration=initialiteration;
 block=initialblock;
 nextga=addtime(iteration,gaperiod, dummy);
 nextreport=addtime(iteration,reportperiod,dummy);
 nextplotreport = addtime(iteration,plotrepperiod,dummy);
}


void timer::initreptimer()
{
 //Initial timer report
 fprintf(outfp,"\n");
 fprintf(outfp,"Timer Parameters \n");
 fprintf(outfp,"------------------\n");
 fprintf(outfp,"Initial iteration = %d \n",initialiteration);
 fprintf(outfp,"Initial block     = %d \n",initialblock);
 fprintf(outfp,"Report period     = %d \n",reportperiod);
 fprintf(outfp,"Genetic algorithm period  = %d \n",gaperiod);
 fprintf(outfp,"Plot Period = %d \n", plotrepperiod);
}


void timer::time()
{
 //registers time and sets flags for user specified events
 int carryflag=0;
 int dummyflag=0;

  iteration = addtime(iteration, 1, carryflag);
  if (carryflag)
   ++block;
  reportflag = (nextreport == iteration);
  if (reportflag)
    nextreport = addtime(iteration, reportperiod, dummyflag);
  gaflag = (nextga == iteration);
  if (gaflag)
    nextga = addtime(iteration, gaperiod, dummyflag);
  plotrepflag = (nextplotreport == iteration);
  if ( plotrepflag)
    nextplotreport = addtime(iteration,plotrepperiod,dummyflag);
}


void timer::reporttime()
{
 // print out block and iteration
  fprintf(outfp,"Block = %d Iteration = %d \n",block,iteration);
}


int timer::addtime(int t, int dt, int& carryflag)
{
 int tempadd;

 tempadd = t+dt;
 carryflag = (tempadd >= iterationsperblock);

 if (carryflag)
  tempadd = (tempadd % iterationsperblock);
 return(tempadd);
}
```

//   Utilitv routines and the integer arrav class utilitv

```
const int ArraySize = 24;  //default size

class IntArray {
public:
// operations performed on arrays
  IntArray (int sz = ArraySize);
  IntArray (const IntArray&);
  ~IntArray() {delete ia;}
  IntArray& operator=(const IntArray&);
  IntArray& operator=(int szz);
  int& operator[](int);
  void add(int x);
  void rangeCheck(int index);
  int getSize() {return size;}
protected:
//internal data representation
  int size;
  int *ia;
};


IntArray::IntArray(int sz)
{
 size = sz;

//allocate an integer array of size
//and set ia to point to it
ia = new int[size];

//initialize array
for (int i = 0; i<sz; ++i)
   ia[i] = 0;
}


IntArray::IntArray(const IntArray &iA)
{
 size = iA.size;
 ia = new int[size];
 for (int i = 0; i<size; ++i)
   ia[i] = iA.ia[i];
}


IntArray& IntArray::operator=(const IntArray &iA)
{
 delete ia; //free up existing memory
 size = iA.size; //resize target
 ia = new int[size]; // get new memory
 for (int i=0; i<size; ++i)
  ia[i] = iA.ia[i]; //copy
 return *this;
}


IntArray& IntArray::operator=(int szz)
{
 delete ia;
 size = szz;
 ia = new int[size];
 for (int i=0; i<size; ++i)
  ia[i] = 0;
 return *this;
}


void IntArray::add(int x)
{
 int* oldia = ia;
 int oldsize = size;
 size+=1;
 ia = new int[size];
 // copy elements of the old array into new array
for (int i=0; i<oldsize; ++i)
```

```
 ia[i]=oldia[i];
// init remaining elements to x
for (;i<size; ++i)
 ia[i]=x;
 delete oldia;
}


int& IntArray::operator[](int index)
{
 rangeCheck (index);
 return ia[index];
}


void IntArray::rangeCheck (int index)
{
 if  (index < 0 || index >= size) {
  cerr << "index out of bounds for IntArrayRC: "
        <<  "\n\tsize: "<< size
        <<  "\tindex: " << index "\n";
  exit(17);
 }
}


int randomchar(double pgen)
{
 if (flip(pgen))
   return(2);
 else if (flip(0.5))
   return (1);
 else
   return (0);
}


void initrandomnormaldeviate()
/* initialization routine for randomnormaldeviate */
{
   rndcalcflag = 1;
}


double randomnormaldeviate()
/* random normal deviate after ACM algorithm 267 / Box-Muller Method */
{
   double t, rndx1;

   if(rndcalcflag)
   {
      rndx1 = sqrt(- 2.0*log((double) trandom()));
      t = 6.2831853072 * (double) trandom();
      rndx2 = sin(t);
      rndcalcflag = 0;
      return(rndx1 * cos(t));
   }
   else
   {
      rndcalcflag = 1;
      return(rndx2);
   }
}


double noise(double mu ,double sigma)
/* normal noise with specified mean & std dev: mu & sigma */
{
   return((randomnormaldeviate()*sigma) + mu);
}


int roundd (double x)
{
 int integer;
 double fraction;
 integer=(int)(x/1);
 fraction = (x-integer);
 if(fraction <= 0.5)
   return(x/1);
 else
   return((x/1)+1);
}


double avg(double x,double y)
{
 //Returns average
 return((x+y)/2.0);
}


double max(double x, double y)
{
 //returns maximum
 if(x>y)
  return(x);
 else
  return(y);
}


double min(double x, double y)
{
 //returns minimum
 if(x<y)
  return(x);
 else
  return(y);
}
```

//   The Pseudorandom Number Generator

```
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349
int seed;


float ran1(int& idum)
{
 static long ix1, ix2, ix3;
 static float r[98];
 float temp;
 static int iff=0;
 int j;
 //void nrerror();

 if(idum < 0 || iff == 0)
 {
  iff=1;
  ix1 = (IC1-(idum))%M1;
  ix1 = (IA1*ix1+IC1)%M1;
  ix2 = ix1%M2;
  ix1 = (IA1*ix1+IC1)%M1;
  ix3 = ix1%M3;

   for(j=1; j<=97;j++)
   {
    ix1 = (IA1*ix1+IC1)%M1;
    ix2 = (IA2*ix2+IC2)%M2;
    r[j] = (ix1+ix2*RM2)*RM1;
   }
  idum = 1;
 }
 ix1=(IA1*ix1+IC1)%M1;
 ix2=(IA2*ix2+IC2)%M2;
 ix3=(IA3*ix3+IC3)%M3;
 j = 1+((97*ix3)/M3);

 if(j>97 || j<1)
 {
  fprintf(outfp,"RAN1: This cannot happen.\n");
  exit(1);
 }
 temp = r[j];
 r[j]=(ix1+ix2*RM2)*RM1;
 return temp;
}


double gasdev(int& idum)
{
 //Returns a normally distributed deviate with zero mean and unit variance,
 //using random() as the source of uniform deviates.
 static int iset=0;
 static double gset;
 double fac, r, v1, v2;

 if (iset == 0) //We don't have an extra deviate handy, so
 {
  do {
     v1=2.0*ran1(idum)-1.0; //Pick two uniform numbers in the square -
     v2=2.0*ran1(idum)-1.0; //extending from -1 t0 +1 in each direction
     r = v1*v1+v2*v2; //See if they are in the unit circle,
    } while (r >= 1.0 || r== 0.0); //and if they are not try again
    fac=sqrt(-2.0*log(r)/r);
  //Now make the Box-Muller transformation to get two normal deviates.
  //Return one and save the other for next time.
    gset=v1*fac;
    iset = 1; //Set flag
    return (v2*fac);
  } else {
    iset=0;  //We have an extra deviate handy, so unset the flag.
    return (gset);
  }
}


double noise(double mu,double sigma, int& p)
{
//  cout << "P is = " << p << "\n";
 //Normal noise with specified mean & std dev: mu & sigma
 return (gasdev(p)*sigma+mu);
}


void initrnd()
{
 fprintf(outfp,"Enter seed \n");
 fscanf(infp,"%d", &seed);
 float z = ran1(seed);
 fprintf(outfp,"In rnd.h, Init z is = %f \n",z);
}
```