Goldberg, David Edward

COMPUTER-AIDED GAS PIPELINE OPERATION USING GENETIC ALGORITHMS AND RULE LEARNING

The University of Michigan PH.D. 1983

COMPUTER-AIDED GAS PIPELINE OPERATION

USING

GENETIC ALGORITHMS AND RULE LEARNING

by

David Edward Goldberg

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Civil Engineering)
in The University of Michigan
1983

Doctoral Committee:

Professor John H. Holland, Co-chairman
Professor E. Benjamin Wylie, Co-chairman
Professor Jonathan W. Bulkley
Assistant Professor Nikolaos D. Katopodes
Associate Professor Steven J. Wright

RULES REGARDING THE USE OF

MICROFILMED DISSERTATIONS

Microfilmed or bound copies of doctoral dissertations submitted to The University of Michigan and made available through University Microfilms International or The University of Michigan are open for inspection, but they are to be used only with due regard for the rights of the author. Extensive copying of the dissertation or publication of material in excess of standard copyright limits, whether or not the dissertation has been copyrighted, must have been approved by the author as well as by the Dean of the Graduate School. Proper credit must be given to the author if any material from the dissertation is used in subsequent written or published work.

For Nary

ACKNOWLEDGEMENTS

I gratefully acknowledge the steadfast support and assistance of my committee co-chairmen, Professors E. B. Wylie and J. H. Holland. It is most fitting, considering the topic of this dissertation, that I met both men through randomized, inference-guided searches at two different decision junctions in my career.

I met Ben Wylie nine years ago while choosing a specialty within Civil Engineering in "something analytical where one uses computers a lot." Ben was on duty that day and suggested that hydraulics might be the ticket. It was, and it is. Since that time, I have benefited from, and thoroughly enjoyed, his imaginative instruction, keen insight, and gentle guidance through a maze of academic and career choices.

I met John Holland while searching for a thesis topic that might connect control, artificial intelligence, and pipelines. His course, "Introduction to Adaptive Systems," seemed to contain some interesting clues, but I wasn't sure how all this biological stuff could help. Since then, I have come to appreciate, not only the fundamental soundness of his position, but also the breadth and depth of understanding which led him there.

I thank the rest of the committee, Professors J. W. Bulkley, N. D. Katapodes, and S. J. Wright, for their time, suggestions, and support. My contact with these people has made this project as enjoyable as it has been stimulating. I also acknowledge Doug Ward's workmanlike programming on the steady serial model, and I appreciate Carol Miller's comments on draft chapters.

Gary Black of Union Gas Ltd., Phil Isola of Michigan-Wisconsin Pipeline, and Chuck Crider of Colonial Pipeline were instrumental in getting me out of the ivory tower and into the field. I thank them and the many operations people I met during my visits.

This work has been partially supported under U. S. Department of Energy contract DE-FG02-80ER10125. I acknowledge and appreciate this support as well as financial assistance from the M. W. Goodrich Fund, the H. W. King Fund, the Rackham School of Graduate Studies, and the U. M. Office of Energy Research.

Regardless of where one stands on the age-old question of environment vs. heredity, a good deal of credit for this work belongs to my parents. I thank them for implanting the seeds of learning within me and nurturing those seeds so well.

Finally, I am beholden to my best friend and spouse, Mary Ann, for many things. With my single question, she left a secure job, home, family, and friends for a place with none of these. She created a new home, a new career, a

new life, against the long odds of a deep recession. Her energy spurred me on when mine was gone. She had confidence in me when I had none. For these things and so much more, I dedicate this work to her with all my love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

Transmission of gas by pipeline is a vital commercial activity. It is also a somewhat tricky business. The gas dispatcher, the human operator, must balance supply and demand under uncertain circumstances through proper sequencing of equipment that is both expensive to run and maintain. Although computer decision aids are entering the gas dispatching task, these aids have not proven sufficiently capable to face the changing environment autonomously. Furthermore, many existing computer algorithms require large amounts of computational horsepower; the human dispatcher makes the same decision with relative ease. As a result, gas dispatching, like most complex technical tasks, still relies heavily upon the human operator's experience and savvy.

The goals of this study are the development of robust decision-making and learning algorithms for gas pipeline operations. To achieve the desired breadth of behavior, we abandon the traditional techniques of optimization and control theory in favor of methods associated with genetics and artificial intelligence. We adopt these methods to more

closely match the human learning and decision-making experience, because ultimately, we seek algorithms capable of autonomous pipeline control.

In the remainder of this chapter, the scope, methodology, and benefits of this research are detailed. We conclude the chapter with a summary and an overview of the sequel.

1.1 Scope of Research

TIME: 7:00 AM, Monday morning, January 198-

PLACE: Central Dispatch Center
Central Iowa Gas Transmission Company

"Well, Joe, how's it look?"

"Don't know yet. Weather service predicts a cold one. Front moving through."

"Gee, how we gonna make it with the number 2 unit out at Lorraine?"

"Might be tough. Think we can get by if we run the standby and push a bit harder upstream. That worked pretty good last year at Downing. Things at Lorraine are usually a little less hairy anyway."

This fictional account of a dispatcher's informal decision shows the complexity and breadth of knowledge required in the simplest day-to-day operating decisions. In the brief exchange, the two operators 1) recognize and evaluate the severity of the situation, 2) size up a response, and 3) reinforce and informally prove strategy success by citing a complex inequality relation. Note that this whole process is seemingly effortless, a product of complex intuitive thought.

If successful algorithms for learning and decision-

making are to be developed, they must possess skills of similar breadth as our fictional dispatchers. The primary measure of this quality is robustness. Paraphrasing Holland [1], a robust system is one which is efficient over the range of environments it may encounter. A robust pipeline operator (human or artificial) must be capable of learning and making decisions under widely varied circumstances. This requires satisfactory adaptation to normal and abnormal conditions alike. It further suggests that some decision be made in situations which may or may not be completely familiar (an educated guess).

The efficiency required of a robust system is demonstrated in the effortlessness of human intuitive decisions; this type of efficiency is also desirable in an artificial decision maker. The computational intensity of many artificial decision procedures suggests that there is room for a good deal of improvement.

1.2 Basic Approach

To achieve the desired breadth and effortlessness of behavior, we apply computer techniques connected with genetics and artificial intelligence to the pipeline operations problem, proceeding in two separate steps.

First, we apply genetic algorithms (GA) [1] to two problems in pipeline optimization. Genetic algorithms seek improved performance by combining some string manipulations similar to the mechanics of natural genetics and a survival-of-the-fittest mechanism. This application demonstrates the

efficacy of genetic algorithms as a search procedure in practical engineering problems; it also ties the present work to existing pipeline operation and control literature.

This experience leads us to the second step of our study: the development of a learning classifier system (LCS) for pipeline control. Briefly, a learning classifier system is a learning system that learns rules to improve its performance in some arbitrary environment. The LCS uses behavioral reinforcement (reward) and environmental information to guide its learning and decision making. Our experiments with genetic algorithms are quite useful here, because a genetic algorithm serves as one of two major learning mechanisms in the LCS. We expect the system to be capable of learning and responding to varied normal and abnormal operations alike.

Why this Approach?

While this work represents a departure from more traditional methods of optimization and control, it is not a case of being unconventional for its own sake.

Traditional methods of optimization and control suffer from two major shortcomings. First, they most often employ local search procedures. Local techniques are, by definition, myopic; they cannot see the forest for the trees, unless the trees happen to grow in some well-behaved manner. Second, traditional methods are structurally rigid. System models, objective functions and improvement algorithms are usually fixed in form; even those techniques

which adapt to environmental information tend to limit that adaptation to a few, select parameters.

The methods we suggest for this study answer these two objections directly. Genetic algorithms are a global search technique, a result of their stochastic origins. They are not, however, a simple random search; genetic algorithms efficiently exploit past information to explore new regions of the decision space with a high probability of finding improved performance.

Learning classifier systems overcome structural rigidity by requiring the formation and testing of general rules for system decision making. In a sense, a learning classifier system learns by reprogramming itself with better and better rules.

By overcoming locality and rigidity, the development of these tools pushes us closer to computer systems capable of autonomous pipeline control.

1.3 Benefits of this Research

The development of practical techniques for learning control of pipelines has a variety of benefits. Most obviously, this research leads to the installation of systems which learn to operate pipelines. These systems will combine the effortless, robust behavior of the human operator with the vigilance of an on-line computer system.

Yet, while we seek systems capable of autonomous control, we are in no way motivated by a desire to replace existing human dispatchers with silicon surrogates; the

savings in salaries is dubious justification for such an undertaking, and automating for its own sake has never been a persuasive argument with justifiably, hard-nosed utility executives. Instead, our motivations are much closer to those of utility management: we seek safety and efficiency gains by using a learning system as a decision aid and as a permanent storehouse of pipeline operations knowledge.

As a decision aid, the experienced learning system acts as an expert consultant to help shape the human dispatcher's operating sequence. Under the stress of abnormal or emergency operations, the learning system acts as an extra pair of eyes and ears, monitoring conditions and suggesting alternatives to alleviate the problem.

As a storehouse of system operating knowledge, the learning system is the permanent repository of all operating experience in a form which cannot quit, retire, or take a job elsewhere. Thus, loss of key people becomes less of a problem, because their experience does not leave with them. Training of new people is simplified, because they may run through a sequence of case studies with the learning system using its response as a guide. Viewed in this way, the learning system is not a competitor to be feared and avoided; rather, it becomes an invaluable member of the operations team, advising, training, and assisting team members as they strive to improve pipeline safety and efficiency.

This research generalizes to many other industries.

Power plants, chemical plants and factories may, one day, employ these techniques. Energy exploration and production operations may be improved using this technology. Any field where operator intuition is an important ingredient is a candidate for the application of these methods. Similar techniques may be applied in design; however, the step from the operation of a relatively fixed plant to the design of some complex system should not be taken too lightly.

A less tangible, but nonetheless, important benefit of this research is the study of engineering intuition itself. In developing robust methods of decision making and learning, we are drawn toward techniques which have some of the richness of human behavior. The study and further development of these methods can help our understanding of the art of engineering which has for so long been shrouded in almost mystical terms.

1.4 Summary

In this chapter, we have outlined the goals, approach, and benefits of this research. Simply stated, we seek robust techniques for the automatic operation of gas pipelines. We start by applying a genetic algorithm (GA) to two problems in pipeline optimization . This demonstrates the utility of the genetic algorithm as a search technique in practical engineering problems. At the same time, it connects our work to existing literature in optimal control of pipelines. Following the optimization work, we develop and apply a learning classifier system (LCS) to control a

gas pipeline. The system must learn to operate the pipeline under normal and leak conditions.

This research has many benefits; however, replacement of dispatch personnel is not our aim. Instead, we seek improved safety and efficiency by using an experienced learning system as a decision aid and knowledge repository. In addition to improving safety and efficiency, these applications should help eliminate personnel turnover problems and training difficulties.

In the remainder, we start by examining the gas dispatching environment with particular emphasis on the use of computers. The genetic algorithm and learning classifier system are successively developed and applied in pipeline environments.

CHAPTER 2

PIPELINE OPERATIONS AND COMPUTERS

To build decision aids for pipelines, one must have a solid understanding of current pipeline operations methods. In this chapter, we review gas pipeline dispatching practice so the responsibilities, available decision-making options, and system information are clearly defined. We also discuss the computer technology of pipelining to understand its effect on the dispatching mission.

2.1 What is Gas Dispatching?

Many have asked this question. Lafferty [2] answers it well in his time-honored article by stating that the primary function of the gas dispatching department is,
". . . determining the market demand for gas and adjusting the operations of a pipeline system to meet those demands." Although this sounds like a fairly straightforward process, it can be a challenging task because of the many conflicting constraints of the pipeliner's environment.

A schematic of this environment is shown in Figure 2-1. The dispatcher must juggle the needs of gas users with the availability of supply, using equipment with highly dynamic response characteristics. This juggling act is performed

against a backdrop of vagaries in weather, corporate policy, and economic conditions. Lafferty discusses many of these issues and gives examples of typical dispatcher solutions.

Dispatching Goals

What is a dispatcher trying to accomplish when he operates a pipeline? The specific answer to this question lies in the complex interaction between the dispatcher, his immediate supervisor, and perceived corporate policy. A more general answer is available when the basic dispatching task is considered. Fundamentally, a dispatcher must deliver sufficient quantities of gas to market safely and efficiently. These three goals, sufficiency, safety, and efficiency must be balanced to achieve a good operating strategy. In many companies, there is an understandable tendency to weight sufficient and safe delivery more heavily than efficiency. This is a source of difficulty for traditional artificial decision makers which tend to weight efficiency most heavily because of their structure.

Dispatch Information and Controls

In a study of decision making--artificial or human--one of the most important ingredients is the information flow from the different components of the environment. The following is a list of data available to a dispatcher:

1. Pipeline - pressures, flows, temperatures.
2. Supply - available quantity and pressure.
3. Demand - current and predicted demand.

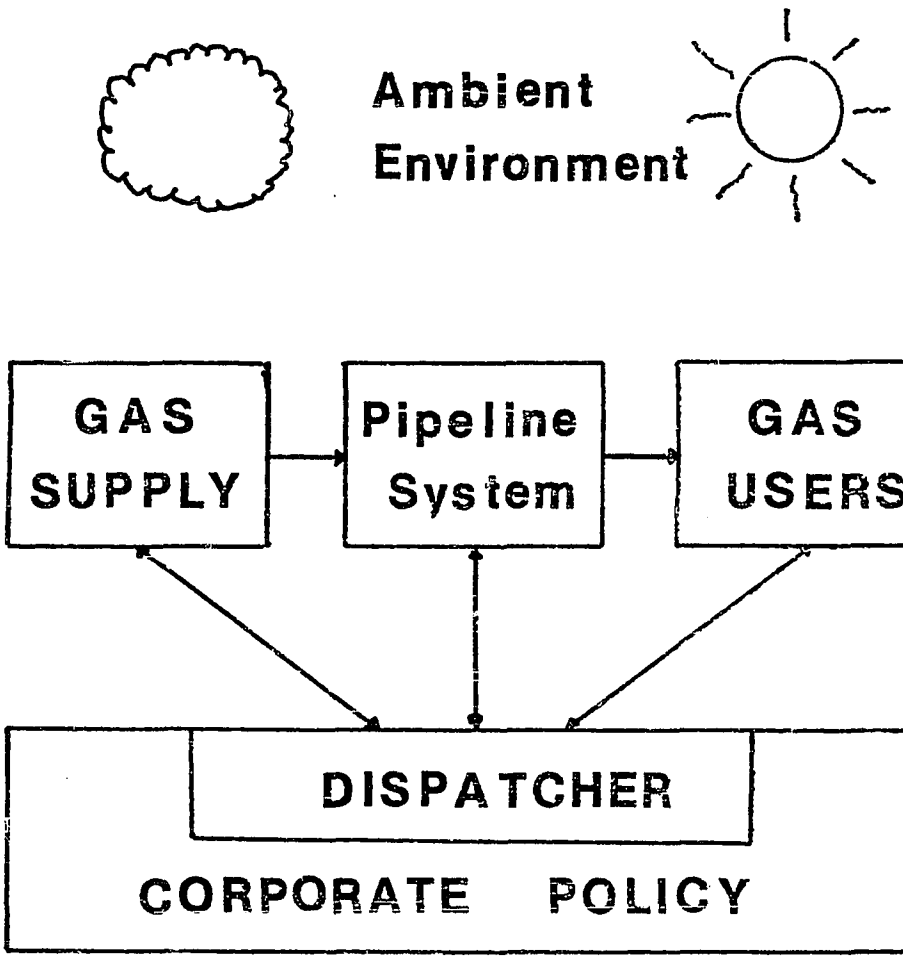


Fig. 2-1. Gas Dispatching Environment

4. Weather - temperature, wind speed and direction, humidity, etc.

5. Corporate - contractual obligations, standard operating procedures, management praise and punishment.

The first four categories are easily quantified, while the last presents some difficulty. The importance of this category and difficulty in its representation are major shortcomings in existing artificial decision procedures.

While a dispatcher receives a large information flow from his environment, he has relatively few actions he can take to maintain acceptable conditions on his system. These actions are related to the components of his environment over which he possesses some control:

Supply - Distribution of supply (pipeline, storage, or production)

Pipeline - Compressor status and level, controller set points.

Demand - Curtailment of interruptible demand.

Additionally, the dispatcher may have to take steps to alert others to dangerous conditions such as line breaks and explosions.

Dispatcher Learning and Decision Making

A thorough investigation of human dispatcher learning and decision making would take us beyond the reasonable boundaries of this dissertation onto the turf of human psychology and cognition; however, to get an intuitive feel for these processes, two gas control centers at two different gas companies were visited. Informal discussions

were held with operations managers and dispatchers; dispatchers were also observed over the course of several days while they operated their pipeline.

During these observation periods, the dispatchers frequently made operating decisions. During these decision episodes, the dispatcher typically spent most of his time communicating with supply, delivery, and internal operating personnel. After getting a clear picture of the situation, he made a decision and passed it on to all involved parties. The decision rarely required any analysis or calculations; the dispatcher simply knew what to do.

As we might expect, attempts to get dispatchers to explain specific decisions were largely unsuccessful; people running on intuition are not the best at explaining the logic of their actions. Nonetheless, one dispatcher volunteered several of his rules of thumb, culled from his pipeline experience:

> If you are losing 10-15 psi/hour then you must take corrective action.
>
> If in a 6 hour period you lose 70 psi of linepack then replenish before moderating operation.
>
> Try to maintain 700 psi at W____ (a location) during the winter.

At this juncture, we are little concerned with the content of the rules (though we might marvel how three little rules could say so much.); rather, we observe that a dispatcher voluntarily chose to describe his knowledge in rule form. This notion of intuitive, rule-based thought will become more important when we decide upon an underlying

computational structure for our learning system.

How does a dispatcher learn to make good decisions? To observe this in the field requires more than a few casual sessions; however, the management-dispatcher discussions did shed light on this process. The primary mode of dispatcher learning is on-the-job training. A new dispatcher (in both companies visited) is an apprentice until he gains a wide spectrum of operating experience. During this apprenticeship, the novice dispatcher observes the line and operates alongside a more experienced person. Some formal training or coursework may also be required, but this is not considered as important as the on-the-job training by the dispatchers themselves. As the new dispatcher gains confidence, he is allowed to take over more and more operating duties, until he is deemed competent to control the system autonomously. Arriving at this point of autonomy seems to take at least a year, and it can take two or three.

The learning processes that take place during this apprenticeship and throughout a dispatcher's career are manifold and complex; yet, one thing is clear: since a dispatcher is never in the exact same situation twice, he must, both, generalize his experience when it is acquired and apply this generalized knowledge to specific situations as they arise. This must bias our search for learning algorithms towards those that can do likewise.

2.2 Computers in Gas Pipelining

The spread of the commercial digital computer in the

fifties encouraged increasing applications in gas pipelining [3-8]. Primarily, computers have been used for communicating and processing distributed pipeline information for centrally located, human pipeline operators. Computer pipeline simulation techniques have been devised for both real-time and look-forward modeling. Optimization and control techniques have been suggested for closing the operations loop; however, to date, no major pipeline is run by an autonomous computer program.

In this section, we examine the application of computers to gas pipelining. Specifically, we survey the use of computers in data acquisition and remote control, simulation, optimization, and other control applications. We survey the field with a general eye, though our main focus remains with automated decision making and learning.

System Control and Data Acquisition (SCADA)

The computer's greatest single impact on the gas dispatch function has come from the widespread installation of centralized data acquisition and control systems. These systems provide centralized data monitoring and logging functions. Additionally, they often permit control over remote compressors, valves, and controller set points. For examples of modern day installations, see representative articles by Yonker [9] and Kloer [10]. Turner [11] presents a good discussion of general system specifications and requirements. Early efforts in this field are discussed in papers by Wilson (1953) [12], Orlofsky (1958) [13], and

Armstrong (1964) [14]. These articles demonstrate the progression from analog methods, to relay computers, to the modern digital computer. The availability of SCADA computers has improved the dispatcher's ability to monitor and respond to the pipeline environment. Previously, remote information was manually logged over teletype or phone, and it was only updated after long time intervals (hours). With SCADA, this information is available upon demand, permitting the dispatcher to form an accurate picture of pipeline status.

Pipeline Modeling

With computers in widespread dispatching use, suggestions for more involved information processing have occurred quite naturally. There has been a growing discussion of the merits of concurrent and look-forward modeling for dispatcher assistance and training. See for example, Pai and Mugele [15], Heath and Blunt [16], Rachford [17], Covington [18], and Wylie and Streeter [19].

The idea is threefold. First, a real-time (concurrent) model may be used to identify anomalies between predicted and actual response, thereby pointing to leaks or other undesirable events. Second, a look-forward model permits an operator to try alternative strategies in a safe quasi-environment, hopefully improving dispatcher decision making. Last, use of a model as a training simulator reduces the need for costly on-the-job training.

Pipeline modeling on computers had its origins in the

fifties when Hardy Cross techniques were placed on the old punched card computers [20]. More realistic models for transmission lines which include dynamic phenomena may be seen in papers by Nelson and Powers (1958) [21], and Taylor, Wood and Powers (1962) [22]. Since that time, many have contributed to the literature of pipeline modeling [23-28].

Modeling has developed where, today, software is available commercially for real-time and look-forward applications. Several look-forward and training simulator models are being installed, although results from these efforts have not been widely reported in the literature.

Continued use of real-time and predictive modeling promises better information digestion by human dispatchers. Whether dispatchers develop hearty appetites for these tools remains to be seen. The existence of modeling tools has also led to the development of various optimization procedures.

Optimization of Pipeline Operations

The notion of optimization is an old and recurring theme in the engineering literature. Basically, one seeks a design or operating sequence that is best in some well-defined sense. In pipeline operations, this may imply finding the operating sequence that minimizes total cost of transportation subject to various delivery and safety requirements. Many techniques have been applied to the pipeline operations problem.

Dynamic programming has been used for steady state and

transient optimization of simple pipeline operations by Wong and Larson [29] and Larson, Humphrey and Wong [30]. Ade [31] has applied Pontryagrin's principle to the problem of optimizing more complex configurations. This work also offers a solid discussion of the many factors that contribute to formulating a realistic objective function. Sood, et al. [32] have considered the network problem using a gradient search technique. Wienecke [33] has described the use of a method based upon linear programming. This work suggests possible real-time usage of the method by operations personnel. Many others have considered the problem of optimization in pipeline design [34-37].

Although there has been much interest in the development of these optimization procedures, their field implementation and use by working dispatchers has not been widely reported. There are numerous reasons for this. Cost of implementation is a factor; software and hardware costs for these methods can be substantial; however, cost is not the whole story. Dispatch supervisors and dispatchers consulted during the informal visits, felt that dispatchers already did a good job, and the optimization procedures would not bring substantial savings. Furthermore, they argued that a computer procedure could not handle all the exceptions and extraordinary situations that arise.

While some of this may be chalked up to job protectionism and homo sapiens chauvinism, there is some sense in this stance. Good dispatchers (human) do a good

job. They respond to a broad spectrum of events with intuitive flair. By contrast, optimization techniques respond to situations with a pre-cast methodology. Changes in the environment must be handled explicitly or they are not noticed at all. Changes in operating philosophy (objectives) are difficult to model.

Because of this, it is understandable why optimization methods have not caught on in the pipeline dispatch environment. We must have robust methods that handle the breadth of a real pipeline system.

Other Computer Methods

Other methods have been suggested for gas pipeline control. Two papers have discussed the development of heuristic pipeline control algorithms. The early SRI report [30] on dynamic programming suggested a heuristic control scheme based upon some simple operations rules. These rules were generalized by the authors from their dynamic programming model. A later paper by Larson and Wismer [38] outlines a scheme for hierarchical control for more general networks. It is interesting that both papers seek to generalize low computation, heuristic rules from complex optimization procedures. The approach to be studied in Chapter 5 seeks heuristic rules in a more direct manner.

2.3 Summary

In this chapter, we have examined the gas dispatching function and the role of computers in that function.

The dispatcher faces a demanding environment, fraught with uncertainty, complex hardware, and changing objectives. Nonetheless, he meets his environment head on, tackling the task with a flexible, intuitive approach. Observations of dispatchers at two gas companies confirm this; the dispatcher learns and decides easily. During decision episodes, he gets a clear picture of the situation, decides what to do, and does it; there is little analysis or agonizing.

Computers have helped the dispatcher in his role as a communicator. Centralized systems provide reliable information and equipment control at his fingertips; however, the computer has had little to do with aiding the dispatcher's decisions.

Why haven't computers been more helpful in this important role? The primary reason is the robustness gap between man and machine. Good (human) dispatchers do a good job; they approach a tough job with intuitive flair and a flexible attack. By contrast, artificial decision techniques approach the task with a rigid, pre-cast methodology that is guaranteed to fail when it encounters the unanticipated or when its pre-set models don't (model).

Thus, it is little wonder that dispatchers and their managers have been less than enthusiastic in their embrace of these techniques. If the computer is ever to effectively aid in the dispatcher's decision process, our mission is clear: we must try to close the robustness gap and bring

machine performance closer to the intuitively flexible approach of the human dispatcher.

Over the next four chapters we take a few small steps in this direction. First, we investigate a genetic algorithm, an improvement search technique with some of the boldness and innovation of human search, in pipeline optimization. After these optimization studies, we integrate the genetic algorithm into a more complete rule-learning system. This system learns rules of thumb for high performance interaction with its pipeline environment. By doing this, we hope to come closer to the intuitive approach used by the working dispatcher.

CHAPTER 3

OPTIMIZATION VIA GENETIC ALGORITHM

Optimization is a natural first step in a quest for artificial decision-making procedures similar to human processes. The idea is simple: human decision makers seek the best decision in some well-defined sense. While common-sensical, this notion begs to be questioned. Human decision makers perform in situations where both the environment of decision and the concept of best are, at best, ill-defined. Because of this, optimization with its well-defined constraints, objective functions, and system models, is no more than a rigid approximation to natural decision-making processes.

Nonetheless, we study optimization for two good reasons. First, because optimization is well defined, it provides a pure proving ground for search procedures. A robust decision maker must use algorithms enabling improvement with experience. With optimization, we can test, explore, and compare different search procedures and still maintain strict control over the search environment. Second, optimization has proved to be a useful tool in the hands of a skilled engineer or technologist. The user can

carefully tune model and objective parameters to obtain desirable system performance; it is, however, this human interaction, the art of optimization, which keeps such systems from autonomy.

In this chapter, we study one search procedure, a genetic algorithm, with promise in both areas: as a search algorithm in a larger decision-making learning machine and as a practical, engineering optimization tool. We focus on the latter for now, but keep in mind the former. Genetic algorithms have seen growing application in the past decade in computer scientific domains. Here, we explore their use as an engineering optimization tool. This lays the foundation for actual application to two problems in pipeline control in the next chapter.

To get a handle on genetic algorithms, we look at what they are and where they come from. In so doing, we question the motivation for looking at still other techniques of optimization (aren't there enough already?). The mechanics of the algorithm are then presented; we attempt to gain some intuition of why they work. We finish with a more rigorous explanation of the underlying search processes.

3.1 What are Genetic Algorithms?

Genetic algorithms are a class of stochastic improvmal algorithm; they seek improved performance by sampling areas of a parameter space with high probability of success. The algorithms are genetic because the string manipulations employed resemble the mechanics of natural genetics. In a

sense, genetic algorithms enforce a Darwinian survival of the fittest among a population of artificial creatures (strings). Every generation, a new set of creatures is created using bits and pieces of the fittest of the old generation; an occasional new part is tried for good measure.

Yet, one should not assume that genetic algorithms are a simple random walk through some parameter space; these methods are not coin flipping by a fancy name. Genetic algorithms efficiently exploit old information to seek trial points with above average performance.

Genetic algorithms have been developed by John Holland and his students in the Computer and Communications Sciences Department at the University of Michigan. The main goals of their research have been twofold: 1) abstract and understand, mathematically, the adaptive processes of natural systems, 2) design artificial systems software that retain the important mechanisms of natural systems. This approach has led to important discoveries in both natural and artificial systems science.

The central issue of this philosophy is <u>robustness</u>--the balance between efficiency and efficacy necessary for survival in different environments. The implications of robustness for artificial systems are manifold. If artificial systems can be made more robust, costly redesigns can be reduced or eliminated. If higher levels of adaptation can be achieved, existing systems can perform

their function longer and better. Designers of artificial systems--both software and hardware, whether mechanical, chemical, civil, or electrical--can only marvel at the robustness, the flexibility of biological systems. Features for self-repair, self-guidance, and reproduction are the rule in biological systems, whereas they barely exist in the most sophisticated artificial systems.

Thus, we are drawn to an interesting conclusion: where robust performance is desired--and where is it not?--nature does it better; the secrets of adaptation and survival are best learned from the careful study of biological example.

Yet, we do not accept the genetic algorithm method by appeal to this beauty-of-nature argument alone. Genetic algorithms are theoretically and empirically proven to provide robust search in complex spaces. The primary monograph on the topic is Holland's, Adaptation in Natural and Artificial Systems [1] (hereafter ANAS). Many papers [39-47] and dissertations [48-54] establish the validity of the technique in function optimization and control applications. Particularly applicable to the present work are Ph.D. theses by Hollstein [51], De Jong [53], and Bethke [54].

While established as a valid approach to problems requiring efficient and effective search, genetic algorithms have not been widely applied in engineering circles. There is no good reason for this oversight. These algorithms are computationally simple, yet powerful in their search for

above average behavior. Furthermore, they are not fundamentally limited by restrictive assumptions about the search space (continuity, existence of derivatives, etc.). We will investigate the reasons behind these attractive qualities; but before this, we need to explore the robustness of more widely accepted search procedures.

3.2 Robustness of Conventional Search Methods

This is not a comparative study of optimal search techniques. Nonetheless, it is important to question whether conventional search methods meet our robustness requirements. The current literature identifies three main types of search methods: calculus-based, enumerative, and random. Let us examine each type to see what conclusions may be drawn without formal testing.

Calculus-based methods have been heavily studied. These subdivide into two main classes: indirect and direct methods. Indirect methods seek local extrema by solving the usually, nonlinear set of equations resulting from setting the gradient of the objective function equal to zero. This is the generalization of the elementary calculus notion of extremal points. On the other hand, direct (search) methods seek local optima by hopping on the objective function and moving in a direction related to the local function gradient. This is simply the notion of hill-climbing. To find the local best, one climbs in the steepest permissible direction. While these methods have been improved, extended, hashed, and rehashed, some simple reasoning shows

their lack of robustness.

First, both methods are local in scope; the optima they seek are only the best in a neighborhood of the current point. Further improvement must be sought through random restart or other trickery. Second, they depend upon the existence of derivatives. Even if we allow numerical evaluation of derivatives, this is a severe shortcoming. Many practical parameter spaces have little respect for the notion of a derivative and the smoothness this implies. The engineering community has been too willing to accept the tradition of the 18th century classicists who painted a clean world of quadratic objective functions, ideal constraints, and ever present derivatives. The real world of search is fraught with discontinuities and vast multi-modal, noisy search spaces. It comes as no surprise that methods depending upon the restrictive requirements of continuity and derivative existence are unsuitable for all but a very limited problem domain. For this reason and their inherent local scope of search, we must reject calculus-based methods; they are insufficiently robust in unintended domains.

Enumerative, search-for-the-best schemes have taken a variety of forms; but, their consideration in the robustness race must be limited for a simple reason: lack of efficiency. Most parameter spaces are simply too large to search one at a time and still have a chance of using the information to some practical end. Even the highly touted

scheme, dynamic programming, is little more than logical enumeration with a model. It too, breaks down on problems of moderate complexity, suffering from a malady melodramatically labeled the "curse of dimensionality" by its creator [55]. We must conclude that less clever enumerative schemes are similarly--and more abundantly--cursed for real problems.

Random algorithms have achieved increasing popularity as researchers recognize the shortcomings of calculus-based and enumerative schemes [56]. Yet, strictly random search must also be discounted because of the efficiency requirement. Random searches, in the long run, can be expected to do no better than enumerative schemes. We must be careful to separate random search methods from randomized techniques. The genetic algorithm we investigate is an example of a search procedure which uses random choice as a tool to guide a highly exploitative search through the parameter space.

While not an exhaustive examination, we are left with a somewhat unsettling conclusion: conventional search methods are not robust. This does not imply they are not useful. The schemes mentioned and countless hybrid permutations have been used successfully in many applications; however, as more complex problems are attacked, other methods will be necessary. We shall soon see how genetic algorithms help fill this robustness gap in practical engineering applications.

3.3 Goals of Optimization

We are seeking methods which optimize efficiently and effectively over a broad environmental spectrum. We must now be clearer about our meaning when we say optimize. What are we trying to accomplish in an optimization process? The conventional view is presented well by Beightler, Phillips and Wilde [57]:

> Man's longing for perfection finds expression in the theory of optimization. It studies how to describe and attain what is Best, once one knows how to measure and alter what is Good or Bad . . . Optimization theory encompasses the quantitative study of optima and methods for finding them.

From this we see that optimization seeks to improve performance toward some optimal point or points. Note that this definition has two separable parts: 1) we seek improvement to approach some 2) optimal point. There is a clear distinction between the process of improvement and the destination or optimum itself. Yet, in judging optimization procedures we commonly focus solely upon convergence, whether the method reaches the optimum, and forget entirely about interim performance. This emphasis stems from the origins of optimization in the calculus. It is not, however, a natural emphasis.

Consider a human decision maker, for example, a businessman. How are his decisions judged? What criteria are used to decide whether he does a good or bad job? Usually he is judged by whether he makes good selections within the time and resources allotted. Goodness is judged relative to his competition. Does he make a better widget?

Does he get it to market more efficiently? With better promotion? Our businessman is never judged by attainment-of-the-best criteria; perfection is all too stern a taskmaster. Convergence to the best is not an issue in business or in most walks of life; we are only concerned with doing better relative to others. Thus, if we want more human-like optimization tools, we are led to a reordering of priorities. The most important goal of optimization is improvement. Can we get to some good, sufficing level of performance quickly. Fine tuning can be performed in our spare time. Attainment of some optimum is much less important for complex systems. It would be nice to be perfect: in the meanwhile, we can only strive to improve.

In the next chapter, we watch the genetic algorithm for these human-like qualities. In the meantime, we define a simple genetic algorithm to see how and why it works.

3.4 A Simple Genetic Algorithm

In this section, we investigate a simple genetic algorithm, both its mechanics and why it works. The mechanics of the process are surprisingly simple. We do nothing more complex than string copying and partial string swapping. The explanation of why it works is much more subtle and powerful. This simplicity of operation and power of effect are one of the main attractions of the genetic algorithm approach.

We separate this discussion into two parts. First, we address the structures being processed. Next, we outline

the rules and operators used to modify the structures. Our main efforts, at first, are directed toward understanding the mechanics of the process and gaining an intuitive feel for their robustness. Later, we return for a rigorous examination of genetic algorithm performance.

Strings and Chromosomes

The basic structure processed by genetic algorithms is the string. The strings we consider are a sequence of characters of finite length l composed over some alphabet V. In this study, we limit ourselves to the binary strings over the alphabet V = {0,1} without loss of generality.

Roughly speaking, strings in artificial systems are analogous to chromosomes in biological systems. In natural systems, the chromosome (or set of chromosomes) is a prescription for the operation of some animal or plant. In artificial systems optimization, the string is a description of a parameter set for operating the underlying system. The system designer has a variety of alternatives in coding numeric and non-numeric parameters. We will confront this when we discuss applications in the next chapter. Right now, we aim to see how genetic algorithms can effect improvement regardless of the coding scheme used.

Because the genetic algorithm is rooted in natural genetics and computer science, the terminology used is an unholy mixture of the natural and artificial. We review the terminology to connect with existing literature and also permit the occasional slip of a natural utterance or two.

In the geneticists parlance, a chromosome is composed of genes which may take on a number of values called alleles. A gene is also identified by its position on the chromosome called its locus. The correspondence between natural and artificial vernacular is shown in Table 3-1.

Table 3-1

Natural and Artificial Vernacular

natural	artificial
chromosome	string
gene	character or bit
allele	bit value
locus	position

In this study, we do not distinguish between a gene (character) and its locus (position); the position of a gene determines its meaning uniformly throughout a population and throughout time. More complex chromosome (string) models may be introduced; but, these have not proved necessary in applications studies to date.

As a notational convenience, we refer to strings by capital letters and individual characters by lower case letters subscripted by their position. For example, the string A may be represented as follows:

$$A = a_1 a_2 a_3 \cdots a_l$$

Here the a_i represent the alleles at the ith gene. A particular string is represented in its binary form.

Populations of Strings

We have defined the individual structures as finite length strings. It would be possible to operate on strings one at a time, thereby generating a sequence of individual strings with time. Many optimization algorithms operate this way, moving gingerly from one parameter set to another: gradient methods, as an example, use this individual sequential approach. Yet, nature does not work this way, and nor shall we. In natural systems, a population of individuals exists at any one time. As time progresses, new generations are born and older generations die away, creating constantly changing populations. Similarly, genetic algorithms generate a sequence of string populations. This factor alone gives genetic algorithms much of their differential advantage over conventional search methods.

By working from a population, genetic algorithms maintain a rich database of well-adapted diversity from which new members may be created. By maintaining this diversity, these algorithms can search different regions of a parameter space in parallel. This results in a search with a much broader, more global flavor than any method that searches from a single point.

As a matter of notation, we consider a sequence of populations $\underline{A}(t)$ where the underscore indicates a vector of strings and the index t refers to the time step. For simplicity, we consider non-overlapping populations $\underline{A}(t)$

where all the members of the population undergo mating and genetic action to create the members of generation t+1. We also limit ourselves to populations of constant size. Having adopted these guidelines, we examine how genetic algorithms act to create a sequence of improving populations.

Reproductive Plans and Genetic Operators

At some point in time, we imagine a population of strings A(t). The job of a genetic algorithm is to perform a series of simple operations on the current population to generate a new population in the next time step. This is done with a number of transition rules.

Genetic algorithms are composed of two types of transition rules:

1. Reproductive plans
2. Genetic operators

Reproductive plans determine the number of copies (offspring) of an existing string to make during a reproductive cycle (iteration). Genetic operators determine the modification and combination of these strings which will form the strings of the next generation.

One simple genetic algorithm, which gives good practical results, is composed of three rules, one reproductive plan and two genetic operators:

1. Fitness proportionate reproduction
2. Simple crossover
3. Simple mutation

Fitness proportionate reproduction is a simple rule whereby the probability of reproduction during a given cycle is proportional to the fitness of the individual string. Fitness is defined as some non-negative measure of merit (an objective function to be maximized). The effect of this rule is clear. High fitness individuals have a higher expected number of offspring than low fitness individuals. Reproduction is, thus, the survival-of-the-fittest or emphasis step of the simple genetic algorithm.

One common implementation of this rule evaluates a reproduction count for every member of the old population. This count is simply the individual's fitness u_i divided by the average fitness of the population $\bar{u}$. This normalization assures a population of constant size N. Note that the reproduction count is usually some non-integral value. To round off to integral values, two things are often done. First, the population count may be scaled so the best individual gets at least two copies. This insures some pressure toward the best strings while still maintaining constant population size. Second, the non-integral reproduction counts may be rounded probabilistically to the next higher or lower integer using the fractional part to bias a simulated coin toss.

After the reproduction phase, simple crossover may proceed in two steps. First, members of the newly reproduced generation are mated at random. Then, each pair of strings undergoes crossover as follows: an integer

position k along the string is selected uniformly at random on the interval $1 \leq k \leq l-1$. Two new strings are created by exchanging all characters between positions 1 and k, inclusively.

For example, consider two strings A and B of length 7 mated at random:

$$A = a_1a_2a_3a_4a_5a_6a_7$$

$$B = b_1b_2b_3b_4b_5b_6b_7$$

Suppose the roll of a die turns up a four. The resulting crossover yields two new strings:

$$A' = b_1b_2b_3b_4a_5a_6a_7$$

$$B' = a_1a_2a_3a_4b_5b_6b_7$$

What is the effect of crossover on the search process? Clearly, crossover is some sort of randomized, yet, structured information exchange. Together with reproduction, this simple operator gives genetic algorithms much of their surprising power. A rigorous explanation of this may be given in terms of schemata. A more intuitive feel can be obtained by considering strings as <u>ideas</u> and substrings as containing <u>notions</u>.

A string is a complete <u>idea</u> or prescription of how to do a particular task (in our case, a description of how to operate a pipeline). Substrings contain various notions of what's important or relevant to the task. Viewed in this manner, the population is a body of knowledge containing a multitude of notions and rankings of those notions for task performance. The act of crossover with previous

reproduction, combines various notions of high performance strings to form new ideas. Intuitively, exchanging notions to form new ideas is appealing if one thinks in terms of the process of innovation. What is an innovative idea? Most often, it is a combination of things that have worked well in the past. In much the same way, reproduction and crossover combine to search potentially pregnant, new ideas.

It is as if various widget experts from around the world gathered at a trade show to discuss the latest in widget technology. After the paper sessions, they all pair off around the bar to exchange widget stories. Well-known widget experts, of course, are in greater demand, and exchange more ideas, thoughts and notions with their lesser known widget colleagues. The show ends and the widget people return to their widget laboratories to try out a surfeit of widget innovations. The process of reproduction and crossover is precisely this kind of exchange among experts. High performance notions are repeatedly tested and exchanged, seeking better and better performance.

While reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and destroy some potentially useful genetic material. The mutation operator protects against such an unrecoverable loss. Mutation is the occasional random alteration of a string position. In a binary code, this simply means changing a 1 to a 0 and vice versa. By itself, mutation is a random walk through the search space. When

used sparingly with reproduction and crossover it is an insurance policy against premature loss of important notions.

Other genetic operators and reproductive plans have been abstracted from the study of biological example; however, the three examined in this section, proportionate reproduction, simple crossover, and simple mutation have proven to be both computationally simple and effective in solving optimization problems. In the next section, we perform a hand simulation of our simple genetic algorithm to demonstrate both the mechanics and effect of the method.

3.5 Genetic Algorithm at Work - A Simulation by Hand

Consider the problem of finding high performance strings (l=5) where the objective functions is $f(x) = x^2$ and strings are interpreted as binary integers on the interval $[0,2^l-1]$. In this section, we perform one generation of the simple genetic algorithm to drive home the mechanics and concept of the method in a simple problem domain.

To start, a small initial population of four strings (N=4) is selected at random. Bit positions have been chosen by flipping an honest penny. The decimal values x are shown with their respective fitness values in Table 3-2. Reproduction counts are set by taking the integer part of the normalized fitness and adding a count with probability equal to the remaining fractional part. In the particular simulation, 3 coins have been flipped to approximate this process to the nearest eighth.

Table 3-2

Hand Simulation of Genetic Algorithm

Initial Population	X Value	f(x)	$\frac{f(x)}{\bar{f}(x)}$	Count	String Copies	Mated With	Cross-over Site	New Population	X Value	f(x)
01101	13	169	0.58	1	01101	2	4	01100	12	144
11000	24	576	1.97	2	11000	1	4	11001	25	625
01000	8	64	0.22	0	11000	4	2	11011	27	729
10011	19	361	1.23	1	10011	3	2	10000	16	256
Average		293								439

NOTES: 1) Reproduction count rounding, crossover and mating performed at random using one or more coin tosses.

2) Mutation probability p_m assumed small enough to be negligible over a single generation.

3) X interpreted as binary integer [0,31].

Reproduction proceeds by copying the number of strings specified in the reproduction counts. Random mating of the strings follows, using coin tosses to pair off the happy couples. After mating, crossover is applied to each pair by randomly selecting a crossover site. The mutation probability has been assumed to be small. Therefore, following reproduction and crossover, the new population is ready to be tested.

The results of a single generation of the simulation are shown in the table. While concrete conclusions from a single trial of a stochastic process are, at best, a risky business, we start to see how genetic algorithms combine high performance notions to achieve better performance. Note how both the maximal and average performance have improved in the new population. Although random processes help cause this happy circumstance, we can see how this improvement is no fluke. The best string of the first generation (11000) receives 2 copies because of its high, above average performance. When this combines at random with the next highest string (10011) and is crossed at location 2 (again at random), one of the resulting strings (11011) proves to be a very good choice indeed.

This event is an excellent illustration of the ideas and notions analogy we developed in the previous section. In this case, the resulting good idea is the combination of two above average notions, namely the substrings 11--- and ---11. While still somewhat heuristic, we start to see how

genetic algorithms effect a robust search. In the next section, we tighten down these concepts by analyzing genetic algorithms in terms of schemata.

3.6 A Rigorous Reappraisal

The intuitive viewpoint developed thus far has much appeal. It places the genetic algorithm in similitude with certain human search processes commonly called innovative or creative; however, as engineers and technologists, we need to have a better handle on genetic algorithm performance.

To get this, we examine the raw data available for any search procedure and discover that we can increase the information available by comparing strings and exploiting important similarities in a population. We develop the framework of similarity templates or schemata to rigorously show how genetic algorithms work. We show how this leads us to consider a keystone of the genetic algorithm process, the building block hypothesis.

Grist for the Search Mill - Important Similarities

For much too long we have ignored a fundamental question. In a search process where we only have payoff data (fitness), what information is contained in a set of structures (strings) to help guide a directed search for improvement? To make this clearer, consider the strings and fitness values originally displayed in Table 3-2 from the simulation of a previous section and gathered below for convenience:

String	Fitness
01101	169
11000	576
01000	64
10011	361

What information is contained in this population? On the face of it, there is not much: four independent samples of different strings with their fitness values. As we stare at the page, however, quite naturally we start scanning up and down the string column. We notice certain similarities among the strings. Furthermore, we note certain similarities that seem highly associated with good performance. The temptation is great to experiment with these high fitness associations. It seems reasonable to play with those particular substrings that are highly correlated with past success. For example, in the sample population, the strings starting with a 1 seem to be among the best. Might this be an important ingredient in optimizing this function? Certainly with this function ($f(x)= x^2$) and coding (binary integer) we know it is. But, what are we doing here? Really, two separate things. First, we are seeking similarities among strings in the population. Second, we are looking for causal relationships between these similarities and high fitness. By doing this, we admit a wealth of new information to help guide a search. To see how much and precisely what information is being considered, we introduce the important concept of a schema

or similarity template.

Schemata - Similarity Templates

We are no longer interested in strings as strings alone. Since important similarities can help guide a search, we need to define how a string is similar to other strings. In what ways is a string a representative of other string classes with similarities over certain string positions? The framework of schemata provides the tool to analyze these questions.

A schema is a similarity template describing a subset of strings with similarities over certain string positions. The template may be motivated by appending the symbol * or don't care to the normal alphabet V. With this extended alphabet, the characters of V retain their normal significance. A don't care in a string position means that any character of V at that position will satisfy the template. Thus, positions that are important are specified by elements of V; positions where the similarity is unimportant are occupied by a don't care. We point out that the * is a meta-symbol; it is not actually processed by any algorithm. It is simply a notational device which allows us to describe all possible similarities.

As an example, consider the strings of length 5. The schema *0000 describes a subset of strings, namely {10000, 00000}. The schema *111* describes a subset with 4 members {01110, 01111, 11110, 11111}. In this way, schemata provide a straightforward means of describing all the similarity

subsets possible among the strings of given length.

Counting the number of schemata is an enlightening exercise. In the previous example, with l=5, we note 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. At first blush, it appears that schemata are making our search lives more difficult. For an alphabet of cardinality k there are only (only?) k^l different strings. Why consider schemata and enlarge the space of concern? The answer lies in the number of strings and schemata contained in each sampled population. How many strings are there in a population of size N? Obviously, there are N strings in a population of size N. How many schemata are contained in a population of size N? To see this, consider a single string of length 5: 11111, for example. This string contains 2^5 schemata because each position may take on its actual value or a don't care symbol. In general, a particular string contains 2^l schemata. As a result, a population of size N contains somewhere between 2^l and $N*2^l$ schemata depending upon the population diversity. This fact verifies our earlier intuition. The original motivation for considering important similarities was to get more information out of a string population. The counting argument shows that there is indeed a wealth of information about important similarities contained in even moderately sized populations. We will examine how genetic algorithms effectively exploit

this information. At this juncture, we suspect the need for some parallel processing if we are to make use of all this information in a timely fashion.

Schema Properties

All schemata are not created equally. Some are more specific than others. Some have defining positions that span a greater or lesser proportion of the string. To identify these differences we identify two properties of a schema following Bethke [54]: order and defining length.

The order of a schema h, denoted by o(h), is simply the number of ones or zeroes present in the template. It is a measure of schema specificity; the higher the order the more specific the template.

Defining length of a schema h, denoted by $\delta(h)$, is the distance between the first and last specific string position (one or zero). Defining length is a measure of schema span.

To illustrate these properties consider a schema h_a over the strings of length 7:

$$h_a = *10***1$$

What are the order, $o(h_a)$ and defining length $\delta(h_a)$ of this particular schema? The order is straightforward. There are two ones and a single zero. Hence, the order of this schema is 3. To calculate the defining length, we note 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. While these two properties are easily calculated for any schema, they play an important role in

identifying genetic algorithm performance.

Processing of Schemata by Genetic Algorithm

Schemata are an interesting notational device for discussing similarities in strings. More than this, they provide the basic means for analyzing the performance of genetic algorithms. We have already considered how genetic algorithms process string populations. We now consider how they process schemata, the underlying similarities in the string population. We examine reproduction, crossover, and mutation to identify the expected effect of each transition rule.

The effect of reproduction on the number of schemata in a population unveils quite directly. Recall that a string receives $u / \bar{u}$ copies during fitness proportionate reproduction. For any schema h, we assume there are m(h,t) such schemata in a population at time step t. After reproduction, we expect $m(h,t)*u(h)/\bar{u}$ schemata h, where u(h) is simply the average fitness over all the strings containing schema h. Considering reproduction alone, the number of schemata will grow or decline depending upon the ratio $u(h)/\bar{u}$--whether a schema is above or below the current sampling average.

This rate of schemata growth has been connected to the multi-armed bandit problem by both Holland [1] and De Jong [53]. Simply stated, this problem seeks the optimal allocation of trials among a number of alternatives with unknown payoff (a bank of slot machines). The objective of

the problem is to minimize the expected losses from the allocation. The form of the solution derived by Holland suggests that an exponentially increasing number of trials should be allocated to the observed best alternative. In fact, the optimal strategy is not realizable because it requires knowledge of outcomes before their occurrence. Nonetheless, the plan forms an important bound on performance which a time sequential procedure should attempt to emulate. The simple reproductive plan does precisely this. Fitness proportionate reproduction allocates exponentially increasing numbers of trials to the observed best schemata. As the number of trials increases, the allocation by this procedure approaches the optimal allocation. As such, it is an important, yet easily realized, near-optimal allocation procedure.

Reproduction by itself is not too interesting in complex systems as no new points are explored. Crossover provides the structured, though randomized, information exchange between strings to effect a search of new points. How does crossover affect schemata growth? Clearly, many schemata remain unscathed because the crossover site does not fall between schemata defining positions. To see this, examine the following schema:

10*1***

Recall that simple crossover proceeds with the random selection of a crossover site and the exchange of material through that site inclusively with the chosen mate. We note

that the first defining position of the schema is at location 3 and last defining position is at location 6. Therefore, this schema survives the crossover operation a majority of the possible cases. The schema survives with crossover at locations 1 or 2 or locations 6 through 10. Conversely, the schema is broken with a crossover site at locations 3, 4, or 5. Since the site is selected uniformly at random, the probability of schema survival is easily calculated. For the particular example, there are 10-3 = 7 survival sites out of 10 possible sites, probability=0.7. More generally, this crossover survival probability may be related to the defining length $\delta(h)$. The survival probability is equal to $1 - \delta(h)/(l-1)$ for a particular schema h because the schema survives if the crossover site does not fall within the defining length. Actually, this estimate is conservative because it does not include the probability of swapping identical defining positions between two strings.

The probability of mutation leaving a schema unscathed is also easily calculated. The number of defining positions in a schema has been called its order o(h). If the probability of mutation is constant and specified as p_m, the probability of a schema surviving intact is simply $(1-p_m)^{o(h)}$. For small values of p_m, this is well approximated by $(1-o(h)\cdot p_m)$.

The combined effect of all three transition rules, reproduction, crossover, and mutation, is easily derived.

The expected number of schemata h to survive into the next generation is simply the product of the expected number from reproduction alone and the survival probabilities of crossover and mutation:

$$m(h,t+1) = m(h,t)\cdot u(h,t)/\bar{u}(t) \cdot(1-\delta(h)/(l-1))\cdot(1-o(h)\cdot p_m)$$

Identical results are presented in ANAS. The factor multiplying the m(h,t) may be thought of as a growth factor. If it is larger than one, the expected number of schemata h, will continue to grow; otherwise, it can do no more than remain constant in number. We note that this relationship holds for all schemata contained in the population. In other words, the simple genetic algorithm processes all schemata in this manner. We observe that highly fit schemata tend to survive because of the factor $u/\bar{u}$. Short definition length schemata are also preferred because the crossover survival probability is closer to one. As a practical matter, mutation usually plays little role as mutation probabilities are often quite small (<0.001); this has little effect except on schemata of very high order.

Standing back from the computation, we observe several things: short schemata, $\delta(h) << 1$, are sampled at near-optimal rates. Longer schemata are sampled correspondingly less frequently than this. In all, Holland [58] 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. While this is considerably less than the $2^l-N\cdot 2^l$ schemata which exist in a

population, it still represents a large amount of data processing for populations of even moderate size (50-100). This process proceeds in parallel with the action of simple operators applied to strings only; no explicit computation is ever made to correlate or keep track of schemata. They simply behave as we have shown--the best getting more and more trials. This property of genetic algorithms has been called implicit parallelism by Holland because large number of schemata are handled simultaneously without explicit manipulation or recognition.

Building Blocks

The picture of genetic algorithms is much clearer with the perspective afforded by schemata. High performance schemata of relatively low defining length are sampled, recombined and resampled to form strings of potentially higher performance. In a sense, the problem has been reduced in complexity; instead of seeking the highest performing string by trying every conceivable combination, we try to construct better and better strings from partial solutions of past samplings.

Because high performance, low defining length schemata play such an important role in the action of genetic algorithms, Holland gives them a special name: building blocks. Just as a child creates magnificent fortresses from simple blocks of wood, a genetic algorithm seeks optimal or near-optimal performance through the juxtaposition of short, high performance schemata or building blocks.

There is, however, one catch in our discussion to this point. Repeatedly we have claimed that notions combine to form better notions. Just now, we have claimed that building blocks combine to form better building blocks. While these ideas seem perfectly reasonable, how do we know whether they hold true or not?

Certainly there is a growing body of empirical evidence to support these claims. Starting in 1967 with Bagley's pioneering thesis [48] through De Jong's careful study [53] of genetic algorithms on a broad problem spectrum, this building block hypothesis has held up in many different problems. Smooth uni-modal problems to noisy multi-modal problems have been successfully attacked using virtually the same reproduction, crossover, and mutation model. While limited empirical evidence does no theory prove, it does suggest that genetic algorithms are appropriate for many of the types of problems we normally encounter.

More recently, Bethke [54] has shed a great deal of light on this topic. Using discrete Walsh transforms and clever transformations of schemata, he has obtained a method to identify the actual schemata average fitnesses. In this way, he is able to identify whether for particular functions and codings, the building blocks combine to form the optimum. Attempts to generalize these results to arbitrary codings prove difficult, although he does derive sufficient conditions on the derivatives of a function of a single variable which has been encoded by a normal fixed point

coding. Despite the current limitations, Bethke has provided an important tool for the analysis of genetic algorithm performance.

As part of this research, Bethke also has generated a number of test cases which are genetic algorithm hard (GA-hard): they are not easily solved by the reproduction, crossover, and mutation procedure. While the results are inconclusive, they tend to suggest that when functions are genetic algorithm hard, they tend to contain isolated optima; the best points tend to be surrounded by the worst. Practically, many of the functions we encounter do not have this needle-in-the-hay-stack quality. There is usually some regularity in the function and coding that may be exploited by the building blocks. Nevertheless, it is important to keep in mind that, fundamentally, the simple genetic algorithm depends upon the combination of high performance building blocks to seek the best points. If the building blocks are misleading due to the coding used or the function itself, the problem may not be solvable by the simple algorithm.

3.7 Summary

In this chapter, we have laid a foundation for understanding genetic algorithms. We are lead to these methods by our search for robustness; natural systems are robust--efficient and efficacious--as they adapt to wide ranging environments. By abstracting the adaptation mechanism of natural systems in algorithm form we hope to

achieve similar breadth of capability. In fact, genetic algorithms have proven their breadth in analytical and empirical studies.

The detailed mechanics of a simple genetic algorithm have been presented. Genetic algorithms operate upon populations of strings. The strings are coded to represent the underlying parameter set. Reproduction, crossover, and mutation are applied to successive string populations to create new string populations. The operations performed are simple string copies and partial string swaps, yet the effect is extremely powerful. Genetic algorithms realize an innovative notion exchange among strings. The best strings provide the largest number of notions contributing to continued improvement. A hand simulation of a simple genetic algorithm has been helpful in illustrating both the detail and power of this method.

In this examination, we notice there are three notions underlying the genetic algorithm concept which separate it from more familiar search techniques:

1. Direct manipulation of strings (structure)
2. Search from a population, not a single point.
3. Search rules as stochastic operators

Genetic algorithms manipulate control variable representations at the string level to exploit similarities in above average population members. Other methods deal with functions at the variable level; the underlying coding is only a necessary evil to obtain a computer solution.

Because genetic algorithms operate at the coding level, they are difficult to fool even when the function is difficult.

Genetic algorithms work from a population; many other methods work from a single point. Genetic algorithms heed the old adage "security in numbers." By retaining a population of sample points, the probability of reaching a false (local) peak is reduced.

The transition rules of genetic algorithms are stochastic; many methods have deterministic transition rules. We are careful, however, to distinguish between these randomized operators and random search. Genetic algorithms use random choice to guide a highly exploitative search. This may seem unusual, using chance to achieve a particular result (the best points); nature is full of precedent [46].

A more rigorous appraisal of genetic algorithm performance has been undertaken using schemata or similarity templates. A schema is a string over an extended alphabet, V+(*), where V is the normal string alphabet and the asterisk is a don't care symbol. This notational device greatly simplifies the analysis of the genetic algorithm method because it explicitly recognizes all the possible similarities inherent in a population of strings. We have shown how building blocks (short, high performance schemata) are combined to form strings with expected higher performance. This occurs because short building blocks are sampled at near-optimal rates and recombined via crossover.

Mutation has little effect on the building blocks; it does help prevent the unrecoverable loss of potentially important genetic material. Whether building blocks combine to form better strings depends upon the function and the coding used While there are functions which are genetic algorithm hard (GA-Hard), these problems tend to have remote, highly isolated optima and are difficult for other optimizers as well.

Genetic algorithms seem to have much to recommend them. In the next chapter, we apply the method to two problems in pipeline optimization. This application will help identify whether the method is as practical as it is promising.

CHAPTER 4

APPLICATION OF GENETIC OPTIMIZATION IN PIPELINING

In this chapter, we apply the simple genetic algorithm to two problems in pipeline optimization. Our goal is to illustrate genetic programming's effectiveness in practical problem domains. We also try to clarify some of the more persistent implementation details we have encountered along the way.

We first explore four issues which confront the genetic optimization user: discretization and coding, constraint handling, minimization mapping, and genetic algorithm parameter selection. We then solve two problems in pipeline optimization: the serial, steady state problem and the transient, single line problem. The simple genetic algorithm presented in the previous chapter is used with constant, fixed parameters. Problem formulations and results from both problems are presented and analyzed; techniques are suggested for improving the already acceptable performance. We finish the chapter by reviewing the method's salient features.

4.1 Discretization and Coding

Genetic algorithms process populations of strings to

search a particular problem space for improved performance. While problems exist where the natural coding is a string (crossword puzzles, word games), the usual engineering problem is formulated as a parametric or continuous variational problem.

In a parametric problem, a finite number of real parameters may be adjusted for best performance. In the variational problem some real function must be varied in time or space to control the process. In either case, the use of genetic algorithms forces us to transform the underlying formulation to a finite string through some discretization and coding process.

Discretization may be required at a variety of levels. With variational problems, we must first reduce the problem to a finite number of parameters. This is accomplished through the type of discretization associated with interpolation theory, finite elements, and other related areas. Typically, the continuum is subdivided into discrete chunks. Parameters are associated with points in the space, and some functional relationship--step function, piecewise polynomial, smooth spline, etc.--is assumed to describe the function's behavior in between the discrete points.

Upon reduction of the problem to a finite number of parameters (if necessary), another kind of discretization becomes important: the discretization resulting from the choice of coding. In most numerical computer work, we take our parameter codings for granted. Typically, in using an

algorithmic language, FORTRAN as an example, we simply create a set of parameters of type REAL and perform our numerical work upon the parameters as if they were, in fact, real numbers; we tend to forget that we are working with a finite simulation of the real numbers. With genetic algorithms, we must be very explicit about both the coding and the way we combine the codings to form a string.

While we could form strings by simple concatenation of the normal REAL representation, there is strong motivation to avoid this approach. In the previous chapter, we learned how genetic algorithms process building blocks--relatively high performance, short schemata. If we use the normal floating point representation on most computers (30-60 bits), the resulting strings are enormous for problems with even modest numbers of parameters. For these codings, adequate interaction of different parameters or even different parts of the same parameter would require building blocks with long defining lengths. As we know, longer building blocks are destroyed with high probability by the crossover operator. Therefore, it is better to custom tailor shorter parameter codings which span the control space with adequate precision.

A number of alternatives are available to code individual parameters. Shortened fixed point (integer) and floating point (real) codings are possible. De Jong [53] discusses a mapped, fixed point parameter where the j bit substring is interpreted as the usual, unsigned binary

integer on the interval $[0,2^j-1]$. This integer is linearly mapped to some specified interval of the real numbers, $[u_{min},u_{max}]$. The precision π of this type of coding is uniform and depends upon the length of the parameter substrings j:

$$\pi = \frac{|u_{max}-u_{min}|}{2^j-1}$$

This type of parameter coding is adopted in this study.

Floating point codings have been suggested [54] for genetic optimization. In a typical floating point representation, the string contains two parts, a mantissa and and exponent. This type of coding gives greater range at the expense of precision. They are advantageous when parameters are known to vary over many orders of magnitude. In this study, uniform precision is more important than wide range; floating point codings have not been used.

Other coding schemes are possible. For example, Gray codes have been investigated with genetic algorithms [51]. Gray codes map adjacent integers to binary strings which differ by precisely one bit. While interesting, this work proves no consistent differential advantage for these more esoteric codings, and we stick to conventional schemes.

Once individual parameters have been coded, we must decide how to assemble them on a string. Again, a variety of options confronts us. The simplest and most highly investigated scheme simply concatenates the individual parameter codings into a single string. Other methods have

been suggested which intermix the bits of different parameters. The hope here is to create highly relevant, short length building blocks through rearrangement of the string structure. Bethke [54] has suggested a preset interleaving of bits from each parameter. A more general approach to this parameter mapping problem is to use the inversion operator discussed by Holland [1]. Inversion, a genetic operator with precedent in nature, is a structured yet randomized rearrangement of locus (string position). In this way, natural selection not only finds highly fit strings, it finds highly fit arrangements of the bits on the string. Implementation of this operator requires that we tag each gene with its logical position; physical position no longer is sufficient to specify function. While potentially important in complex problems, it has not been considered in this study. Instead, we go part way and permit a specified shuffling of parameter bits with a bit map. Before decoding each string, the bit map rearranges the physical string into the logical string which is viewed as a concatenation of parameter substrings. If no bit map is specified, the physical string and logical string are identical; the string is a simple concatenation of parameter substrings.

4.2 Constraints and Genetic Algorithms

Thus far, we have only discussed genetic algorithms for searching unconstrained fitness (objective) functions. Typical engineering problems often contain one or more

constraints which must also be satisfied. In this section, we see how constraints may be incorporated in a genetic algorithm search.

Constraints are classified whether they are equality or inequality relations. Since equality constraints may be subsumed into the system model, we are only concerned with inequality constraints.

At first, it would appear that inequality constraints pose no particular problem. A genetic algorithm generates a sequence of parameter sets to be tested using the system model, objective function, and constraints. One simply runs the model, evaluates the objective function, and checks to see if any constraints are violated. If not, the parameter set is assigned the fitness corresponding to the objective function evaluation. If constraints are violated, the solution is infeasible and thus, has no fitness. This procedure is fine, except that many problems are highly constrained; finding a feasible point is almost as difficult as finding the best. As a result, we might want to rate infeasible solutions as well, perhaps degrading their fitness ranking in relation to the degree of constraint violation. This is what is done in penalty methods [59].

In a penalty method, a constrained problem in optimization is transformed to an unconstrained problem by associating a cost or penalty with constraint violations. This cost is included in the objective function evaluation. Consider the original constrained problem:

$$\begin{aligned} \text{minimize} \quad & f(\underline{x}) \\ \text{subject to} \quad & g_i(\underline{x}) \geq 0 \qquad i=1,2,\ldots,n \\ \text{where} \quad & \underline{x} \text{ is an m vector} \end{aligned}$$

We transform this to the unconstrained form:

$$\begin{aligned} \text{minimize} \quad & f(\underline{x}) + r * \sum \phi(g_i(\underline{x})) \qquad i=1,2,\ldots,n \\ \text{where} \quad & \phi \text{ - penalty function} \\ & r \text{ - scaling coefficient} \end{aligned}$$

A number of alternatives exist for the form of the penalty function ϕ. In this study, we simply square the violation of the constraint: $\phi(g_i(\underline{x}))=g_i^2(\underline{x})$ for all violated constraints i. Under certain conditions, the unconstrained solution converges to the constrained solution as the scaling coefficient r approaches infinity [59]. As a practical matter, r 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. This sizing procedure is discussed in more detail for each problem.

Suggestions exist to provide an increasing sequence of penalty coefficients during the optimization process [59]. At first, when many of the trials are infeasible, useful payoff information is obtained. As the solution progresses, higher and higher penalties are enforced, driving the solution to the actual optimum. This idea has much merit; it is not adopted here because it introduces more parameters, complicates the process, and shifts our attention from the real concern, the use and workings of

genetic algorithms.

4.3 Fitness Mapping

In many cases, optimization studies are naturally formulated as minimization problems. The genetic algorithm depends upon maximizing a fitness function, the non-negative, increasing figure of merit discussed earlier. As a result, we sometimes must find a way to transform a minimization problem to a non-negative maximization problem.

In normal optimization practice, we can transform minimization to maximization or vice versa by multiplying the objective function by a -1. With genetic optimization we must also satisfy the non-negativity requirement. One way to do this is with the following simple mapping relationship:

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

where C_{max} - nominal maximum cost
$u(x)$ - fitness function
$g(x)$ - cost function

This is the method adopted throughout this study. In the current implementation, C_{max} is specified by the user; it could have been selected automatically as the highest value of g(x) calculated thus far; however, this has not been done to keep the run comparisons meaningful.

There are other alternatives for this mapping function. Consider the following rational function:

$$u(x) = C_1 / (C_2 + g(x))$$

Clearly, as g(x) goes to infinity, u(x) goes to zero as desired. C_1 and C_2 may be selected to scale u(x) appropriately. This function may be particularly useful for functions with a wide range of values.

4.4 Setting Genetic Algorithm Parameters

Genetic algorithms have a number of parameters which must be selected: population size N, crossover probability p_c' and mutation probability p_m. The effect of these parameters upon genetic algorithm performance has been investigated extensively by De Jong [53]. He has performed parametric studies of the basic genetic algorithm over a set of five test functions. These results point in several directions:

- -Crossover probability should be high to obtain maximum search of new samples
- -Mutation probability should be low to prevent destruction of well-adapted schemata.
- -Population size should be moderate to avoid problems of genetic drift while encouraging rapid improvement and avoiding the inertia of large numbers.

In this spirit, the following values have been chosen for these parameters:

'In the previous chapter, we assumed a crossover probability of 1. It is a simple extension to determine whether an individual string is to be mated and crossed via random choice. This process introduces the crossover probability as a parameter.

Population size = 50

Probability of Crossover = 1.0

Probability of Mutation = 0.001

Parameter selection can have an effect upon performance of the genetic algorithm. De Jong's work accounts for the contributing factors, backing up his conjecture with both intuitive and mathematical reasoning. In the present study, we, too, could search for optimal parameter settings, but this would counter our real objectives. We are interested in how well the method works when we haven't twiddled with parameters and fine tuned results. This is a better measure of the practical performance we might expect when applying the method in other problem domains.

4.5 Steady State Serial Line Problem

One important problem of pipeline control is the long term optimization problem. If demand remains steady for a long period of time, how do we supply and compress gas to minimize transportation cost, yet still meet delivery and safety constraints? This is a static problem because the flow is assumed constant for a long period. In actuality, the strict conditions of long term optimization are never met in practice: small fluctuations from steady conditions always exist. Nonetheless, for many pipeline systems, particularly long transmission systems, the approximation is a good and useful one because lines are set up and operated to give relatively, if not perfectly, steady operation.

In this section, we study the long term optimization of

a serial pipe-compressor system using a genetic algorithm. The serial configuration is of practical importance because it is so commonly used in transmission practice. This configuration has also been studied using other optimization methods. As a result, we are able to identify optimal results independently and compare performance. In the remainder, we define the problem, the system model, objective function, and constraints; we also discuss important implementation detail and present results of genetic algorithm trials.

Modeling Equations and Control Parameters

The problem we study here has also been solved by other methods. Wong and Larson [29] originally formulated and solved the problem using dynamic programming. More recently, Edgar, et. al. [37], have solved the same problem using a gradient procedure. As a point of comparison, we adopt the notation and problem formulation of Wong and Larson.

A schematic of the line configuration is presented in Figure 4-1. We envision a serial system with an alternating sequence of compressors and pipelines. A fixed pressure source exists at the inlet; gas is delivered at line pressure to the terminus. Along the way, compressors are used to boost pressure with fuel for the compressors taken from the line. To model this system we consider the system equations for pipe flow and compression.

Steady state pipe flow of natural gas is well studied.

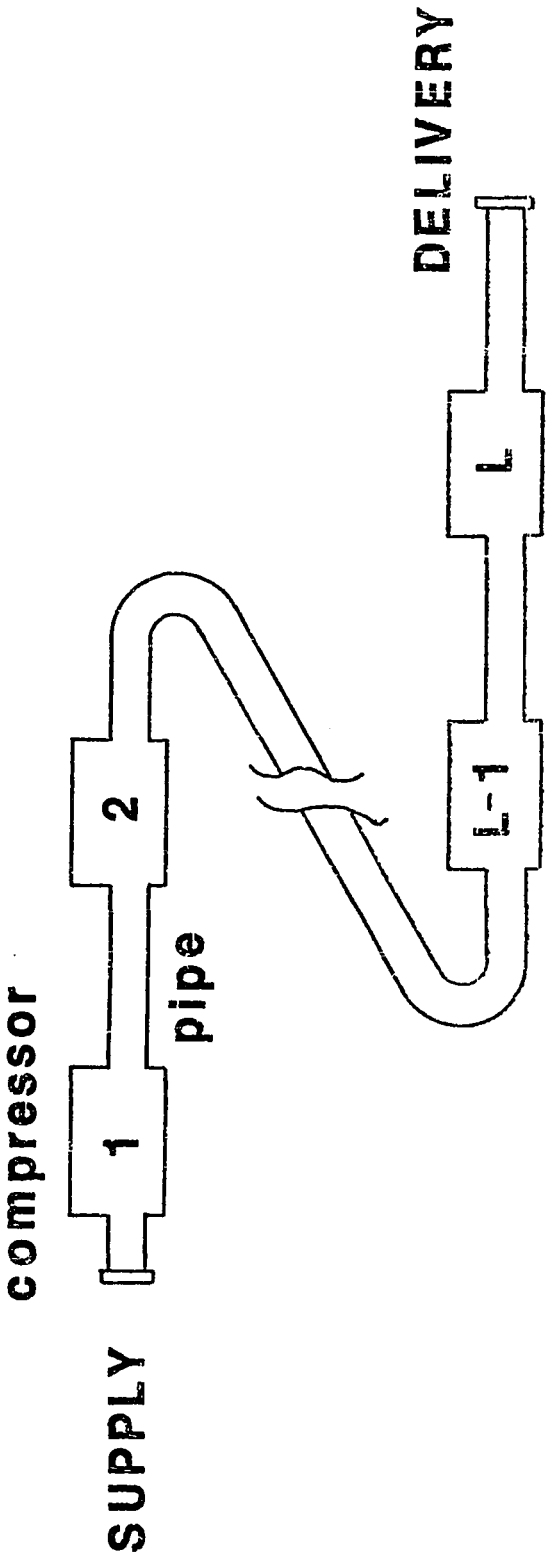


Fig. 4-1. System Schematic - Steady Serial Problem

Assuming isothermal conditions and a level pipeline, the difference of the squares of the absolute pressures (ΔP^2) is proportional to the standard volumetric flow[2] rate squared. The pressure-flow relationship may be stated as follows:

$$PD_i^2 - PS_{i+1}^2 = K_i \cdot Q_i^2 \qquad i=1,2,\ldots,N$$

where PD_i - discharge pressure of the ith compressor (psia)
PS_i - suction pressure of the ith compressor (psia)
Q_i - standard flow of gas ith pipe (MMCFD)

Wong and Larson identify this as the Weymouth equation; actually, it represents any number of equations of the form $\Delta P_i^2 = K_i \cdot Q_i^2$ with the proper choice of the K_i. Generally speaking, the constant of proportionality depends upon friction, temperature, pipe diameter, pipe length, as well as dimensional constants and standard conditions. In this study, we assume these constants are given for each line segment.

The pressure-flow relationship for a compressor may be obtained by considering the adiabatic compression of an ideal gas. The resulting equation is of the following form:

$$HP_i/Q_i = A_i \cdot (PD_i/PS_i)^{R_i} - B_i$$

where HP_i - Power required for ith compressor (horsepower)
Q_i - Flow rate (MMCFD)
A_i, B_i, R_i - Compressor Constants for ith compressor

The coefficients A_i, B_i and R_i may be selected to match

---

[2]In natural gas practice, it is common to measure gas quantities as a standard volume, the volume of gas occupied by a certain mass expressed at some reference pressure and temperature (standard conditions).

either theoretical considerations or the performance of an actual unit; the latter is preferred in practice. The power consumption HP_i is often regarded as the primary control parameter. In Wong and Larson [29], an auxiliary control relationship and variable is introduced; power is calculated as a side computation. For consistency we follow their procedure and introduce the set of control parameters $U_i=PD_i^2-PS_i^2$. This parameter, the difference of the squared discharge and suction pressures, is computationally convenient; it does not correspond to usual gas operations practice. Dispatchers usually watch power, compression ratio or simply suction and discharge pressure to control and monitor a line. In a numerical procedure this choice is somewhat arbitrary, however.

In natural gas systems, the flowing fluid often provides fuel for the compression equipment. To express this fact, a fuel removal factor describes the reduction in flow rate experienced downstream through the following relationship:

$$Q_i = (1-r_i)\cdot Q_{i-1}$$

where r_i - fuel removal factor at the ith station

The fuel used is related to detailed compressor settings. For simplicity the r_i are specified as input constants.

Objective Function and Constraints

The objective for this problem is to minimize the total power consumption. We state this succinctly in the following relation:

minimize $(\sum HP_i)$

Keep in mind that this objective function is only one of many alternatives. A more complete discussion of other factors in gas transmission objectives is found in Ade's work [31].

This minimization proceeds subject to a number of constraints. The pipelines are not permitted to carry gas at high pressures for safety reasons. Furthermore, there is usually a required minimal pressure because of contract requirements. Together, these constraints may be written as follows:

$$P_{i-1_{min}} \leq PS_i \leq P_{i-1_{max}}$$
$$P_{i_{min}} \leq PD_i \leq P_{i_{max}}$$

Additionally, constraints are placed upon the pressure ratio, the ratio of discharge pressure to suction pressure, at each of the compressors. These correspond to physical constraints of power available and prudent operation. We represent them by the following set of relations:

$$1 \leq PD_i/PS_i \leq S_{i_{max}}$$

The lower bound corresponds to no compression, while the upper bound values S_i are the physical constraints for each compressor unit.

Computational Considerations

The steady state system model, objective function, and constraints have been posed. We proceed to solution of the problem by putting forth numerical parameters and clearing some of the final details of discretization and constraint

representation.

The particular problem we solve is the 10 compressor, 10 pipe problem of Wong and Larson [29]. Numerical parameters are presented in Table 4-1. The pipe segments are fairly uniform except for numbers 3 and 9 which are about twice as long as the average pipe. The compressors are roughly equivalent except for units 4, 6, and 7. Unit 4 has a pressure ratio limit of 1.3 due to its more limited capacity, while unit 7 has a better-than-average pressure ratio limit of 1.75 due to its additional capacity. All the units are equally efficient except for number 6 which is two thirds as efficient as the others. We might expect a lower utilization of this unit compared to the others.

To perform the optimization, each string must decode to a set of control parameters. For this problem, the string decodes to a set of 10 U_i values (ΔP^2 across the compressor station). As indicated previously, we use a mapped fixed point coding. Each parameter is treated as a j bit integer which maps linearly to the interval $[U_{min}, U_{max}]$. For this problem, we select a discretization of j=4 bits. Additionally, we must select the bounds for U_{min} and U_{max}. The natural lower bound corresponds to a compressor at rest, $PD_i=PS_i$ or $U_{min}=0$. The upper bound corresponds to the maximum possible difference:

$$U_{max} = \max\{P_{i_{max}}\}^2 - \min\{P_{i_{min}}\}^2$$
$$= 1000^2 - 500^2 = 7.5(10^5)$$

The resulting precision π of this coding is easily

Table 4-1

Numerical Parameters
Steady Serial Problem

Pipe/ Compressor i	Pipeline Coefficients	Compressor Coefficients				Constraints		
	K_i	R_i	A_i	B_i	r_i	P_{imin}	P_{imax}	S_{imax}
1	0.800	0.217	215.8	213.9	0.005	500.0	1000.0	1.6
2	0.922	0.217	215.8	213.9	0.005	500.0	1000.0	1.6
3	1.870	0.217	215.8	213.9	0.005	500.0	1000.0	1.5
4	0.894	0.217	215.8	213.9	0.005	500.0	1000.0	1.3
5	0.917	0.217	215.8	213.9	0.005	500.0	900.0	1.6
6	0.989	0.217	323.7	320.8	0.005	500.0	1000.0	1.6
7	0.964	0.217	215.8	213.9	0.005	500.0	900.0	1.75
8	1.030	0.217	215.8	213.9	0.005	500.0	1000.0	1.5
9	1.950	0.217	215.8	213.9	0.005	500.0	1000.0	1.6
10	1.040	0.217	215.8	213.9	0.005	500.0	1000.0	1.6

calculated:

$$\pi = (U_{max} - U_{min})/(2^4-1) = 5(10^4)$$

This precision in the control variable U_i translates to an average precision in pressure of 34 psi over the range 500-1000 psia. This level of precision proves adequate for engineering purposes; additional precision may be obtained by lengthening the parameter substrings. We must also decide how to form the full string from the parameter substrings. In this study, the full string is obtained with normal bit map; the resultant is a string of length 4•10=40, the concatenation of 10, four bit substrings. With j, U_{min}, U_{max}, and bit map selected, we may create a population of binary strings and interpret the strings as U_i values. For example, with j=4, U_{min}=0, U_{max}=7.5(10^5), and normal bit map the string 1000 0011 0111 0110 1111 0000 1100 1000 1110 0110 (low to high=right to left, spaces added for readability) translates to the set of U_i values (30000, 70000, 40000, 60000, 0, 75000, 30000, 35000, 15000, 40000) (low to high=left to right).

Constraints have been specified on both minimum and maximum pressures as well as compression ratio. As previously indicated, these constraints are adjoined to the problem with a penalty method. A cost is added to the unconstrained objective (total horsepower) proportional to the square of the constraint violations. Table 4-2 displays values of penalty coefficient adopted in this study. These coefficients have been sized to penalize a nominal violation

on each of the compressors by about 10% of a nominal operating cost.

Table 4-2

Penalty Coefficients
Steady Serial Problem

Constraint Type	Penalty Coefficient	Nominal Violation
maximum pressure	10	5 psia
minimum pressure	10	5 psia
pressure ratio	$1(10^6)$	0.02

With objective function, discretization, and constraints in place, we are almost ready to perform the genetic optimization. One final computational detail must be explored. Since we have formulated the problem as a minimization we transform the objective function to a fitness function with the transformation described previously:

$$u(x) = C_{max} - g(x)$$

For this problem, the maximum of g is very large; however, we choose $C_{max}=1(10^6)$. This permits adequate admission of infeasible solutions, while maintaining a reasonably competitive range.

Results of Computation

The model, objective function, constraints, and genetic algorithm have been programmed as described. In this section, we examine results from independent trials and

compare to published results.

To initiate simulation, starting populations (N=50) are chosen at random. For each trial, the genetic algorithm is run to generation 60. This represents a total of 50•61=3050 function evaluations per trial. This may seem like a large number of function evaluations until we consider the size of space being searched. Recall that the binary strings are of length l=40. This represents a total of $2^{40}=1.1(10^{12})$ possible different alternatives in the search space. In this light, 3050 function evaluations is a miniscule fraction, 0.00000028%, of the possible unique alternatives. To put this performance in perspective, if we were to search this efficiently for the best person among the world's 4.5 billion inhabitants, we would only examine 13 people before making our selection.

The results of three independent trials--using different starting points for the pseudo-random number generator--are displayed in Figure 4-2. This figure shows the cost of the best string of each generation as the solution proceeds. At first, performance is poor. By the action of selection, mating, and crossover, better and better strings are formed. In all three cases, near-optimal results are obtained by generation 20 (1050 function evaluations). Careful examination of the best-of-run results is instructive. Table 4-3 shows the cost breakdown for the best of each run. It is comforting to see that three independent simulations consistently give near-optimal

results. In all cases, power cost alone is very near-optimal, with most of the excursion incurred as penalty. This is the result of small constraint violations caused by the fairly crude degree of pressure control (≈34 psi).

Another useful performance measure is displayed in Figure 4-3, the population average performance. At first, most of the population is infeasible; as a result, the average is near the arbitrary specified maximum cost of $1(10^6)$. For a time, improvement comes easily. During early and middle stages, the creative notion exchange brings vast, rapid improvement. After awhile, the population enters a stagnation period. An examination of the individual strings at this point shows substantial convergence at most bit positions. This fact is also reflected in the closeness of the population average and the maximum value by comparing Figures 4-2 and 4-3 during later generations. We may wonder why the solution exhibits a form of convergence which does not contain the global optimum. There are three reasons which have been identified for this behavior: discretization and constraint handling, genetic drift, and genetic algorithm hard problems. We will examine these reasons and their solutions near the end of the chapter.

In Figure 4-4 we compare the optimal pressure profile (solid line) to the best of run SS.1 (triangles). First, we note that the optimal solution is highly constrained. Six pressures rest at a maximum or minimum pressure constraint. Figure 4-5 shows the situation in run SS.1 for the

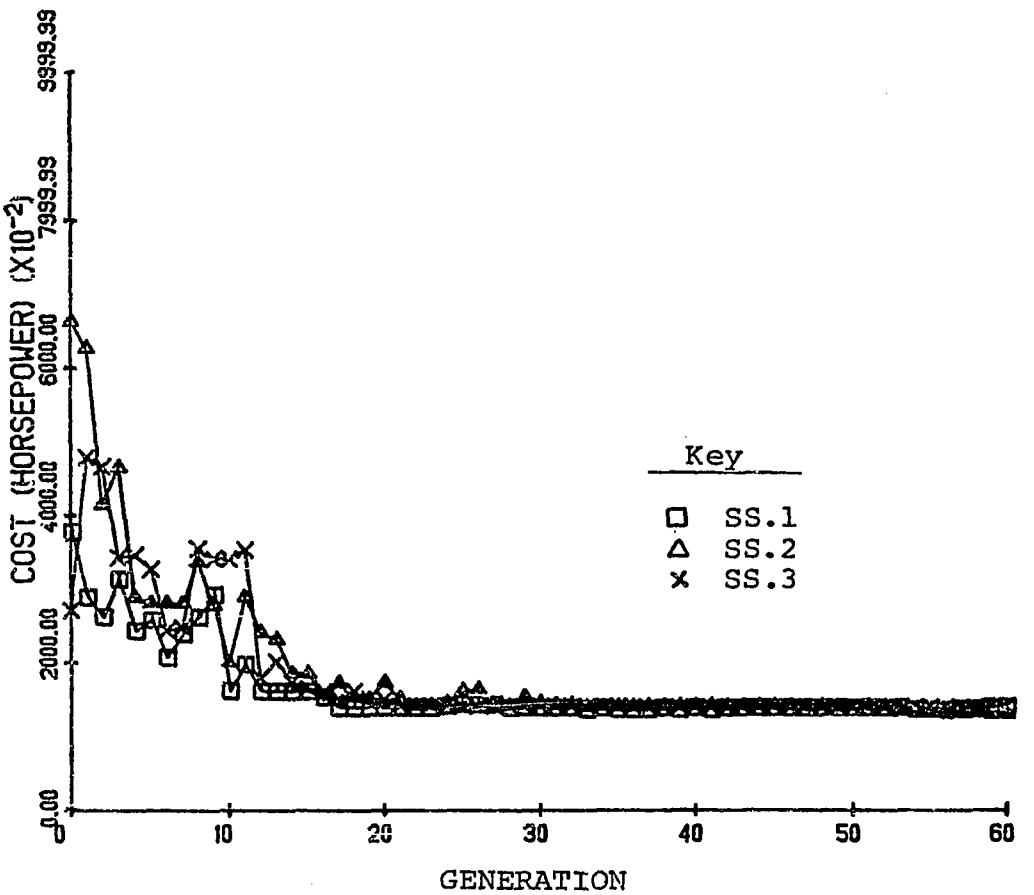


Fig. 4-2. Best-of-Generation Results - Steady Serial Problem

Table 4-3

Best-of-Run Results
Steady Serial Problem

Run	Least Cost of Run	Power Cost	% Difference from Optimal	Penalty Cost	% of Optimal
SS.1	1.380	1.148	+1.1	0.232	20.4
SS.2	1.310	1.187	-4.6	0.123	10.8
SS.3	1.400	1.057	-6.9	0.343	30.2
Mean	1.363	1.131		0.233	20.5
Optimum	1.135	1.135	0	0	0

NOTES: 1) All costs are in units of 10^5 horsepower.

compression ratio. In the optimal results, 4 compressors rest at maximum compression ratio. In all, 10 points are constrained. This results naturally from the characteristics of compressible flow. In a pipe carrying an isothermal compressible fluid, friction gradient decreases with increasing pressure. Furthermore, for a given flow, compression power required is a sub-linear function of compression ratio. As a result, other things being equal, it is desirable to run at the highest pressures and compression ratios while just satisfying delivery pressure requirements. The optimal solution reflects this expected behavior well. The only pressures not dictated by constraint are discharge pressures at station 6 and 10. Station 10 is set to just satisfy the outlet pressure after

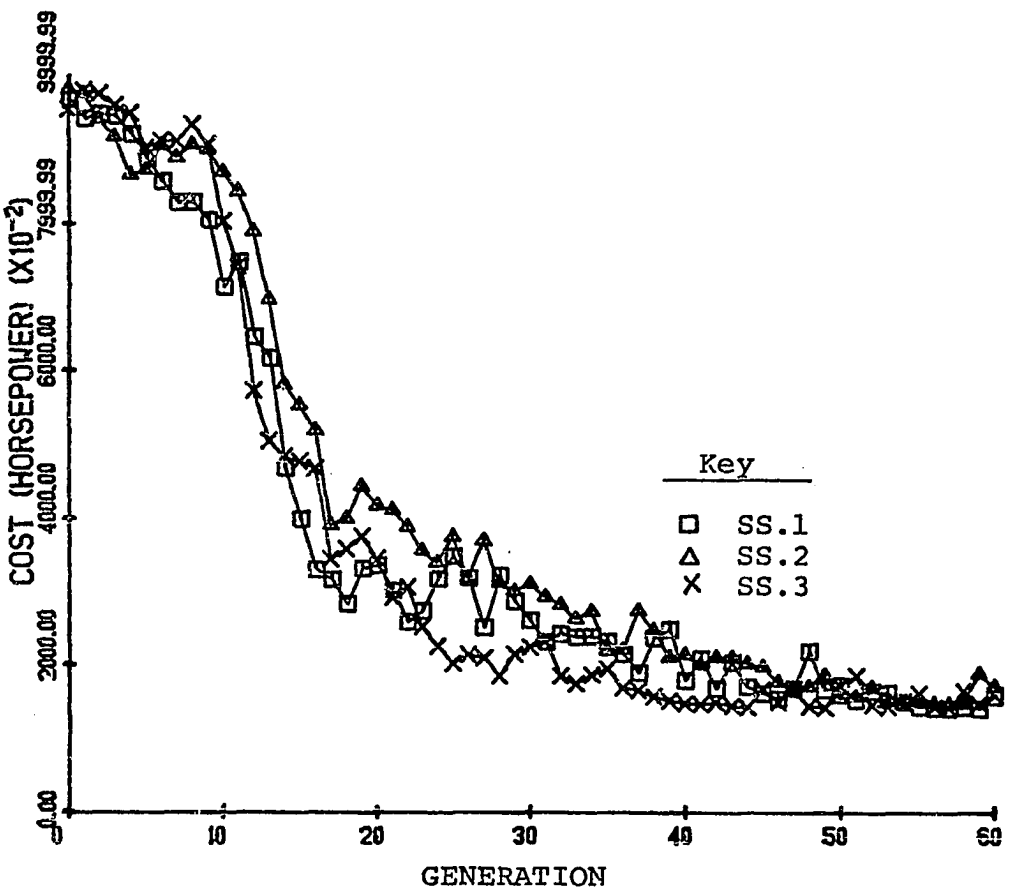


Fig. 4-3. Generation Average Results
Steady Serial Problem

accounting for losses through the last pipe. Compressor 6 is not fully utilized because of its low efficiency. Recall, that compressor 6 is two thirds as efficient as the other units. As a result, it is advantageous to use more economical compression in its place.

The genetic algorithm solution also reflects these basic principles. Run SS.1 agrees well with the optimal solution except at compressors 1, 4, 6 and 8. Over compression occurs at compressors 1 and 4 with attendant under compression at 6 and 8. The under compression at 6 is in line with the low efficiency of that unit. The over compression at units 1 and 4 is consistent with the principle of elevated pressure; however, these excursions are less than optimal because of pressure ratio constraint violation. The under utilization at unit 8 is not explained by any means; somewhere along the way this set up was advantageous in the context of other existing substrings.

4.6 Single Line Transient Problem

Another important area of application for optimization is in time-varying flow problems. On many gas transmission lines the assumptions of steady flow are hardly, if ever, met. In these situations, it is important to model the dynamics of pipeline flow in addition to the friction losses that play the predominant role in steady flow. In this section, a problem in transient flow on a single gas pipeline is studied to further illustrate the use of the genetic algorithm. Results are presented and compared to

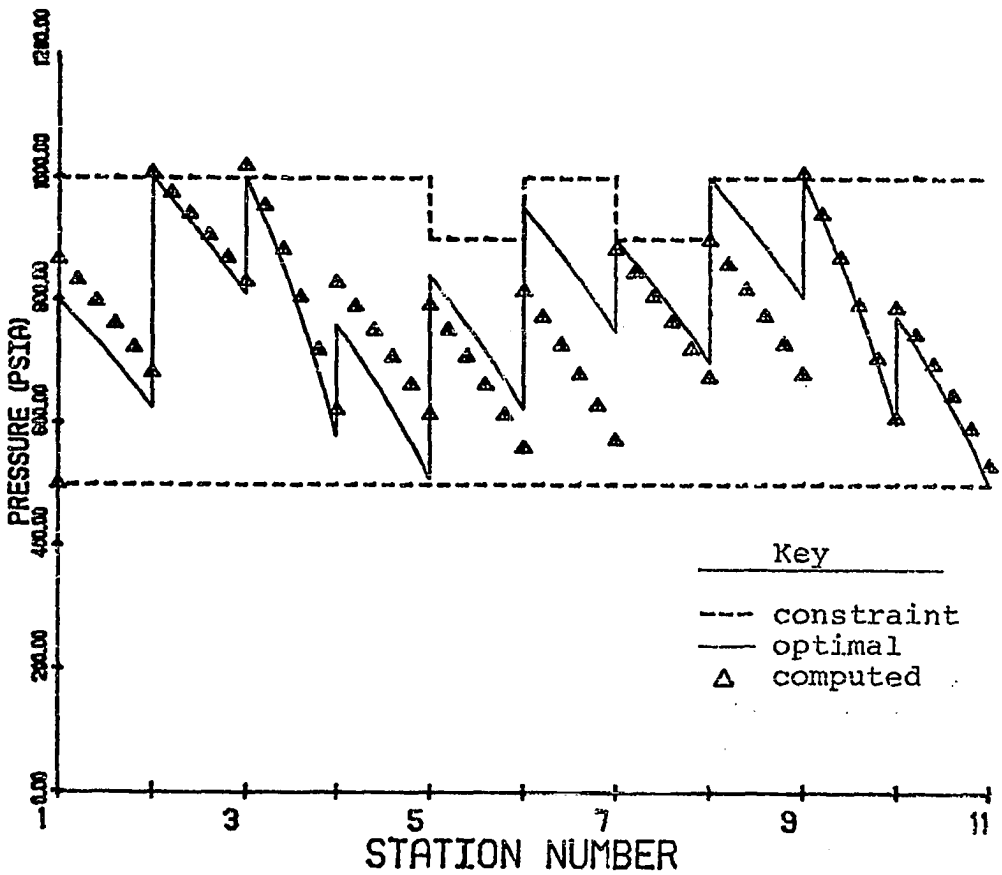


Fig. 4-4. Pressure Profile - Run SS.1
Steady Serial Problem

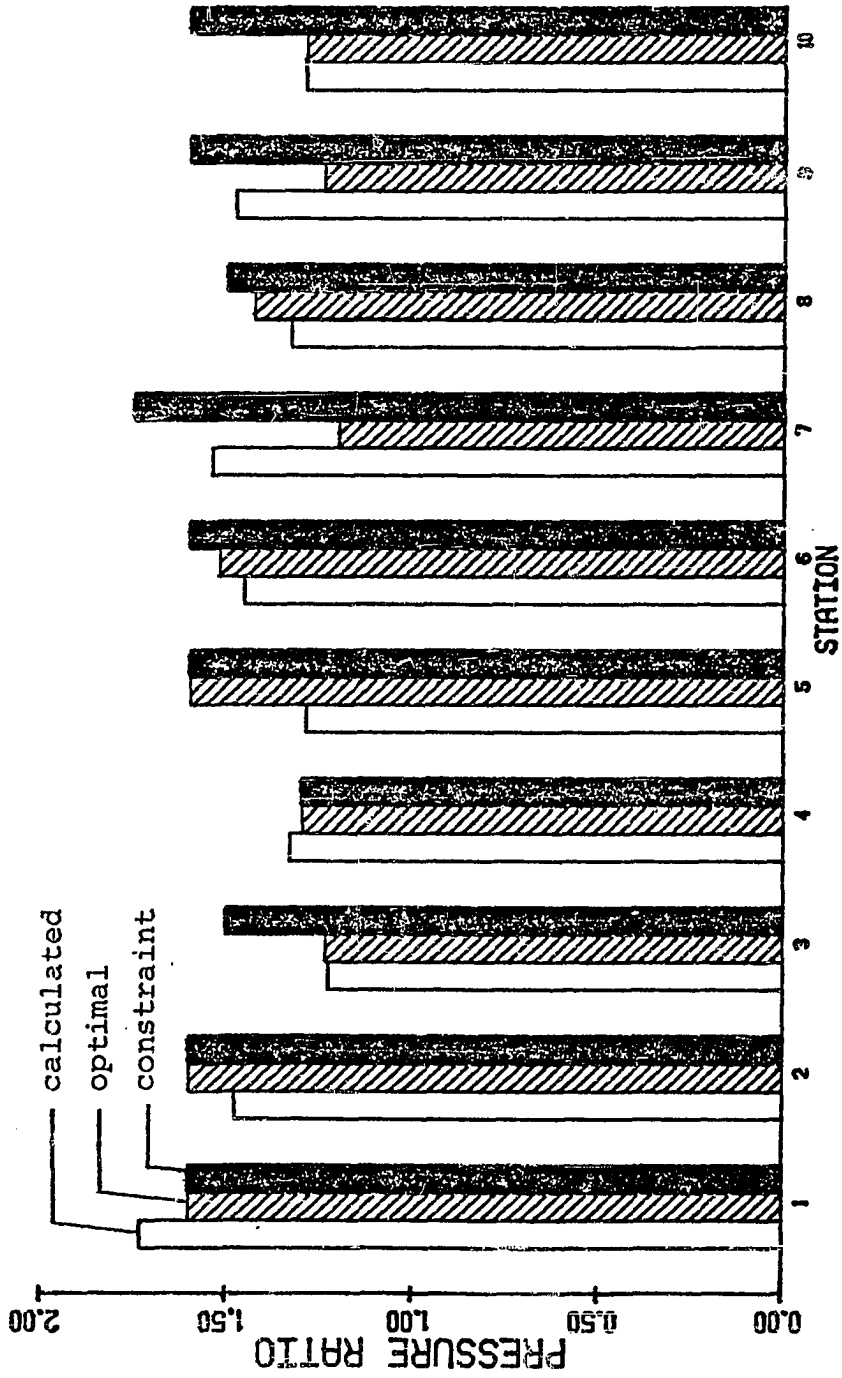


Fig. 4-5. Compression Ratio Results - Run SS.1 - Steady Serial Problem

solutions available in the literature.

The situation we study is illustrated in a schematic diagram, Figure 4-6. A known demand schedule is required at the delivery end of a single gas pipeline of known dimension and characteristics. The supply flow may be adjusted within limits to meet demand. This is done while minimizing the cost of compression subject to minimum and maximum pressure constraints.

This problem has been well studied by other authors using more traditional methods of optimization. These methods were briefly reviewed in Chapter 2. In this section, we again follow the work of Wong and Larson; however, we only adopt their problem specification this time; newer, more appropriate modeling techniques are used in this study.

Modeling Equations and Control Parameters

As with the steady state problem, it is conventional to consider models for each piece of equipment and link them together. In the following, we consider a single pipe with specified flow boundaries and a compressor at the upstream end.

Dynamic pipeline models have been extensively studied and tested. In this study, simplified, one-dimensional partial differential equations of mass and momentum conservation are written and solved using the method of characteristics transformation and a second order finite difference approximation. This model is well established

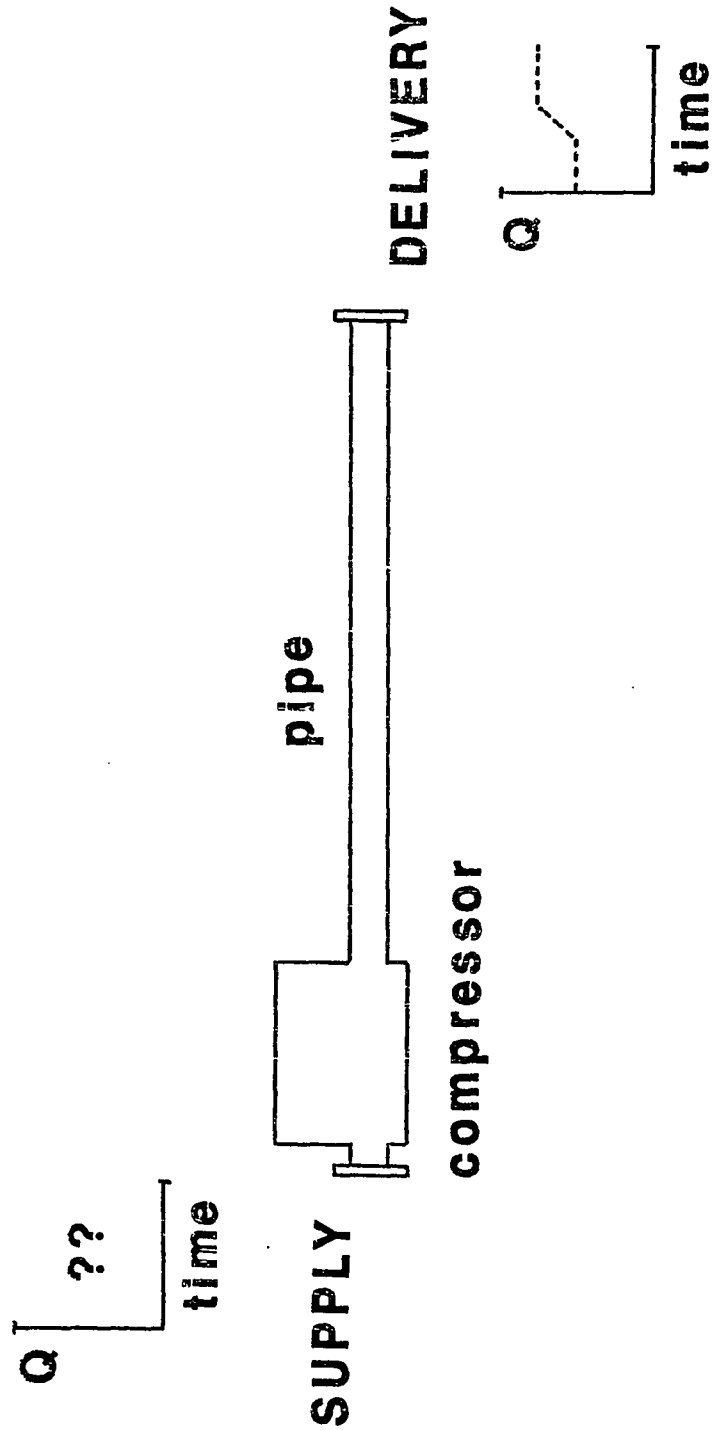


Fig. 4-6. System Schematic - Single Line Transient Problem

and has been used by numerous natural gas companies in operations and design.

As in the steady state case, flow is assumed to be isothermal and all pipes have level profile. These restrictions may easily be lifted; they have been adopted to keep the equation set as simple as possible and still maintain a realistic simulation. The simplified momentum equation may be written as follows:

$$\frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{\partial v}{\partial t} + \frac{f \cdot v|v|}{2 \cdot d} = 0$$

where
- p - pressure
- ρ - mass density
- v - average velocity
- d - pipe diameter
- f - friction factor (dimensionless)
- x - distance along pipe
- t - time

A simplified continuity equation may be written:

$$\frac{\partial m}{\partial x} + \frac{\partial \rho}{\partial t} = 0$$

where
- x,t,ρ - as before
- m - mass flux (mass rate per unit area)

The equation set is completed by recognizing an appropriate equation of state, as well as a relationship between average velocity and mass flux:

$$p = \rho RT$$

where
- ρ,p - as before
- T - temperature (absolute scale)
- R - gas constant

$$m = \rho \cdot v$$

where
- ρ,v,m - as before

Eliminating variables among the four equations we obtain the following two relationships:

$$c^2 \frac{\partial \rho}{\partial x} + \frac{\partial m}{\partial t} + \frac{f \cdot m|m|}{2 \cdot d \cdot \rho} = 0$$

$$\frac{\partial m}{\partial x} + \frac{\partial \rho}{\partial t} = 0$$

where m, ρ, x, t - as before
c - isothermal wave speed ($c^2 = RT$)

Together with appropriate initial and boundary conditions, these equations provide sufficient information to calculate density and mass flux over the entire pipeline for all time. Initial conditions appropriate to this problem may be written as follows:

$$\rho(x,0) = f(x)$$

$$m(x,0) = g(x)$$

where f, g - specified functions of x

It is quite common to assume steady initial conditions; this has been done in this study. To complete the problem specification, we specify mass flux boundary conditions at both pipe ends:

$$m(0,t) = h(t)$$
$$m(L,t) = i(t)$$

where i, h - specified functions of time
L - length of pipe

For this problem, the downstream boundary represents a specified user demand. The upstream boundary is the main control variable. We are attempting to vary flow to obtain optimal (least energy) results.

To solve the equations, we transform the partial differential equations to ordinary differential equations using the method of characteristics. The resulting

equations are solved numerically on a fixed space-time grid. The method of characteristics may be motivated in a number of ways [19]. Consider multiplying our reduced continuity equation by an unknown multiplier λ and adding the product to the reduced momentum equation:

$$\lambda\left[\frac{c^2}{\lambda}\frac{\partial\rho}{\partial x}+\frac{\partial\rho}{\partial t}\right]+\frac{\lambda\partial m}{\partial x}+\frac{\partial m}{\partial t}+\frac{f\cdot m|m|}{2d\rho}=0$$

We recognize from the calculus the form of a total derivative:

$$\frac{d}{dt}\left[m(x,t)\right]=\frac{\partial m}{\partial x}\cdot\frac{dx}{dt}+\frac{\partial m}{\partial t}$$

A similar expression is available for the density. In the combined expression, we recognize the form of the total derivatives and can rewrite it if we are willing to restrict the direction of integration. This process results in the following four differential equations:

$$\pm c\frac{d\rho}{dt}+\frac{dm}{dt}+\frac{f\cdot m|m|}{2\cdot d\cdot\rho}=0$$

$$\frac{dx}{dt}=\pm c$$

These four ordinary differential equations may be conveniently solved using a second order finite difference approximation. The detail of this formulation is presented elsewhere [19] and is not covered here.

The resulting procedure may be thought of as a black box where time-varying upstream and downstream flow are specified and upstream and downstream density (pressure) may be calculated as time goes on.

We also require a relationship for calculating power

consumption in the upstream compressor. The relationship developed in a previous section for steady flow is also used to calculate power use in transient flow. Dynamic effects in compressors are small compared to the inertia and storage capacity of even modest length transmission lines; such effects are usually and presently ignored. The time integral of power consumption (energy) is calculated using a trapezoidal rule approximation, a second order procedure.

Objectives and Constraints

As before we want to deliver the demand flow at minimum total power while satisfying pressure constraints. The objective must now be stated as an integral because the problem is now time dependent:

$$\text{minimize } \int_0^t HP(t) \cdot dt$$

This minimization is subject to minimum and maximum pressure requirements:

$$p_{min} \leq p(x,t) \leq p_{max}$$

With system model, constraints, and objective function specified, we consider the computational details in preparation for solution.

Computational Considerations

The computational detail of our particular solution is now presented. Specifically, we examine numerical parameters, discretization, and constraint handling. The particular problem we solve has been presented by Wong and Larson [29]. Their pipeline parameters were selected from

field studies presented originally by Wilkinson, et al. [23]. The pipeline characteristics and compressor coefficients are presented in Table 4-4.

Table 4-4

Pipeline and Compressor Coefficients
Single Line Transient Problem

Pipeline Parameters		
Length :	15.95	miles
Inside Diameter :	1.63	feet
Gas Molecular Weight :	20.30	(S.G.=0.7)
Friction Factor :	0.01028	
Temperature :	530.7° R	
Compressor Parameters		
A Coefficient :	215.8	horsepower/MMCFD
B Coefficient :	213.9	horsepower/MMCFD
R exponent :	0.217	
Constraints		
Minimum Pressure :	450	psia
Maximum Pressure :	None specified	
Maximum Pressure Ratio :	None specified	

Unlike the steady state problem, we now have a time dependent problem; we must find the input flow function which minimizes accumulated horsepower. To solve by genetic algorithm, we first reduce to a finite number of parameters

by time discretization and then code and discretize each parameter to a suitable precision. Time discretization is accomplished over m equidistant intervals using linear interpolation between the m+1 parameters associated with each of the interval endpoints. For the particular problem, 10 minute intervals are chosen resulting in 15 parameters over a 140 minute simulation. Each parameter is represented by the mapped fixed point coding discussed previously. For this problem, we choose a 3 bit representation over the interval, $[Q_{max},Q_{min}]=[100,170]$ MMCFD[3]. This results in a precision π equal to $|170-100|/(2^3-1)=10$ MMCFD. The string representation of the time series is the concatenation of the fixed point parameters (normal bit map) resulting in a string of length $l=3\cdot15=45$.

Constraints are once again adjoined to the problem via penalty cost. As before, the cost is taken proportional to the square of the violation; however, because the problem is time dependent, we must integrate the violation over the simulation. For the constraints, this is done with a rectangular rule integration, a first order procedure. The penalty coefficient has been sized to permit a nominal 5 psi violation over 50% of the simulation. The penalty coefficient in psi and average horsepower units is 4822. This coefficient yields a nominal average horsepower cost of

---

[3]Although the equations are in mass flux form, we still express the rates as a volumetric mass flux. To convert one to the other, we use the relation, $m=\rho_s \cdot Q_s/A$ where A is the pipe cross-sectional area and the subscripts indicates a quantity expressed at standard conditions.

25.

The minimization is transformed to maximization with the same kind of mapping used in the steady state case. The total cost is subtracted from a nominal maximum cost of 5000. The fitness is the greater of the calculated difference or zero. The cost in these runs is interpreted as an average power consumption. The accumulated power (energy) is time normalized resulting in an average power consumption over the simulation.

Results of Computation

In this section, we examine the results of two independent trials of the genetic algorithm on the transient problem. The model, objectives, and constraints have been programmed and interfaced to the same genetic algorithm used in the steady state problem. Identical parameters have been used for the population size, mutation and crossover probabilities:

population size = 50

$p_{crossover}$ = 1.0

$p_{mutation}$ = 0.001

Once again, the genetic algorithm finds improvement in a workmanlike manner. Figure 4-7 shows the best-of-generation results. Both trials are started from random populations and run to generation 60 resulting in a total of 3050 function evaluations. Improvement is steady, although less dramatic than that obtained in the previous case. This is because the interval of control variable (input flow) has

been more carefully selected. As a result, highly infeasible solutions are difficult to come by. This is in stark contrast to the steady state formulation where much of the parameter space was highly infeasible.

Table 4-5 shows a cost breakdown on the best of the two runs in comparison with the optimal solution. In both cases, results are near-optimal. Unpenalized power is very near-optimal, and the penalties themselves are a small percentage of the optimal average power.

The generation average results are presented in Figure 4-8. Population average lags the best as expected. Near the end of the run, examination of the strings shows convergence at most bit positions. This is also reflected in the closeness of the population average and best results.

Examination of a sample solution is also instructive. Wong and Larson found that the optimal results were obtained by just maintaining the minimum pressure at the downstream point at all times. The genetic algorithm, best-of-run TR.1 results follow this same trend quite closely. Figure 4-9 shows the delivery point pressure with time for run TR.1. The pressure remains near 450 psia except for small excursions in the middle and end of the run. The input flow time history selected by the genetic algorithm results is shown in Figure 4-10. Wong and Larson's results show a similar flow hump mid-run although the results are not directly comparable because the pipeline models are different. The dashed line shows the specified output flow

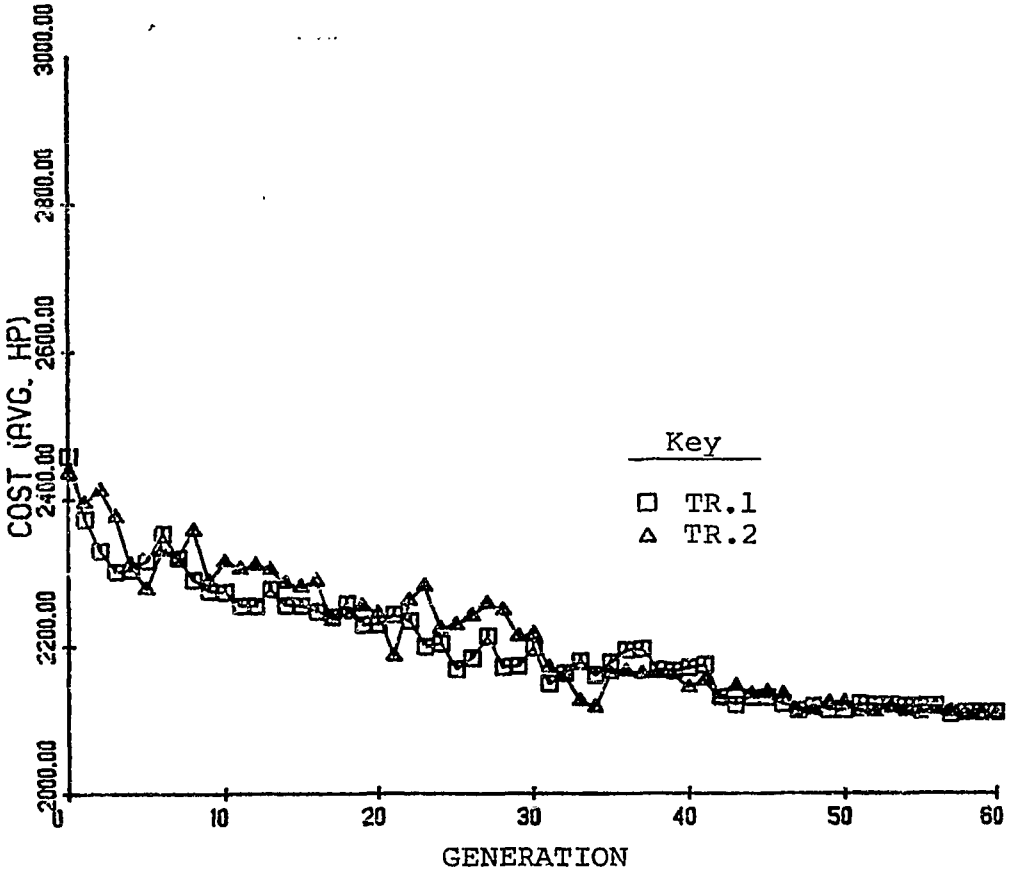


Fig. 4-7. Best-of-Generation Results - Single Line Transient Problem

Table 4-5

Best-Of-Run Results - Single Line Transient Problem

Run	Cost Avg. HP	Power Only	%Δ Optimal	Penalty	% of Optimal
tr.1	2107	2077	+2.4	30	1.5
tr.2	2107	2001	-1.3	106	5.2
average	2107	2039		68	3.4
optimal	2028	2028			

time history.

4.7 Good News and Bad News

As the old story goes, we have some good news and some bad news. The good news is that genetic algorithms have demonstrated their efficacy as improvement finding algorithms in two separate practical engineering optimization problems. The bad news is that absolute convergence to the best is not guaranteed. In this section, we take a closer look at the reasons for the "bad news" and some possible solutions within the genetic algorithm context. We also take a look at the "good news": the genetic algorithms simplicity and power.

Suboptimal Performance and Premature Convergence

In both the steady state and transient problem, an interesting thing has been observed: near-optimal results are easily obtained, while further improvement is difficult.

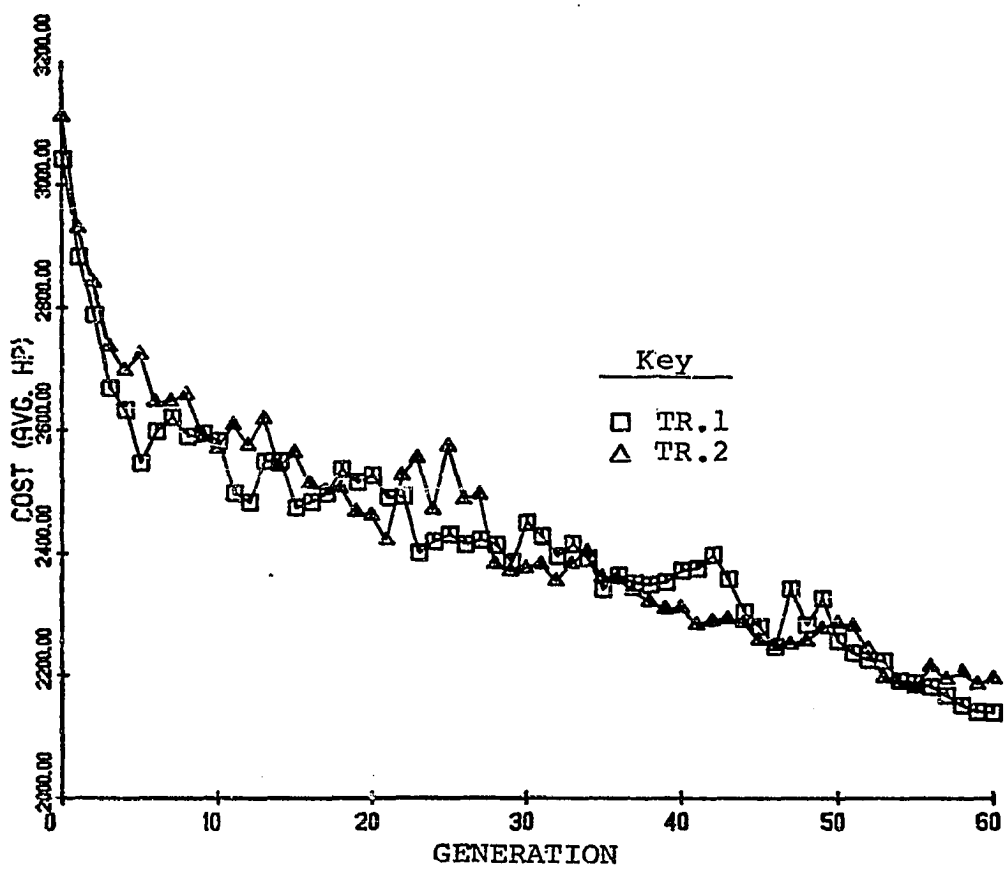


Fig. 4-8. Generation Average Results - Single Line Transient Problem

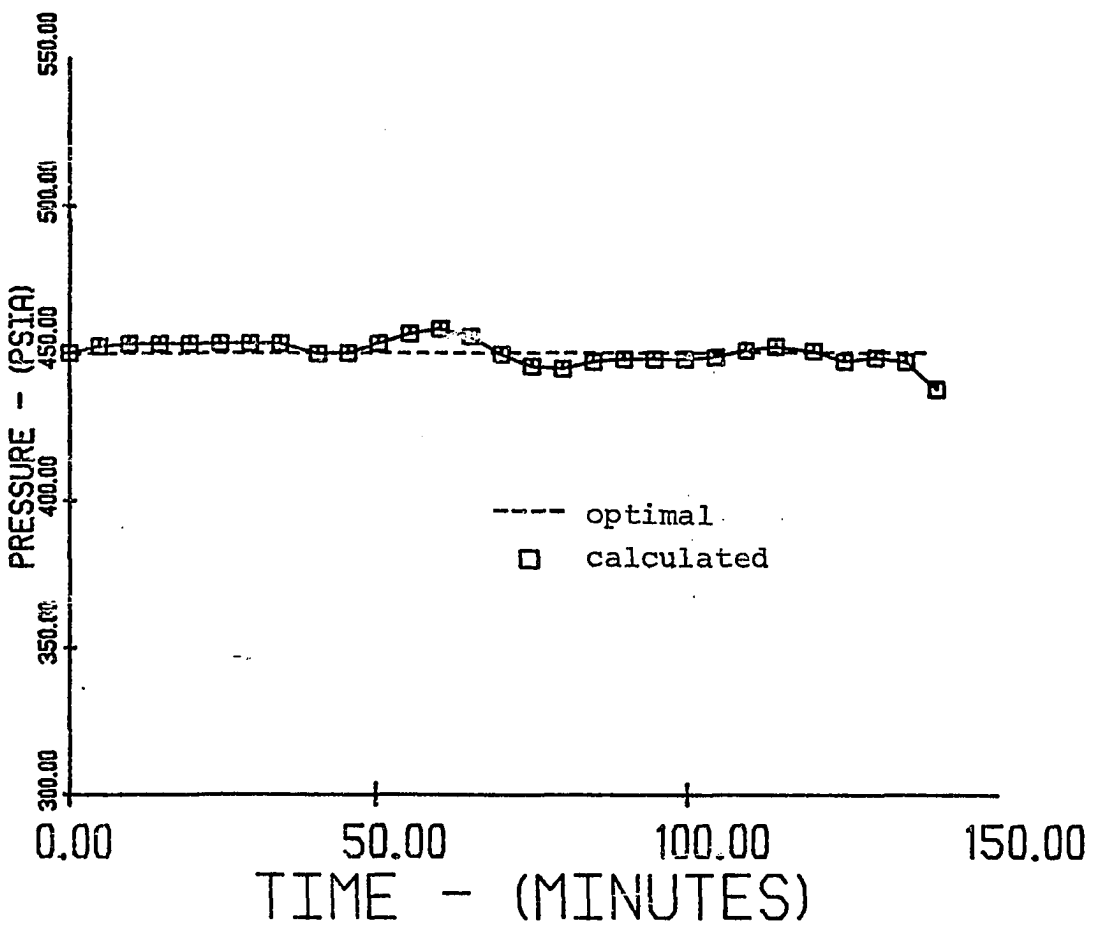


Fig. 4-9. Pressure Time-History - Run TR.1 - Single Line Transient Problem

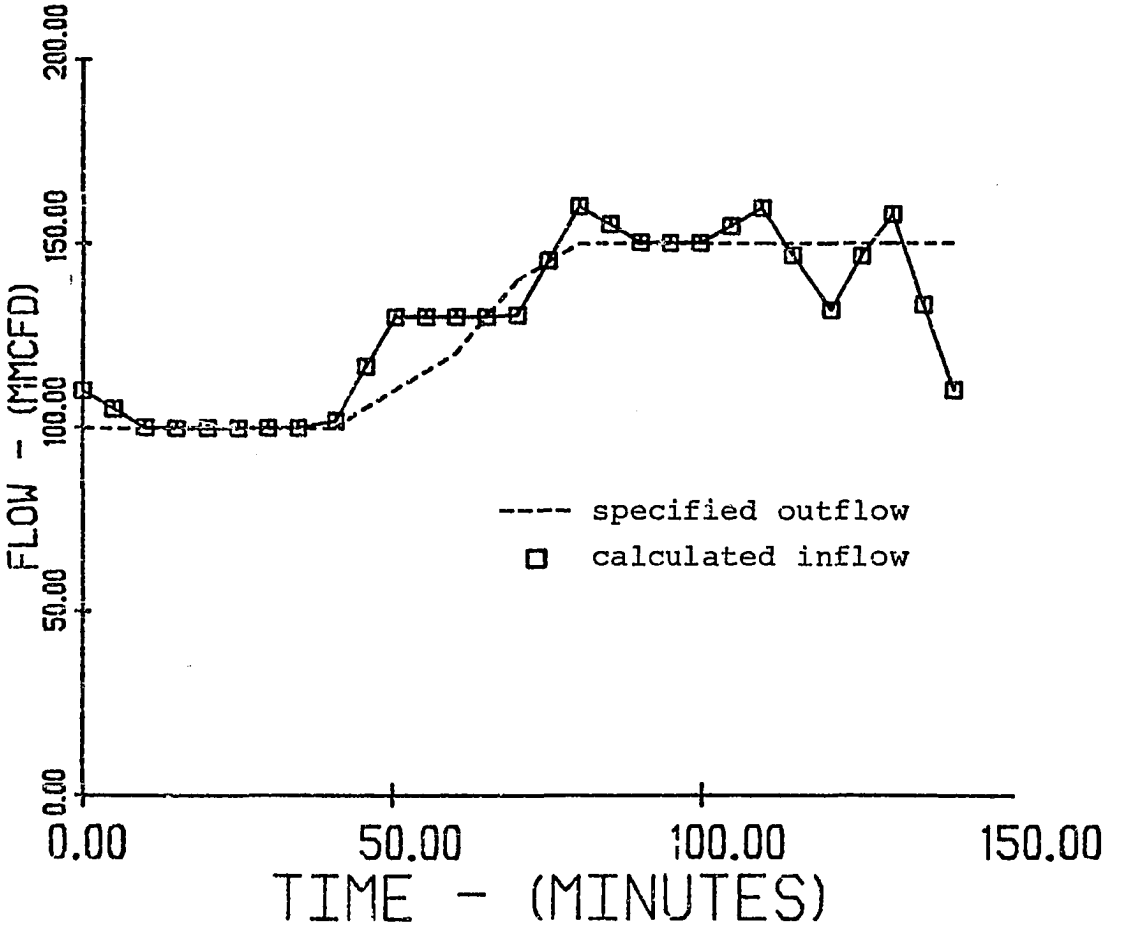


Fig. 4-10. Flow Time-history - Run TR.1
Single Line Transient Problem

Furthermore, examination of the strings contained in the population toward the end of the run shows substantial convergence at all bit positions. Both suboptimal performance and premature convergence have been observed previously by other authors [49,53]. We examine three contributing factors to account for this behavior: discretization and constraints, genetic drift and genetic algorithm hardness.

One reason the genetic algorithm results cannot match the continuous optimum is because we are not solving the same problem. Discretization of the parameter space and introduction of penalty functions transform the original problem so the actual optimum is unattainable. Discretization implies that we are only examining a finite number of points in the continuous parameter space; it is unlikely (an event with zero probability) that the discrete optimum matches the continuous optimum. Furthermore, the introduction of a penalty method with finite penalty coefficients also modifies the problem. Only as the penalty coefficients go to infinity can we hope to be solving the same problem. While discretization and penalties may account for suboptimal performance, neither of these problems is totally responsible for the premature convergence we have observed. Additionally, both discretization and penalty selection may be controlled by the user to get as close as is necessary and practical to the original constrained problem. Therefore, we must turn

to more fundamental explanations of genetic algorithm behavior to explain the phenomenon of premature convergence.

Premature convergence to suboptimal results has been observed in empirical studies by both Cavicchio [49] and De Jong [53]. De Jong likened the primary cause of this behavior 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. We see this more clearly if we again look at our reproductive plan in some detail.

We recall that in reproduction the number of copies is proportional to the normalized fitness $u/\bar{u}$. This rate of sampling has been identified as a near-optimal, realizable strategy. In actual implementations, we must reconcile the fractional nature of the quotient, $u/\bar{u}$, with the need for an integer number of offspring. In this study, we have used a probabilistic rounding, where the population count is rounded up with a biased coin toss, using the fractional part as the bias. This procedure is simple and gives the correct expected number of offspring when large populations are involved. Furthermore, the method adopted here is an improvement over earlier methods which selected N reproduction candidates from the population at large using the selection probabilities $p_i/\Sigma p$; nonetheless, the biased flip of a coin is a high variance process. As a result, excursions from the expected and near-optimal sampling rate

are commonplace. It is this difference that largely permits the drift toward premature, suboptimal convergence.

The idea of reducing the variance can and has been extended further. One particularly interesting approach treats the fractional part of the fitness like an interest payment. The interest is compounded over a number of generations until it compounds to create another individual. While not adopted in this study, this variance reduction method provides a way of reducing the effects of premature convergence due to genetic drift.

While genetic drift is a primary ingredient in premature convergence, there is another important reason why the simple genetic algorithm may converge to suboptimal results: the problem may be genetic algorithm hard (GA-Hard). A recent study by Bethke [54] has rigorously explored whether or not a problem is difficult for the simple, three rule genetic algorithm. In a previous section, we saw how the genetic algorithm depends upon the assembly of short building blocks. Crossover permits near-optimal sampling of short schemata, but effectively destroys longer ones. As a result, in problems where short building blocks do not correctly predict the optimum, we may naturally get convergence to suboptimal points. Although Bethke has developed an approach for determining GA-Hard problems by identifying schema average fitnesses with a Walsh function analysis, the approach is cumbersome for problems without simple analytical description. In

generalizing his investigation he has suggested that these GA-Hard problems tend to have well-isolated optima: this conclusion is limited to single parameter problems using a normal fixed point coding.

In the large, there is little evidence that either of the problems we have solved are GA-Hard. Short schemata building blocks have reliably led to near-optimal portions of the space. In the neighborhood of the optimum, the problems may be partially difficult for the genetic algorithm. In both problems, the parameter cost surface is relatively flat near the optimum. This, together with an interaction of the penalized constraints and relatively crude discretization create the opportunity for irregular multi-modality. This kind of irregularity may be difficult for the genetic algorithm to exploit, however, we must consider it to be a secondary effect when compared to the problem of genetic drift.

The primary method of improving genetic algorithm performance on GA-Hard problems is to seek a reordering of bit positions which more naturally permits short building blocks to lead to the optimum. Parameter interleaving or specified bit maps as described previously are one possibility for this. The more general solution is to incorporate the inversion operator [ANAS Ch. 6]. This may be necessary in tackling tougher problems with longer string representations.

Other methods may be useful for improving convergence.

Bethke [54] has suggested hybrid techniques where a genetic algorithm is used to define potentially important areas of a space and a gradient searcher is used to converge to some localized peak. This idea is attractive as it combines the global perspective of the genetic procedure with the convergence characteristics of calculus-based schemes. We might also consider a hybrid scheme where instead of reverting to the original parameter space for the gradient search, we consider the 1 bit string representation as an 1 dimensional parameter space. In this 1-space, single bit changes are analogous to numerical evaluation of derivatives. Numerous strategies may then be used to combine promising single bit changes into potentially promising multiple bit moves. This procedure is, thus, analogous to a gradient search in the 1-space.

Genetic Algorithm Strengths

In trying to portray an accurate picture of genetic algorithm performance we have investigated a number of genetic algorithm difficulties. Methods have been suggested to answer each of these within the spirit of the genetic algorithm methodology. In focusing upon problems, it is possible to forget the strengths of the genetic algorithm approach. This would be a mistake, as the method's strengths are manifold.

First on the strength side of the score sheet is simplicity. Genetic algorithms are elegantly simple in operation and application. String copying, substring

swapping, and random number generation are the only essential operations required. Application to different problem domains is exceedingly straightforward as we have shown. In the present study, the genetic algorithm and decoding routines for both the steady and transient problems are identical; only the system model, objectives, and constraints have been changed. This simplicity of application results in a clean interface between optimizer and model. The usual genetic algorithm interface involves passing a string down to the model and a fitness back up to the genetic algorithm. This simple interface encourages clean, modular programming; extant modules may be used with little or no modification to either model or optimizer. This is in stark contrast to many optimization methods. Dynamic programming and other clever enumerative schemes, for example, depend upon an unholy mixture of model and improvement algorithm to effect a solution. Calculus-based methods also involve a more complex interaction because at the very least, derivative information is necessary for the improvement algorithm.

The clean interface issue is not simply one of elegance in programming; it also determines where the methods may be applied. For example, dynamic programming and calculus-based methods are generally inappropriate in situations where no model exists as in the direct control of a prototype system. Genetic algorithms have no such restrictions because they only require payoff information.

While simple, genetic algorithms are powerful in their quest for improvement. In this study, two problems with large problem domains (10^{12}-10^{13} alternatives) have been solved using the simple reproduction, crossover, mutation procedure. In all cases, near-optimal results have been obtained in a relatively small number of function evaluations. Furthermore, acceptable interim behavior is noted because of the rapid initial improvement phase in both cases. Genetic algorithms obtain acceptable performance quickly and fine tune performance at a more leisurely pace. This kind of behavior is desirable in practice. As we pointed out when we discussed the goals of optimization, finding the best is not usually important; reliably finding an acceptable solution is.

No small part of the genetic algorithm's power derives from its global perspective. Genetic algorithms work from a database of diverse points. As a result, during the search, many hills are climbed simultaneously. This helps eliminate the myopia of typical algorithms which work from a single point.

Genetic algorithms are problem independent; they are truly a canonical search method. The present study has investigated two very different problems successfully with the same procedure. Previous studies have investigated genetic algorithms in a variety of domains: smooth and discontinuous, deterministic and noisy, unimodal and multi-modal. The combination of this breadth of performance and

power of effect supports the claim that genetic algorithms are robust.

Last, in this study, we have demonstrated the intuitive appeal of genetic algorithms. The valuable notion exchange heuristic which underlies genetic algorithm performance is dramatically similar to human innovative thought. We make no claim that the simple genetic algorithm captures the full richness of human innovation; yet in its simplicity, it bears some strong resemblances. We note that this heuristic is largely an inductive procedure. Genetic algorithms by their nature, generalize from specific example; this is a breath of fresh air among other search algorithms which are methodically deductive.

4.8 Summary

In this chapter, we have applied the genetic algorithm approach to two practical problems in pipeline control: long term optimization of a serial system and transient optimal control of a single line. The applications have proven successful in independent trials on both problems: near-optimal results have been obtained using an infinitesimal sampling of the possible alternatives. Along the way, we have also addressed some practical issues in using genetic algorithms on engineering problems.

To apply genetic algorithms to practical problems, we concern ourselves with four things: discretization and coding, fitness mapping, constraints, and genetic algorithm parameters. Discretization may appear at two levels. In

continuous control problems, the time or space continuum is reduced to a finite space by assuming some convenient functional form. For example, in the transient problem we consider piecewise linear interpolation of the control variable (mass flux) over equidistant time intervals. Once we have reduced to a finite number of parameters, we must discretize the individual parameters and combine them to form a complete string. In this study, all parameters have been coded in mapped, fixed point form. This type of coding gives uniform precision over a well-known interval. We also have discussed how shortened floating point codes and other exotica may be useful for other problems. In this study, strings have been formed in both problems by the simple concatenation of the individual parameters. The computer implementation allows for a specified bit map; however, this feature did not prove necessary as the performance has been deemed satisfactory.

Genetic algorithms are designed for unconstrained problems; to handle constraints we must transform a constrained problem to an unconstrained formulation with a penalty method. A cost is associated with constraint violation; in this study, the cost is proportional to the square of the constraint violations. A sizing procedure has been developed for calculating reasonable penalty coefficients; coefficients are chosen so that nominal violations result in penalties which are a percentage, usually 10% of some nominal cost.

Fitness mapping may be necessary depending upon the problem formulation. The normal genetic algorithm depends upon the maximization of some non-negative figure of merit, the fitness. In minimization problems, we must transform the as-formulated objective function to a proper fitness function form. In this study, we use a simple mapping function: we subtract the minimized objective function value from a specified constant and arbitrarily assign all negative values to zero.

While the parameters of the genetic algorithm affect its performance, nominal values of population size, crossover probability, and mutation probability have yielded acceptable engineering results. These parameters may be fine-tuned--or adapted automatically--to get peak performance; however, this should rarely be necessary for practical results.

The steady state and transient problems have been posed using the problem formulations of Wong and Larson [29]. In the steady problem, we seek the least power operation of a serial system of ten compressors and ten pipes. Maximum and minimum pressure and compression ratio constraints are specified. For this problem, the string is interpreted as the concatenation of 10-four bit substring parameters. Each parameter is a mapped, fixed point code representing the control parameter U_i (ΔP^2) for each of the compressors. In three independent trials of the genetic algorithm with fixed genetic parameters, near-optimal performance is obtained.

In all three cases, power cost alone is very near-optimal (<5%), while nominal pressure and compression ratio violations cause a total penalty violation cost ranging between 10 and 30 percent.

In the transient problem, with known demand time history, we seek the supply schedule which minimizes energy consumption, subject to maximum and minimum pressure constraints. To solve this via genetic algorithm, we first discretize the continuous control schedule into 14-ten minute intervals. A parameter is associated with each interval endpoint and the flow is assumed to vary linearly in between. The 15 parameter sequence is represented by the concatenation of 3 bit substrings using a mapped fixed point coding scheme. Two independent trials of the problem have been performed using the same genetic algorithm and parameters. In both cases, very near-optimal performance has been been determined. The energy-only cost is within 2.5% of the optimal value, while the penalty cost adds less than 6% of the optimal value. Wong and Larson have found that the optimal solution is to just satisfy the minimum pressure constraint at the downstream end for the duration of the simulation. The genetic algorithm solutions follow this trend as well; close examination of one solution shows the pressure hugging the minimum pressure constraint of 450 psia.

While it is encouraging to consistently obtain near-optimal performance, in both problems we note a form of

convergence at most bit positions by the end of each run. We have seen how discretization and penalty methods transform the original problem so we do not expect to obtain the continuous constrained optimum exactly; but, these effects do not adequately explain this premature convergence. Genetic drift has been identified as the primary cause of this problem. The difference between a string's expected reproduction rate and its actual realization may cause small populations to drift away from the proper course. Variance reduction methods have been suggested to mitigate this problem.

Another cause of premature convergence is problem hardness (GA-hard problems). If short building blocks do not reliably predict the optimum, premature convergence may result. In the problems investigated here, there is little evidence of GA hardness. For problems which do display such difficulty, the inversion operator may be necessary. This operator attempts to find satisfactory re-orderings of bit positions so short building blocks predict improvement more reliably; thus far, inversion has not proved necessary for problems with modest string lengths (<50).

While we have noted these difficulties, we also have demonstrated some of the genetic algorithm's many strengths. Genetic algorithms are simple in operation and application. Yet, in their simplicity they have demonstrated great power to search complex spaces quickly. Furthermore, they appeal to our own sense of innovation. Unlike many optimization

procedures, genetic algorithms are inductive, they generalize from specific instances. In a sense, the process is a more human-like search; it is bold and synergistic. It does not plod mechanically from point to point. Instead, it combines its best ideas to speculate on improved performance. As a result, genetic algorithms hold great promise, not only in traditional problems, but as the fundamental search mechanism in a learning system. In the next chapter, we explore this application.

CHAPTER 5

A LEARNING CLASSIFIER SYSTEM

We started this study with the goal of designing, constructing, and testing broadly applicable algorithms for learning and decision making. In one sense, our study of optimization with genetic algorithms in the previous two chapters has been a digression from this goal because we know full well that optimization is too rigid a methodology to be trusted to control even fairly simple systems. In another sense, this work is germane to our goal because it has helped us examine the genetic algorithm's innovative flair for searching rapidly through arbitrary string spaces; they seem more human-like a search mechanism than others we commonly encounter. What then is the problem? Why can't we unleash this innovation in more complex, less completely defined environments? The problem lies not with the genetic algorithm, but rather, with the structures we choose to adapt.

In this chapter, we overcome this difficulty by changing the adapted structure. A learning system is described and tested based upon a population of string rules. The string rules use both environmental information

and the current internal state to decide what to do and what to think next. A genetic algorithm generates new, possibly better, rules for inclusion in the population. In this way, the innovation of the GA is used to reprogram the system with better and better rules. At first we confine our attention to the origins of these rule systems, called learning classifier systems (LCS). We describe in broad terms the operation and structure of the LCS and then test its operation on a simpler problem domain than the ultimate pipeline control setup: one-dimensional control of a frictionless inertial object. By doing this, we can test the simple LCS without the complexities of a more sophisticated operating environment.

5.1 Learning Classifier Systems - Overview

A learning classifier system is an artificial system that learns rules, called classifiers, to guide its interaction in some specified environment. Learning classifier systems are the latest outgrowth of Holland's continuing work on adaptive systems.

In 1962, when Holland outlined his theory of adaptive systems [60], he developed a general theory encompassing many systems but ultimately he was addressing himself toward machines who could program themselves:

> The study of adaptation involves the study of both the adaptive system and its environment. In general terms, it is a study of how systems can generate procedures enabling them to adjust efficiently to their environments. If adaptability is not to be arbitrarily restricted at the outset, the adapting system must be able to generate any method or procedure capable of an

effective definition.

With this foundation, more concrete suggestions emerged for classes of schemata processors [61] which in some limited respects resemble the present day LCS. This work has evolved into the intricately interesting, but as yet unimplemented, broadcast language [1]. The first practical implementation of a learning system based on these theories appeared in 1978. Holland and Reitman [62] describe this first Learning Classifier System which learns a simple maze running task. Though the task is simple, the achievement is remarkable because of its successful marriage of a rule-based knowledge system and a genetic algorithm for discovery of new rules. Holland [63,64] is continuing this work with construction of a user-tailored information retrieval system. Booker [65] has recently completed his study of an LCS-based artificial creature learning to survive in a two dimensional domain containing both food and noxious substances. His work goes to great length to tie the LCS and Holland Adaptive Systems Framework to current work in cognitive science. Wilson [66] is applying the LCS concept to visual pattern recognition. With this as historical background, we need to examine the elements of an LCS to see why it holds promise as an effective learner and decision maker.

A learning classifier system is composed of three main elements:

1. Rule and Message System
2. Apportionment of Credit System
3. Genetic Algorithm

The rule and message system of an LCS is a special kind of production system. In computer science parlance, a production system is a computational scheme which uses rules as its only algorithmic device. Although there is a wide variety of syntax among production systems, the rules are generally of the following form:

if <condition> then <action>

Semantically, the action is taken if the given condition is satisfied.

At first blush, the restriction to such a simple device for the representation of knowledge might seem too constraining. Yet, it has been shown that production systems are computationally complete [67-69]. Their power in representing knowledge involves more than this. They are also computationally convenient. A single production or small group of productions can often represent a complex set of ideas. In procedural languages, FORTRAN, ALGOL, etc. it is rare that a single statement represents a complete thought.

While completeness and convenience are important, we are also drawn toward rule-based systems because, in some sense, human operators seem to store their knowledge in rule form. Earlier on, we noticed in our informal survey of the gas dispatcher's environment, that a gas dispatcher, when

talking about his operation of the system, chose to describe his knowledge in rule-of-thumb form. We repeat the selected rules below:

- If you are losing 10-15 psi/hr then you must take corrective action
- If during a 6 hr period you lose 70 psi of linepack then replenish before moderating.
- Try to maintain 700 psi at W_____ (a location) during the winter.

As we expect, the rules are simple in form but powerful in their summary of a large deal of experience. In designing an artificial learner and decision maker, it seems reasonable to select a similar rule-based structure.

With their simplicity, power, and common sense appeal, it is no wonder that productions systems have been very useful in representing expert knowledge in artificial intelligence systems. Two of the best known and successful expert systems, DENDRAL [70] (Mass Spectroscopy Analysis) and MYCIN [71] (Bacterial Infection Diagnosis), use production systems for their representation of knowledge. More recently and perhaps of more interest to engineers, the Prospector system [72] has been constructed as a geological field assistant to determine areas with high mineral deposits. Other engineering expert systems have been constructed to aid in oil well drilling [73], assist in seismic data analysis, analyze oil well logs, site hydropower developments, and manage a nuclear reactor [74].

Yet, production systems have been less frequently suggested in situations in need of learning. One of the

main obstacles to learning has been complex production syntax. Many productions systems permit involved grammatical constructions for the condition and action portions of a rule. Learning classifier systems depart from the mainstream by restricting the rule (classifier) to a fixed length representation. This has two benefits. First, all strings under the permissible alphabet are syntactically meaningful; this is not true in many production systems and most procedural languages. Second, a fixed representation permits meaningful string operators of the genetic kind. This leaves the door open for a genetic algorithm search of the space of permissible classifiers.

In traditional expert systems, the value or rating of a rule relative to other rules is fixed by the programmer in conjunction with the expert or group of experts being emulated. In a rule learning system, we don't have this luxury. The relative value of different rules is one of the key pieces of information which must be learned. To facilitate this type of learning, Holland has suggested that rules function in a competitive service economy. A competition is held among classifiers where the right to answer relevant messages goes to the highest bidders with this payment serving as a source of income to previously successful message senders. In this way, a chain of middlemen is formed from manufacturer (source message) to consumer (environmental action and payoff). The competitive nature of the economy insures that the good rules

(profitable) survive and the unsuccessful die off.

While we shall soon examine the many details of this apportionment of credit algorithm, one point is crucial: the introduction of an internal currency or figure of merit. The exchange and accumulation of an internal currency provides a natural measure for the application of genetic algorithms. Using the classifier's net worth (which we shall shortly call strength) as a fitness function, classifiers may be reproduced, crossed, and mutated as we have done in Chapters 3 and 4. Thus, not only can the system learn by evaluating and ranking existing rules, but new rules, the offspring of high performance rules, are inserted into the population by a genetic algorithm. We must be a little less cavalier about generating entirely new populations of rules, and we pay more attention to who gets replaced; however, the process is very similar to the one used in our optimization studies.

Together, apportionment of credit via competition and search with genetic operators form a powerful learning heuristic when combined with the computationally convenient and complete framework of classifiers. In the following, we examine the structure of the learning classifier system by detailing each of the component parts: the rule and message system, apportionment of credit system, and genetic algorithm system.

5.2 The Rule and Message System

The rule and message system is central to the operation

of the learning classifier system. Not only does it provide the computational framework for LCS thought and action, it also is the backbone of the competitive service economy and GA learning functions. To see this, Figure 5-1 shows a schematic of the rule and message system integrated with the apportionment of credit and genetic algorithm processes. In this schematic, we see that the rule and message system receives environmental information through its sensors, called detectors, which decode 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. Additionally, certain messages may call for external action through a number of action triggers called effectors. In this way, the rule and message system combines both external and internal data to guide behavior and the state of mind in the next state cycle.

The process is like a popular mode of communication at our widget conventions. At these conventions, widget conventioneers reach their fellow widget delegates by posting notes on the widget convention bulletin board. The notes may be posted by anyone (as long as there is room) and may be addressed to individuals, widget committees, or other groupings of widget delegates with something in common. In

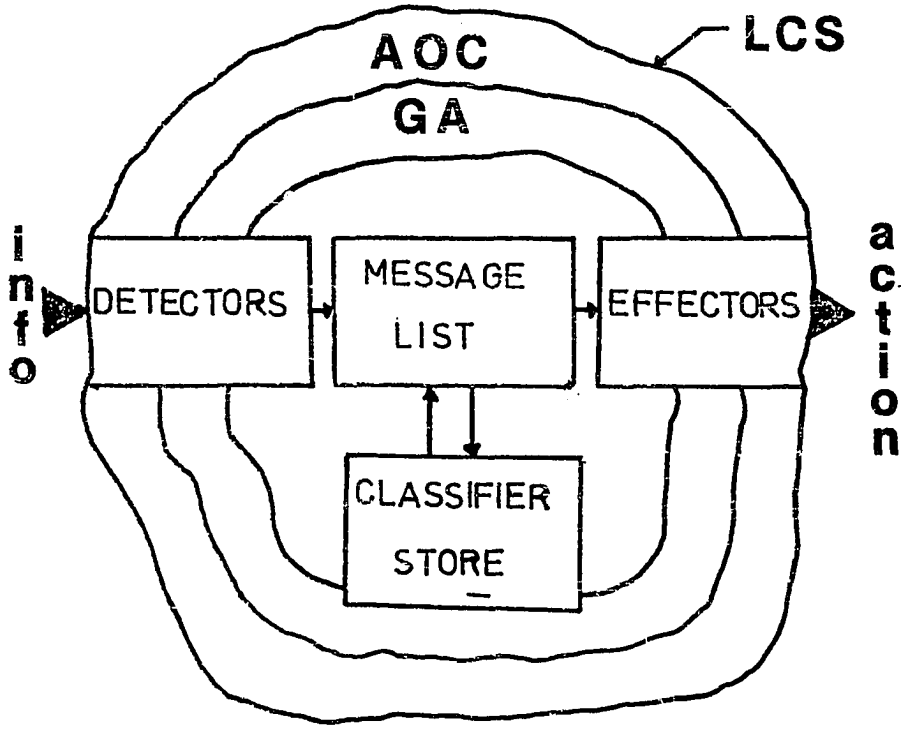


Fig. 5-1. Schematic - Learning Classifier System

this way, individual widgeteers rendezvous, committees change or set their plans, and in general, the direction of the convention is altered. Similarly, classifiers and effectors are accessed and activated by the common communication channel of the message list. Thus, the LCS is able to change subsequent internal state and external action.

To better understand the message processing action, we examine the system's two informational units:

1. Messages
2. Classifiers

In the LCS sense, a message is simply a string of fixed length l, over some finite alphabet, V. In this discussion, we limit V 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 data, internal tags, internal data, and effector codings.

Messages are processed by classifiers. Recall that classifiers are a form of rule in the tradition of production systems. For this study, we limit classifiers to the following form:

$$\langle classifier \rangle \rightarrow \langle condition_1 \rangle \langle condition_2 \rangle \langle message \rangle$$

As in the production systems discussed earlier, the meaning of the classifier is clear: the message is sent

upon satisfaction of both conditions. A condition is a recognition device which depends upon the presence of certain messages on the message list. It would be nice if a single condition could recognize not just a single message, but rather, a class of messages with well-defined similarity. We may achieve this capability quite simply and elegantly by extending our message alphabet V by one character to the alphabet V+ = {0,1,#}. Thus, a condition is defined as an 1 position string over V+:

$$\text{<condition>} \text{ --> } \{0,1,\#\}^1$$

Under the alphabet V+, 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 condition #1111 is triggered by either of the messages 01111 or 11111. At the other extreme, the condition #1### fires on any of the $2^4 = 16$ messages with a 1 in the second position. In this way, the # is a wild card symbol permitting explicit recognition of any of the subsets of messages with one or more similarities. This is particularly useful in a learning system that must generalize and instantiate new rules from the ratings of the current rule store.

The mechanism of the rule and message system is fairly straightforward; however, to reinforce these ideas, let us examine a simple example. In Table 5-1, we see a single iteration of the rule and message system. In this example, we have a message list with two messages, $n_{mess} = 2$, four classifiers in the store, $n_{class} = 4$ and string width of 4

positions, 1 = 4. In the example, the classifiers numbered 1 and 3 are matched completely and therefore send their messages to the message list in the next time step (for simplicity, we have ignored any environmental messages). This raises an interesting question: What would happen if we matched more classifiers, that is, the number of potential messages exceeded the size of the message list? We must be concerned with this question because we are very likely to have many matched classifiers with a severely restricted message list. In the next section, we examine how the apportionment of credit mechanism handles this and other conflicts which arise.

Together, the picture of the rule and message system is complete. Messages, either environmental messages or internal messages, are placed on the finite size message list. In turn, these messages may either match effectors which cause external action or they match other classifiers which may in turn send internal messages. In this way, the rule and message system promotes behavior which depends upon external stimulus and internal state of mind.

5.3 Apportionment of Credit

We now see how the rule and message system provides a convenient method of storing and using rules for performance in arbitrary environments. Yet, we are left with several pressing questions: How can we select among many potentially active classifiers when the communication channel (the message list) is of finite size? How can we

Table 5-1

Example Cycle of Rule and Message System

Message List (t)	Classifier Store (t)	Matches		Message List (t+1)	Sent by
		$Cond_1$	$Cond_2$		
m1) 0010 m2) 1101	c1) 1###:##10/0001 c2) 01##:####/0010 c3) 110#:0###/1110 c4) ###1:#11#/1101	m2 - m2 m2	m1 m1,m2 m1 -	m1) 0001 m2) 1110	c1 c3

apportion credit for performance among different rules which may call each other in complex ways? As we have already suggested, our answers to these questions are related to the creation of an internal, competitive service economy.

This service economy consists of two elements:

1. An auction
2. A payment clearinghouse

During the auction, classifiers matched by the message list bid for the right to send messages to one of the n_{mess} slots of the next time step. Following the auction, the winning classifiers make payment to the clearinghouse where each payment is divided among all those classifiers responsible for activating the particular payment-making classifier.

We can see one complete cycle of this payment process in Figure 5-2. In this diagram, classifiers 10, 20, and 55 are activated auction winners at time t and as a result, they send their messages to the 3 position message list (n_{mess}=3). After another auction, classifiers 4, 6, and 19 are activated by the messages as shown. In consideration of this activation, classifiers 4, 6, and 19 make payments which are divided among the time t classifiers. As examples, c_4's payment is divided among all three classifiers (c_{10},c_{20} and c_{55}) whereas c_{19}'s payment is only paid to c_{55}. Hence, classifiers receive payment from the outside world whereby they then distribute payment amongst themselves and accumulate payment for their own accounts.

To implement a well-defined procedure, we must be a bit

AOC

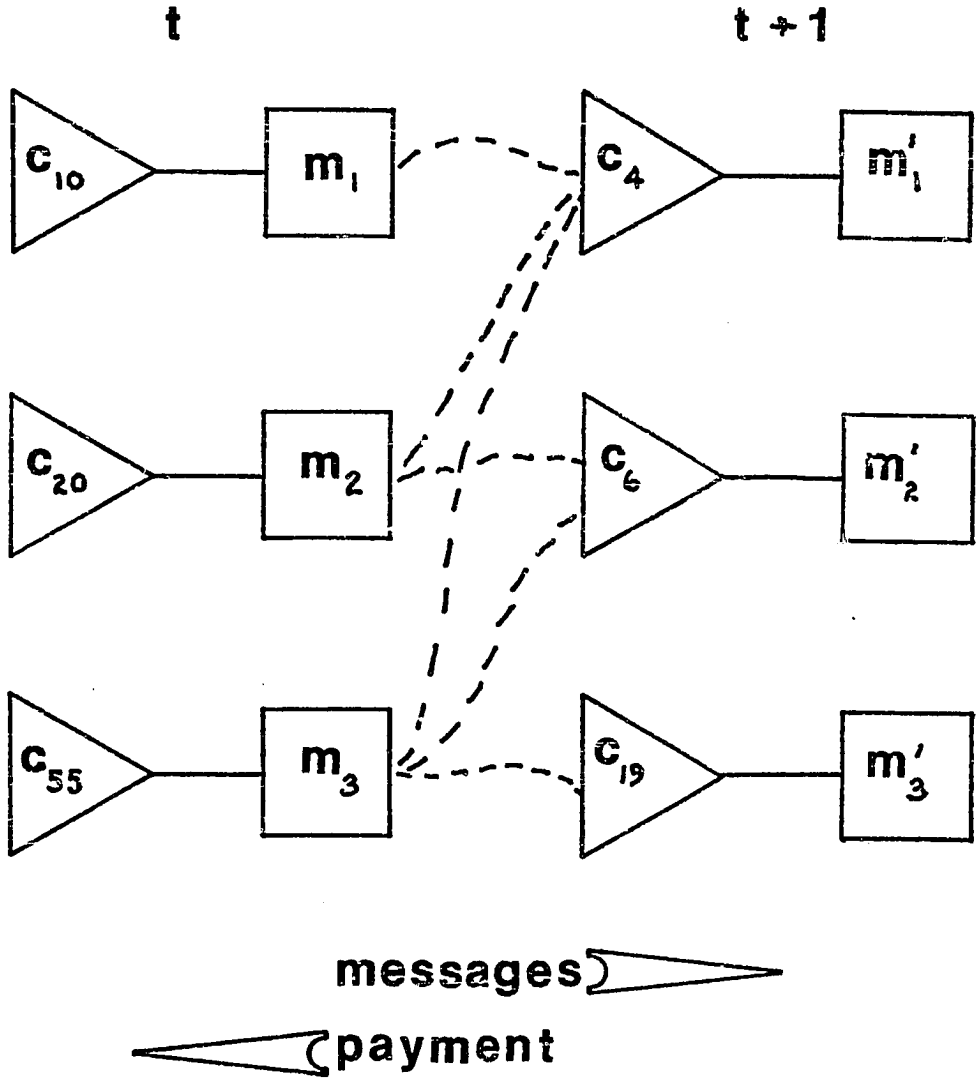


Fig. 5-2. Apportionment of Credit - Paying and Receiving Bids

more rigorous in detailing the auction and payment scheme. During the auction, classifiers make bids designated by the letter B. Winning classifiers turn over their bids, B, to the clearinghouse as payments, P. A classifier may also have receipts from its previous message sending activities; we designate a classifier's receipts by the letter R. In addition to bids and receipts, we also permit one or more forms of taxation, T. For each classifier, the receipts and payments are made to and from a single bank account. The classifier's account balance is called its strength, S. In essence, strength is a classifier's net worth; we shall see how it is related to its ability to make a profit by setting up subsequent reward.

Taken together, we write an equation governing depletion or accretion of a classifier's strength as follows:

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t)$$

where:
S - strength
P - payment
T - taxation
R - receipt
t - time index
i - classifier index

This system of first-order difference equations is the major component of our apportionment of credit scheme. To understand its effect upon classifier activation and utilization, we look at the circumstances of bidding, receipts and taxation. We also consider the detail of effector activation and reinforcement.

Bidding and the Auction

During the auction, the n_{mess} highest bidding classifiers are chosen to send their messages in the next time step. Each classifier's bid must reflect its value to the system in setting up fruitful action and subsequent reward. One component of this value is a classifiers strength. Strength is a measure of the classifier's relative ability to profit by receiving external reward through classifier chains of payment. Another component of value is a classifier's relevance to the matching messages. Not only do we want winning classifiers to be strong, we also require them to be strongly related to their activating messages. One measure of this relevance relationship is the simple matchscore:

$$M = \Sigma \Sigma m(a_{kj})$$

where: k - condition index
j - position index
a - position value
m(a) - 0 if a=#
1 if a=1 or 0

The matchscore is, thus, a simple count of a matched classifiers total specificity, the number of condition positions with non-wild characters.

Since we are interested in promoting classifiers with high strength and high specificity, following Holland [63,64], we define our bid to be proportional to the product of matchscore and strength:

$$B_i = C_{bid} * M_i * S_i$$

where: B - bid
M - matchscore
S - strength
C_{bid} - constant
i - classifier index

We could simply stop things at this point and choose the auction winners deterministically by selecting the n_{mess} highest; however, this would unreasonably bias results towards the status quo [75]. Instead, we hold our auction in the presence of random noise. Specifically, we calculate an effective bid for each matched classifier, the sum of the deterministic bid and a noise generator:

$$EB_i = B_i + N_i(\sigma_{bid})$$

where: EB - effective bid
B - bid
N_i - noise generator
σ_{bid} - noise deviation

For this study, we use the noise generator defined by a 9 point discrete approximation to zero-mean, Gaussian random noise shown in Figure 5-3. The noise deviation, σ_{bid}, is a specified system parameter which may be varied to provide more or less randomness to the auction.

Receipts and the Clearinghouse

After our somewhat noisy auction and the selection of winners, payment must be made to those classifiers responsible for sending the messages that activated the winners. The winners pay their total bid to the clearinghouse where each payment is divided evenly among condition 1 and condition 2, and thence it is divided evenly

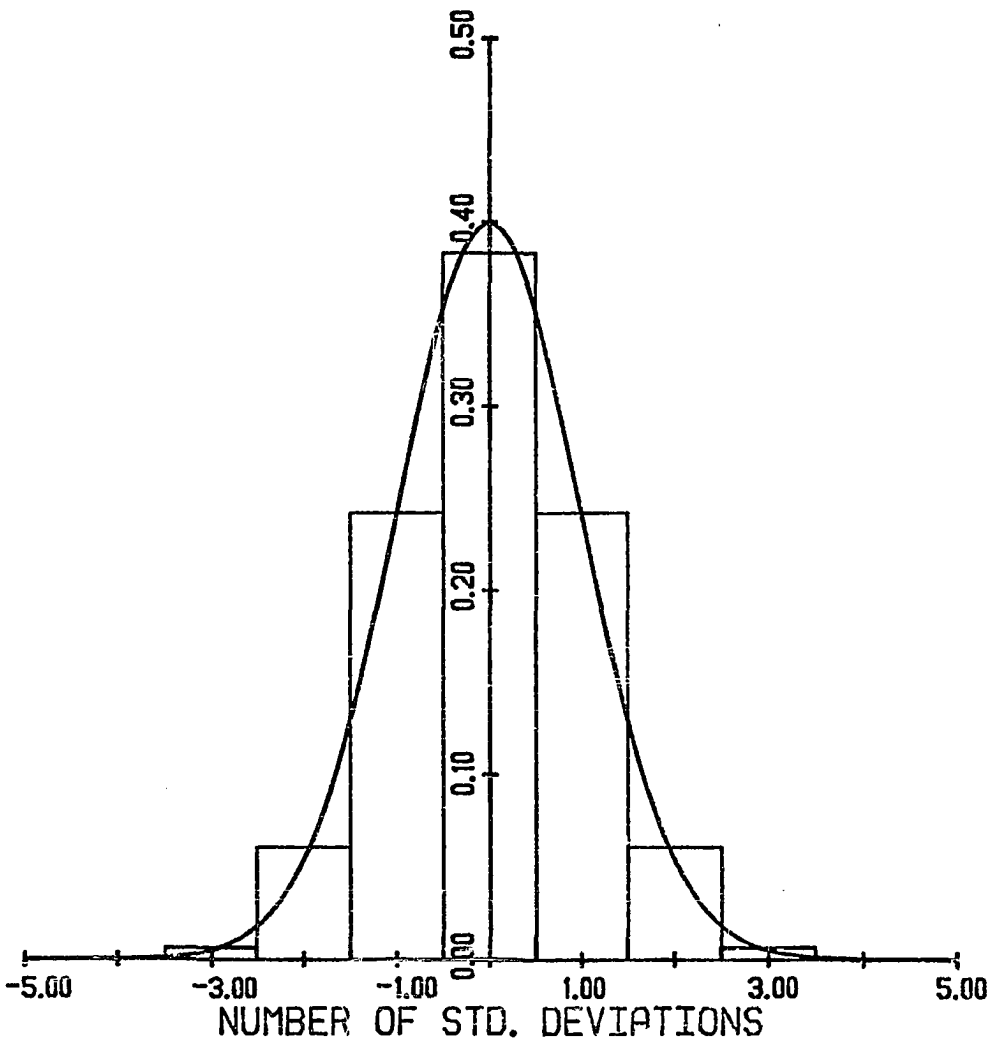


Fig. 5-3. Nine Point Discrete Approximation to Gaussian Distribution

among all the classifiers which sent matching messages to each of the conditions. Thus, a classifier sending a message which matches condition k of classifier j receives the share of payment, SP, given by the expression:

$$SP_i = P_j / (2*n_{sharek^j})$$

where: SP - share of payment
P - payment
n_{share} - number of shares
k - condition index
j - paying index
i - receiving index

For each message sending classifier, the total receipts are simply the sum of the share payments from all classifier conditions matched.

Taxation

Each classifier is taxed to prevent freeloading, thereby biasing the population toward productive rules. Many schemes are available; we simply collect a tax proportional to the classifier's strength:

$$T_i = C_{tax} * S_i$$

where: T - taxation
S - strength
C_{tax} - tax constant
i - classifier index

Activating Effectors

The foregoing mechanisms for bidding, payment, and activation are strictly true for pure classifiers, those whose only effect is to send a message. For those classifiers that ultimately set and perform an external

action (Holland calls these e-classifiers) we must be careful to arbitrate among mutually exclusive activities. Just as a switch cannot be both on and off at the same moment, we too must choose among competing alternative actions. In the prototype LCS, this is done very simply. After classifiers are matched and the auction is held, the winners' messages are immediately matched against the effector store. If different classifiers match mutually exclusive actions, the classifier-action pair with the highest effective bid is selected for operation. Note that the effective (noisy) bid is used, as with the pure classifiers, to eliminate bias toward existing higher strength e-classifiers.

Reinforcement

Intermittently, the LCS is rewarded with some payment from the environment. This reinforcement is given to all e-classifiers active during the previous time step. The reasoning behind this is that these most recent actions and their activating chains are most clearly responsible for the current reward. Holland has suggested that the entire message list receive reward; however, this has not been adopted here for fear of encouraging unproductive classifiers. This point should be kept in mind in future studies where these free-floating classifiers may be more useful in look-forward modeling or setting up future classifier-action chains.

Stability and Effect

We now have a fairly complete picture of the workings of the apportionment of credit algorithm; however, we need to explore the stability of this algorithm and its effect on rule ratings and selection.

To examine both of these, we recast the apportionment of credit equation into a more useful form where all payments and taxes have been replaced by their strength equivalent.

$$S(t+1) = S(t) - C_{bid}*M*S(t) - C_{tax}*S(t) + R(t)$$

We have dropped the index i and all terms are as defined previously. Grouping terms we obtain the following:

$$S(t+1) = (1-K)*S(t) + R(t)$$

$$\text{where:} \quad K = C_{bid}*M + C_{tax}$$

To see when this equation is stable we perform the usual Z transform [76] on the homogeneous system and obtain the characteristic equation:

$$Z - (1 - K) = 0$$

Stability is assured when $| Z | \leq 1$,which implies that $0 \leq K \leq 2$; however, in practice we never permit $K > 1$ to enforce non-negativity of the strength. This analysis is only fully valid for the classifier which remains activated thereby maintaining a constant K. However, the system remains stable even with the switching non-linearity introduced by the activation and deactivation of real classifiers as long as the changing K meets the stability criterion.

Stability is essential, but to see the effect of the mechanism, we are primarily concerned with performance in the time domain. Assuming some initial strength, S(0), we may define the strength on the nth time step by the expression:

$$S(n) = (1 - K)^n S(0) + \Sigma\, R(j) \cdot (1 - K)^{n-j-1}$$

Once again we have ignored the switching non-linearity although this could be incorporated as a time-varying K(j).

To further identify the effect of this mechanism we examine the steady state response of the algorithm. If the process continues indefinitely with a constant receipt R(t) = R, we obtain the steady state strength, S(t), as t approaches infinity:

$$S_{ss} = R / K$$

Therefore, the strength is simply the receipt amplified by the gain coefficient 1 / K. Furthermore, the steady bid is derived directly:

$$B_{ss} = C_{bid} * M / K * R = C_{bid} * M / (C_{bid}*M + C_{tax}) *R$$

Since C_{tax} is usually quite small, the steady bid value is simply: $B_{ss} \approx R$. In other words, the bid value approaches the receipt. For non-constant inputs we see that the bid is a geometrically weighted average of the input. As such, it acts as a filter of the possibly intermittent and noisy receipt values.

5.4 Genetic Algorithm

The apportionment of credit algorithm gives us a clean method of valuing rules, deciding among alternatives, and

weeding out unprofitable rules. Yet, we still must come up with a way of injecting new rules into the system. This is where the genetic algorithm steps in. Using a genetic algorithm similar to the one described in Chapter 3, new rules are created by the now-familiar reproduction, crossover, and mutation process. These rules are then placed in the population and processed by the probabilistic auction, payment, and reinforcement mechanism to properly evaluate their role in the system. In this section, we concentrate on the differences between the GA used in the LCS and the one described in Chapter 3. Specifically, these differences include overlapping generations, roulette wheel selection, partial replacement and crowding, restricted crossover, and ternary mutation.

Overlapping Generations

In the previous description of the genetic algorithm, the populations were non-overlapping; we completely generated a new population at each iteration. This is not desirable for the current application. With the LCS, we are concerned with maintaining a high level of performance as the system adapts. This requires that we leave our current best rules alone and try to form better rules with a small portion of the population. To do this, we introduce the parameter, PROPORTION and generate n_{class} * PROPORTION new classifiers at each call to the genetic algorithm. To do this conveniently, we must modify our method of selection.

Roulette Wheel Selection

In Chapter 4, we described a variance limiting process, where we attempted to give each population member the correct expected number of offspring $u / \bar{u}$. Although this has certain advantages, it becomes cumbersome with overlapping generations and therefore, we abandon it. Instead, we select parents for mating via the weighted roulette wheel technique.

In this method, we spin a weighted roulette wheel, pictured in Figure 5-4, n_{class} * PROPORTION times where the wheel weights are given by $S_j / \sum S_i$. In this way, we still bias the parent selection toward high strength members and thereby schemata in the proper proportion; however, the bookkeeping is greatly simplified. If this tradeoff between programming simplicity and efficacy proves too deleterious, we may always return to more sophisticated techniques.

Replacement and Crowding

Because we no longer generate entire populations, we must be careful when choosing population members for replacement. While it makes sense to replace low strength members, simple replacement of the worst is probably not good enough. This encourages a higher loss of alleles than is desirable. To this end, we implement a crowding mechanism among a low performance sub-population. With this technique described by De Jong [53], when a child is generated for insertion into the population, $n_{replace}$ replacement candidates are selected from the

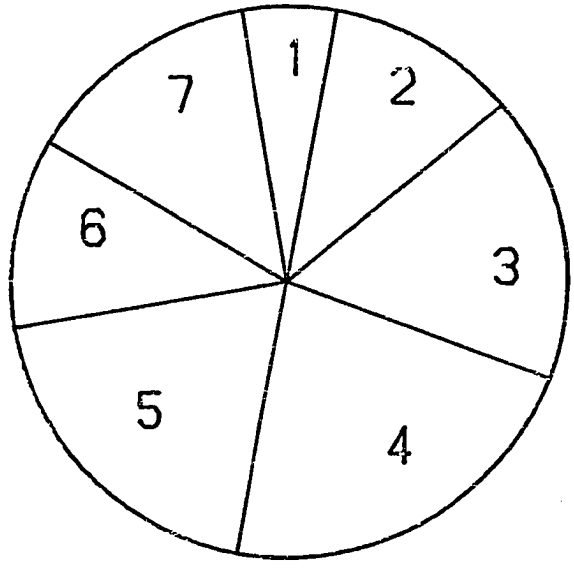


Fig. 5-4. Roulette Wheel Selection

$n_{replace}$ * PROPORTION * n_{class} members of the lowest performance population. These members are compared to the child and the child replaces the most similar candidate on the basis of similarity count, where similarity count is a simple count of the positions where both child and candidate are identical (#=#, 1=1, 0=0). In this way, children replace similar population members and a pressure exists to maintain diversity within the population. This pressure helps counterbalance the occurrence of premature convergence noted earlier.

Restricted Crossover

Previously, our structure was a simple string and we permitted crossover at any crossing site between 1 and the string length l. With a classifier, we may also perform simple crossover if we view the rule as the concatenation of 3 strings, two conditions and the message. For example, with l = 4:

#	#	1	#	#	0	0	#	1	0	1	0
|		C_1		|		C_2		|	M	|

Once again, we could simply permit crossover anywhere along the string; however, to retain greater control over the kinds of offspring we permit, we introduce 3 new parameters, XLO, XHI and GAMODE. The parameters XLO and XHI define the region of permissible crossover. For example, XLO = 1 and XHI = l-1 permit crossover along the entire condition or message, whereas other values would specify some sub-range.

This mechanism permits the allocation of protected string regions which may not be modified by crossover.

The parameter GAMODE specifies which elements of the classifier engage in the crossing as follows:

GAMODE	Elements
0	C_1, C_2, M
1	C_1 only
2	C_2 only
3	M only
4	C_1, C_2
5	C_1, M
6	C_2, M

By this mechanism, regions of crossing may be further restricted, giving more control over the learning process.

We note that with the GAMODE parameter, we still only permit crossing of similar elements: C_1-C_1, C_2-C_2, M-M. Holland [63,64] has suggested extensions to this crossover, where messages may be crossed with conditions thereby promoting an increased probability of linkage between rules. These extensions are important if we expect useful chains of rules to form in a timely fashion. In this study, however, we keep the system as simple as possible to investigate the capabilities of the basic mechanism.

Ternary Mutation

Previously, with the binary alphabet, mutation was simply a bit inversion with probability, $p_{mutation}$. The

message portion of the classifier also uses a binary alphabet and therefore, message mutation is interpreted as before. With the conditions, we have the ternary alphabet V+ and we extend the mutation operator so that with probability $p_{mutation}$ we change characters. The changed character is selected with probability 0.5 from the two remaining characters.

5.5 Application to a Simple Control Problem

In this section, we apply the learning classifier system to the control of a pure, inertial object in a frictionless, one-dimensional domain. The system is pictured in Figure 5-5, and the plant dynamics are given by Newton's second law:

$$m \frac{d^2x}{dt^2} = f(t)$$

where:
- m - mass
- f - force
- x - distance
- t - time

The objective is to center the object given a decision space of two alternatives: the LCS may apply a force of given magnitude, F_{mag}, in the positive or negative directions. We assume complete state knowledge; however, this information is limited by the discretization we select.

The problem seems simple, perhaps too simple to be of much use in evaluating the LCS method. Nonetheless, the simpler problem is interesting for two reasons. First, time optimal control of an inertial object is a fundamental problem in optimal control theory. Its bang-bang solution

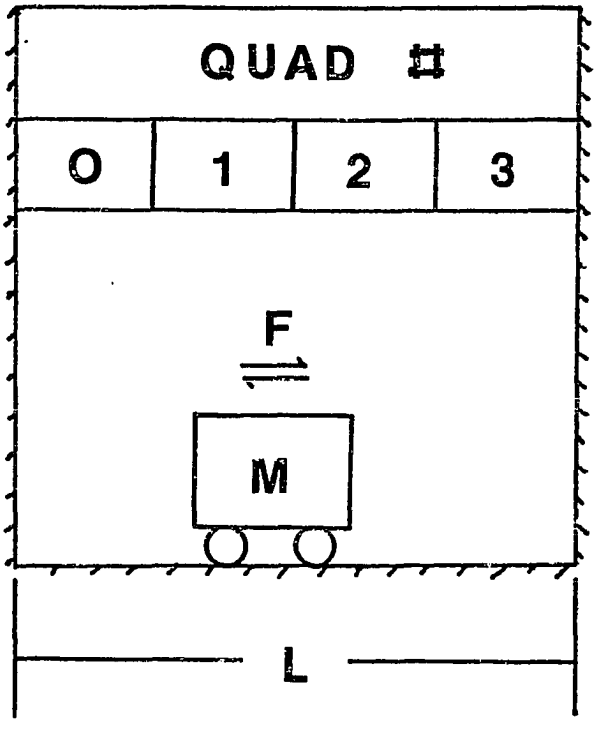


Fig. 5-5. Inertial Object Domain - Schematic

is well known [77] and thus, we connect the LCS work with extant control literature. Second, the control of an inertial object is a problem people and other mobile creatures solve everyday: whether driving a car, riding a bicycle or moving our limbs through space, we repeatedly face and solve this problem effortlessly. If the LCS is to be useful, it too must be capable of attacking this recurring model problem.

Problem Specification

Specifically, the LCS is faced with the following domain parameters:

L - wall to wall distance = 50 m

M - object mass = 5 kg

F_{mag} - force magnitude = 2 N

Δt - cycle time = 1 s

The LCS is born with detectors capable of deciphering the following information with the specified discretization within the stated limits:

Measurement	Discretization (in bits)	Low Level	High Level
x - distance	2	0	50
u - velocity	2	-2	2
F_{mag} - force magnitude	1	0	2
F_{sign} - force sign	1	-1	1
P - payoff	1	0	1

With this discretization, the LCS perceives the domain as 4

subregions. Speed is recognized in 4 discrete sub-ranges, 2 forward and 2 reverse, and the other detectors are simple binary switches.

To better see how the LCS sees its domain, we outline the format of the environmental message in Figure 5-6. The message is of length l=8 with distance, velocity, force, and payoff measurements allocated as shown. We note that the tag field is one position long; this is sufficient in this study as 2 classes of messages, external and internal, are all that are required.

With the stated parameters and discretization we may learn some additional information about the domain through some elementary calculation. The wall to wall travel time assuming constant application of F_{mag} may be calculated as follows:

$$\begin{aligned} t_{\text{wall to wall}} &= \text{sqrt}(2ML/F_{mag}) \\ &= \text{sqrt}(\ 2(50)5/2\ ) \\ &= 15.8\ \text{s} \end{aligned}$$

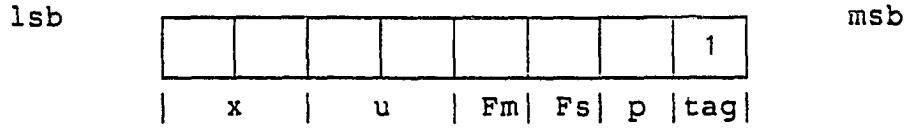
Under these conditions the terminal velocity before striking the opposing wall is given :

$$\begin{aligned} V_{max} &= F_{mag} * t_{wtw}/M \\ &= 6.3\ \text{m/s} \end{aligned}$$

This value is the maximum obtainable with inelastic walls. As a result, with the specified sensitivity, the velocity measurement may saturate occasionally.

Another parameter of interest is the minimal centering time T_{cmin}. Assuming the object is on the wall and

FIGURE 5-6
Environmental Message and Coding



x		u	
Sub-message	Meaning	Sub-message	Meaning
00	quad 0 (leftmost)	00	u<-umax/2
10	quad 1	10	-umax≤u<0
01	quad 2	01	0≤u<umax/2
11	quad 3 (rightmost)	11	umax≤u

F_{mag}		F_{sign}	
Sub-message	Meaning	Sub-message	Meaning
0	no force	0	negative force
1	force=max	1	positive force

P-payoff		Tag	
Sub-message	Meaning	Sub-message	Meaning
0	no payoff	0	internal message
1	payoff	1	environmental message

stationary, the time optimal control is to apply the maximal restoring force until the object is halfway to the desired location and then apply the maximal force in the opposite direction until the objective position is achieved and the object is once again stationary. For our problem, this results in a minimal centering time as follows:

$$T_{cmin} = 2 * \text{sqrt}(2F_{mag}(L/4)/M) = 15.8 \text{ s}$$

We should hope that the LCS is capable of positioning the object from any location in a time on the order of this value after sufficient learning has taken place. These results will be useful when we look at the results from actual learning simulations.

Reward Mechanism

As described earlier, rules which activate rewarded effectors, receive a point score as a payment. In natural domains, reward is usually in the form of food or other substances necessary for survival. In our artificial domain, the reward may come from one of two sources: the keyboard (a knowledgeable instructor) or a computer subroutine which rewards the system according to some systematic method. In an actual installation, the use of an expert instructor may be the preferred mode of instruction; however, in the interest of uniformity and repeatability, a systematic computer procedure is adopted for this study.

For the inertial object environment, the reward procedure works according to the following scheme:

```
if (T mod T_eval = 0) (* every T_eval th time step *)
 and ( not accelerating away from the target )
 and ( (Near the walls and applying restoring force)
      or (Near the target and (going slow or
      decelerating))
 then points=MAXPOINTS
  else points=0
```

The conditions, near the wall and going slow, are made rigorous by introducing the input parameters x_{tol} and u_{slow} respectively.

Implementation

A computer procedure encompassing the LCS described in this chapter as well as the inertial object environment, reward mechanism, input-output routines, and necessary interface procedures has been implemented in Pascal and 6502 assembler for execution on an Apple II computer. Skeletal pseudo-code descriptions outlining the program's modular structure are presented in Appendix B. The code is written primarily in Pascal to enhance portability. The portions of code written in assembler may be replaced by equivalent Pascal code; however, the use of machine code speeds up the fundamental rule matching process by a factor of between 3 and 10.

Setting LCS Parameters

In both the apportionment of credit system and the genetic algorithm, a number of parameters must be selected before proceeding. Here, we examine the method of choosing these parameters, relying on both limited simulations and past experience with GA optimization.

To select the apportionment of credit parameters, a series of limited simulations have been undertaken with performance compared over a fixed interval. There are three parameters of interest: bidding coefficient C_{bid}, taxation coefficient C_{tax} and the bid spread σ_{bid}. Additionally, these coefficients must be considered in relationship to the regularity of reward from the environment. In our system, this may be controlled by adjusting the period of evaluation T_{eval}.

In the restricted simulations, a known set of good rules is specified with some bad rules implanted to test the ability to weed out the bad and elevate the good. Table 5-2 shows the rule set, individual rule meanings in shorthand, and a brief explanation of each rule's purpose. For this set of experiments, the apportionment of credit algorithm is enabled while the genetic algorithm has been turned off.

In the first set of experiments, the parameter C_{bid} is varied over a range of values while σ_{bid} and C_{tax} are set to zero: there is deterministic bidding and no taxation. The population is started from a relatively high value of strength, and the calculations proceed. In this manner, unfit rules, continuously lose strength while the good rule set adapts to its appropriate steady state value.

The results from 3 different values of C_{bid} are shown in Figure 5-7, a graph of total evaluation, TOTALEVAL, vs. time. With the largest value, apparently too much is wagered too soon and the performance lags that of the lower

Table 5-2

Parameter Adjustment Tests - Specified Rule Set

Rule	Shorthand Description	Rule Purpose
#1#####1:#######1/00001000	(x=R) -> f-	pure restoration
#1#####1:#######1/00001100	(x=R) -> f+	bad rule (the monkey wrench)
#0#####1:#######1/00001100	(x=L) -> f+	pure restoration
1011###1:#######1/00001000	(x=1,u=2) -> f-	pure brake
0100###1:#######1/00001100	(x=2,u=-2) -> f+	pure brake

Shorthand Key: x values = 0 - quad 0
1 - quad 1
2 - quad 2
3 - quad 3
Θ - odd quads (1,3)
E - even quads (0,2)
L - left quads (0,1)
R - right quads (2,3)

u values = -2 - fast negative
-1 - slow negative
1 - slow positive
2 - fast positive
+ - positive (1,2)
- - negative (-1,-2)
Θ - odd speed (-1,2)
E - even speed (-2,1)

f values = + - positive force
- - negative force

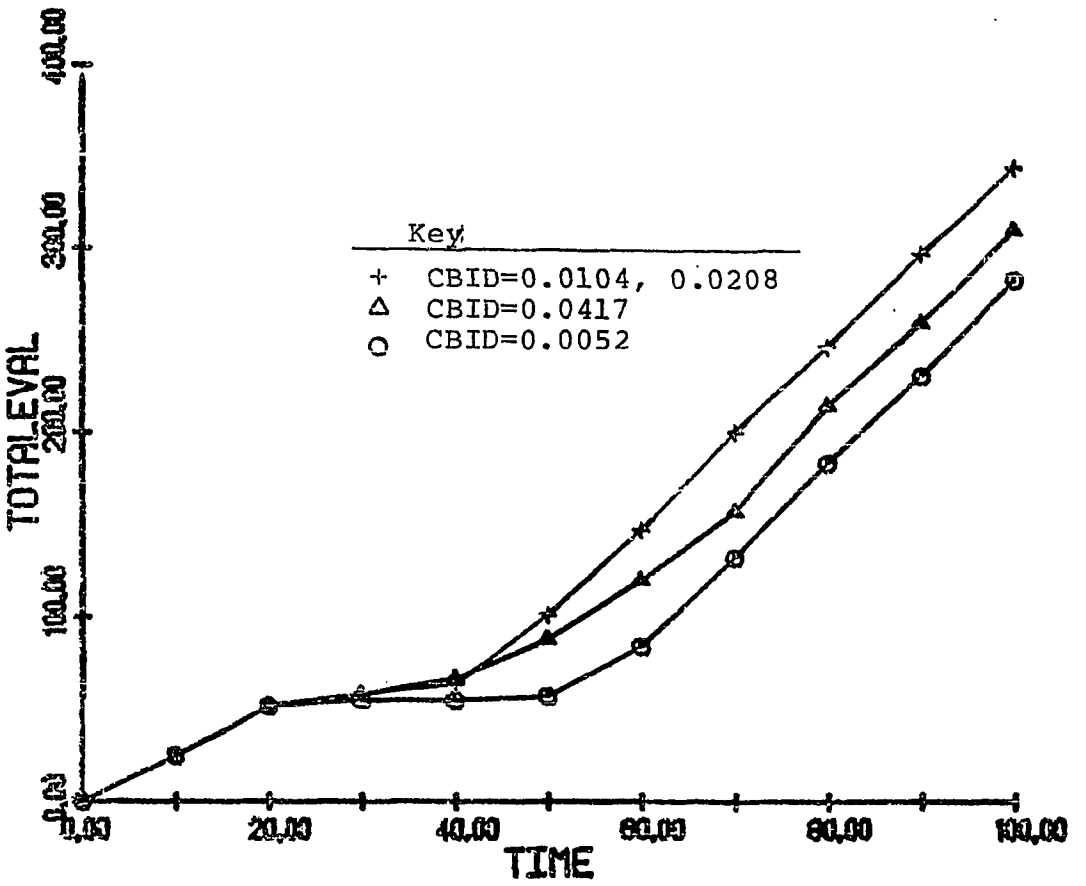


Fig. 5-7. Variation of CBID - TOTALEVAL vs. Time

values.

To gain more perspective on the effect of C_{bid} we need to look at the interaction with other system parameters. Holding C_{bid} constant we vary the interval of evaluation, T_{eval}, over a range between 1 and 5 time steps. The total evaluation is plotted versus time in Figure 5-8. For the values 1, 3, and 5 the results are as we might expect; as the interval of evaluation goes up, the rate of evaluation goes down roughly proportionately. In the cases with $T_{eval} = 2, 4$, something strange occurs. The performance is much worse than we expect; hardly any reward is achieved in either case. Close examination of the run details shows the reason. There is a continuous oscillation between a good rule and a bad rule. The oscillation causes the bad rule to be active during reward time steps and the good rule to be active on time steps without reward. This continues until the object enters a zone where the bad rule's action may receive reward as a braking rule. This reward is then sufficient for the bad rule to drive the object to the wall where the process repeats itself. The problem here is caused by our deterministic bidding process ($\sigma_{bid}=0$) in concert with the rhythmic reward pattern. As the good and bad rules are activated alternatively, each must pay its bid, reducing its strength some small amount. This makes its next bid less than its competitor by some small amount thereby continuing the oscillation. Thus, we encounter our first empirical need for noisy bidding to eliminate bias in

the decision process. In this case, some small randomness in bidding should destroy the certainty of the oscillation and decrease the impact of small differences among competing bids.

We investigate this by holding C_{bid} = 0.0208 and T_{eval} = 2 while varying σ_{bid} the amount of stochastic bid spread over the values, 0.001, 0.01, 0.1, 1.0 times MAXPOINTS. These results are summarized in Figure 5-9 showing the values at time = 150 vs. σ_{bid}/MAXPOINTS. The introduction of even the smallest amount of randomness, results in a drastic improvement over the deterministic decision process; the certainty of the oscillation is destroyed as we predicted.

Yet, we should not assume that this small a level will encompass all possible cases. Remember, we will be generating new rules for insertion into the population at the average strength of their parents. As a result, if these rules are ever to have a possibility of trial, the σ_{bid} parameter must be sized to lift them above the leaders upon occasion. To examine this possibility, a test is set up where two good rules are placed at artificially low levels of strength and σ_{bid} is varied to investigate the value required to raise the rules from their initially low values during bidding competition. The results of this test are shown in Figure 5-10, a graph of the average strength of the 2 reduced strength rules at time = 300 time steps vs. σ_{bid}/maxpoints. Below 15%, the randomness is

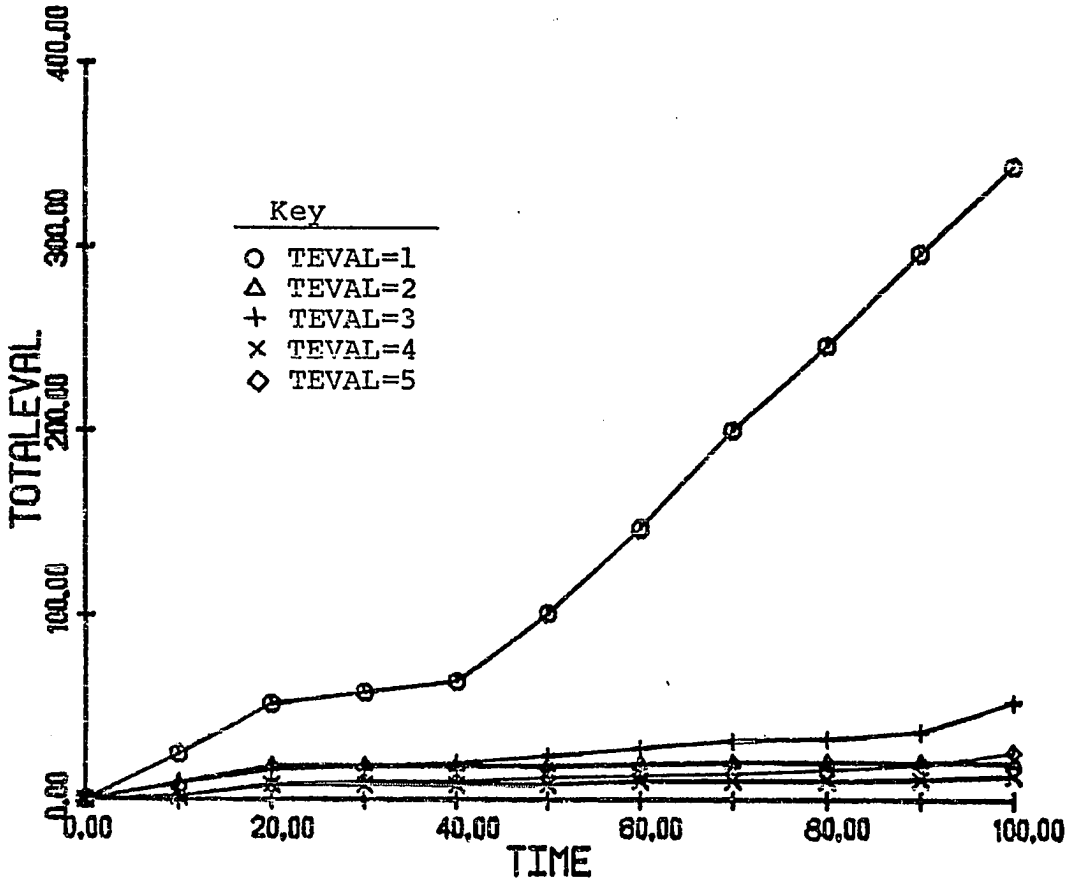


Fig. 5-8. Variation of TEVAL - TOTALEVAL vs. Time

insufficient to give the two rules the needed boost as the strength is simply the free fall value at the tax rate of 0.002. Above 15%, the mechanism gives sufficient assistance to start to raise the rules from their artificially low values.

With C_{bid} and σ_{bid} selected, our attention turns to the taxation coefficient, C_{tax}. In setting the tax rate, the main consideration is the free fall half life. Since the apportionment of credit algorithm simply reduces an inactive rule by the tax rate, after n iterations of inactivity we have the following value of strength:

$$S(t+n) = S(t)(1-C_{tax})^n$$

Thus, the half-life may be given:

$$n = \log(1/2) / \log(1-C_{tax})$$

The half-life is tabulated below for convenient values of C_{tax}:

C_{tax}	Half Life
0.1	6.6
0.01	69.0
0.005	138.3
0.004	172.9
0.003	230.7
0.002	346.2
0.001	692.8

Since new rules are regularly inserted into the population at possibly elevated levels of strength (average of parents'), the tax rate must be set to insure that inactive rules are degraded sufficiently before reproducing and inserting new rules. If this is not done, relatively inactive rules can retain an unrealistically high level of

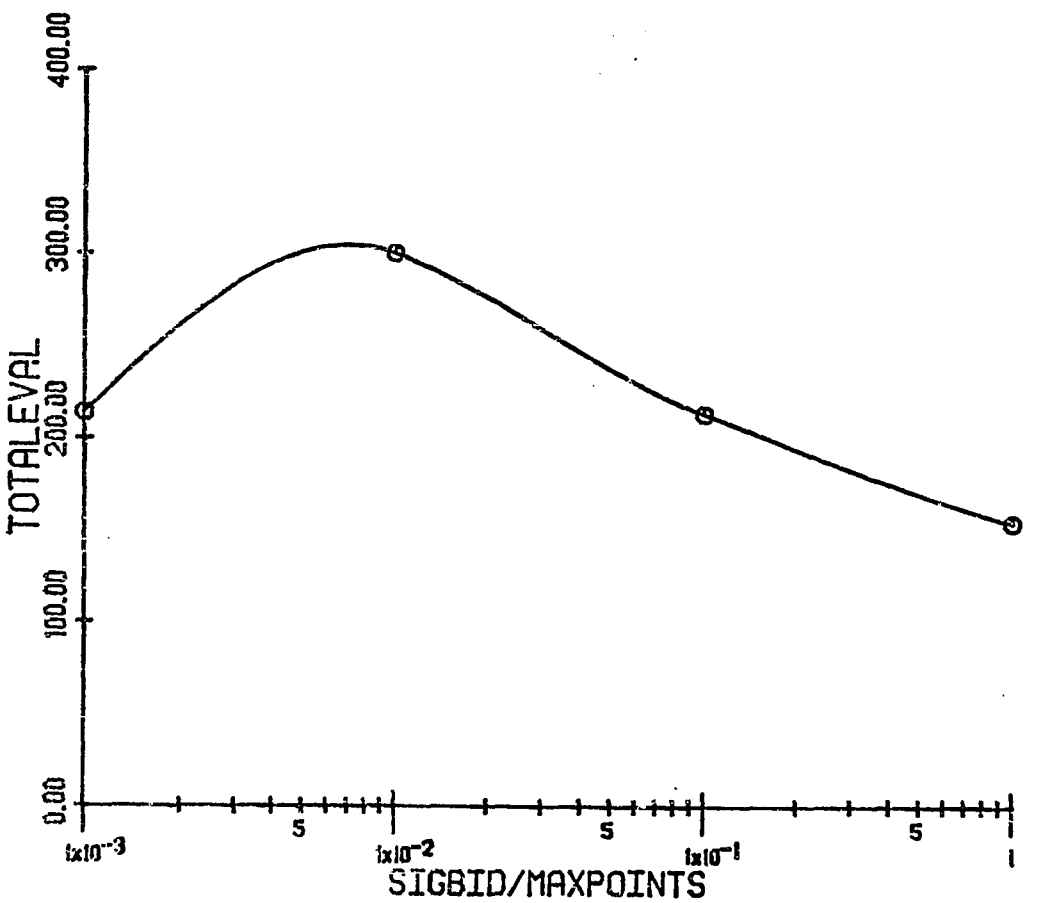


Fig. 5-9. Variation of SIGMABID - TOTALEVAL vs. SIGMABID/MAXPOINTS

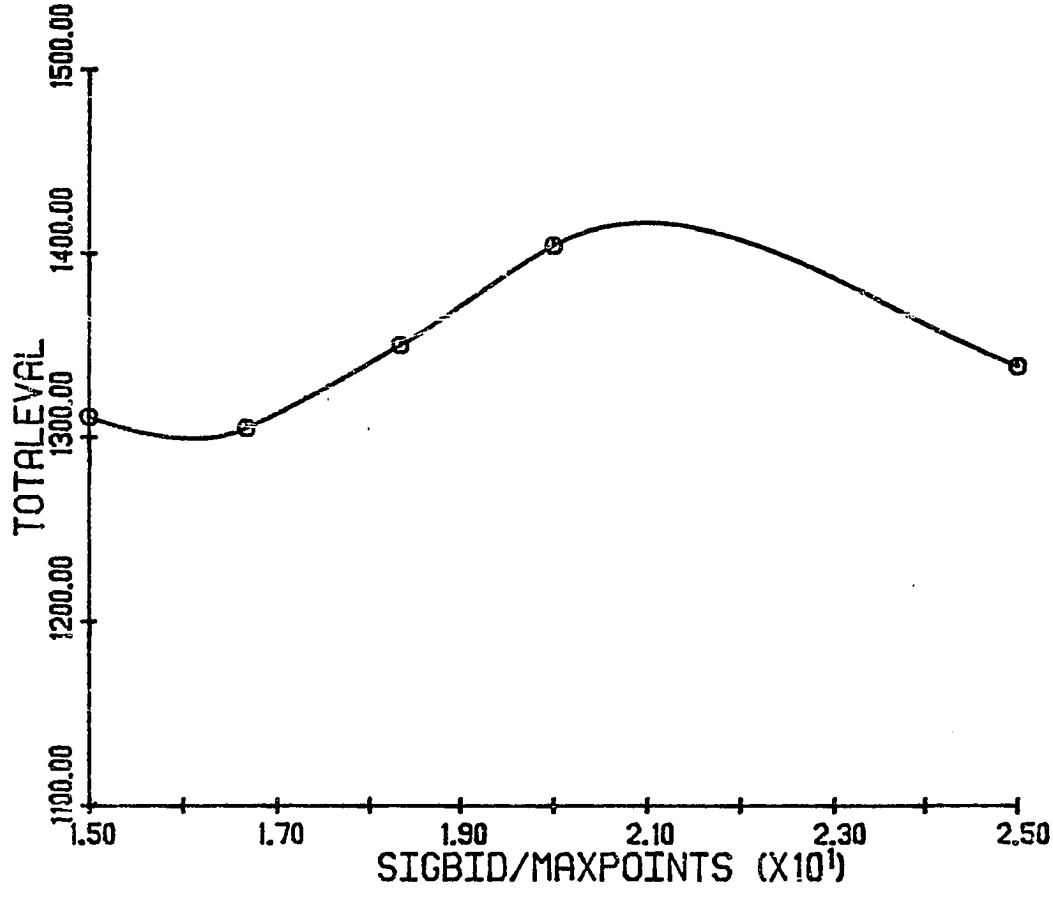


Fig. 5-10. Variation of SIGMABID - Low Initial Strength Average TOTALEVAL (2 rules) vs. SIGMABID/MAXPOINTS

strength and ultimately reach reproduction themselves, thereby cluttering future rule sets with large numbers of overrated, inactive rules. Therefore, C_{tax} must be set to yield a half-life on the order of the insertion period of the genetic algorithm, T_{ga}.

In these initial runs, an insertion period of $T_{ga} = 200$ has been set. This period is long enough to encompass a meaningful span of system behavior. It is an order of magnitude greater than the minimum centering time, T_{cmin}; it permits meaningful evaluation of new and extant rules. In early tests, a C_{tax} value of 0.001 was used (Half Life=693) In conjunction with the GA period this resulted in large numbers of over-evaluated rules. In subsequent tests, a value of $C_{tax} = 0.002$ (Half Life=346) eliminated this difficulty by reducing the half life sufficiently without detrimental deterioration of memory.

With C_{tax} and T_{ga} set, the other GA parameters have been selected based on a combination of past experience and current expectations. PROPORTION, the proportion of a population selected for replacement at a given GA invocation, has been varied to obtain a small number of new trials (usually between 2 and 4) per GA activation. Contrasting the current learning tests to the optimization studies in a previous chapter, this is a necessary step if we want to maintain current performance at high levels while exploring new rules.

In this study, we select a value of $n_{replace} = 3$, the

number of low strength rules in each replacement subpopulation. This value is chosen arbitrarily although it is consistent with De Jong's findings [53] in an optimization setting.

Results

With the LCS implemented and parameters selected, we perform some learning tests. In these tests, we start with a random selection of rules and proceed forward monitoring performance by summing the total reward, TOTALEVAL; we also examine an important auxiliary performance characteristic related to goal achievement. In these tests, our ultimate goal is to locate the object at some specified location (the center). The total point score reflects the achievement of this goal; however, it is easier to monitor if we keep track of the number of times it is achieved. Since, it is unreasonable to expect the goal to be achieved exactly, we create a more workable definition. For this study, the centering criterion is achieved when the object is within the target zone $|x - x_{target}| < x_{tol}$ and is going slow, $|u| < u_{slow}$ for $n_{criterion}$ consecutive time steps. The values used for these parameters are as follows:

$$n_{criterion} = 10$$
$$u_{slow} = 1.5$$
$$x_{tol} = 12.5$$

Upon reaching the goal, the goal statistic, TOTALGOAL, is incremented by one and the object is disturbed by a force of $\pm F_{disturbance}$ for one time step. We use a value of

$F_{disturbance} = 50$. In this way, the LCS is repeatedly forced to perform over a broad range of space and velocity values thereby providing a fairer test of the procedure's capability.

To test the system, two series of tests have been performed. These tests examine the LCS performance both, with and without genetic algorithm under both undisturbed and rule-deprived conditions.

In the first series of tests, we start the LCS from a randomly generated population of 30 rules of the forms:

RRRR#R#1:#######1/00001100 (force +)

RRRR#R#1:#######1/00001000 (force -)

Half of the population is of the first type and half is of the second type. In this notation, the colon separates conditions, the slash separates conditions from the message, the 0, 1, and # characters have their normal meaning, while the R designates a position to be selected using random choice. To permit control over the level of generality (number of #s), we introduce the parameter $p_{generality}$; a don't care symbol, #, is selected with probability $p_{generality}$ while a 0 or a 1 is selected with probability $(1-p_{generality})/2$. We fix $p_{generality} = 0.75$ giving a fairly general initial rule set.

Table 5-3 shows the initial population generated for the first series of cases. In the first run, number IOLCS.1, we start from this population and proceed with rule and message system and apportionment of credit system

enabled and the genetic algorithm disabled. In run IOLCS.2, all subsystems, including the genetic algorithm, are enabled. The parameters are the same for both runs as follows:

$$C_{bid} = 0.0208$$
$$\sigma_{bid} = 1.0$$
$$C_{tax} = 0.002$$
$$T_{ga} = 200$$
$$P_{mutation} = 0.0$$
$$\text{PROPORTION} = 0.0667$$
$$T_{eval} = 1$$
$$\text{MAXPOINTS} = 6$$

In Figure 5-11, we compare the results with and without genetic algorithm to random performance on the basis of time-averaged accumulated evaluation, TOTALEVAL/T. In the random results, the decision to direct the force right or left is determined by the flip of a fair coin (p=0.5). We note that both the LCS runs are much better than random performance. Furthermore, case IOLCS.2 (with GA) eventually overtakes and outperforms run IOLCS.2 (without GA). In fact, while the differences appear small on this basis, the difference in physical control is much better in the case with genetic algorithm.

To see this, we shift the basis of comparison to the more sensitive measure, time-averaged number of criterion achievements, displayed as Figure 5-12. Again, LCS performance is far better than random. Performance with the

Table 5-3.

Initial Rule Population
Learning Tests IOLCS.1 and IOLCS.2

<CONDITION$_1$>:<CONDITION$_2$>/<MESSAGE>
###0###1:#######1/00001000
0######1:#######1/00001000
010##1#1:#######1/00001000
1#00#1#1:#######1/00001000
#0#####1:#######1/00001000
##00###1:#######1/00001000
01#####1:#######1/00001000
##10#0#1:#######1/00001000
#######1:#######1/00001000
#0#0###1:#######1/00001000
0##1###1:#######1/00001000
1######1:#######1/00001000
#######1:#######1/00001000
#######1:#######1/00001000
#1#####1:#######1/00001000
##11###1:#######1/00001100
#1#0###1:#######1/00001100
#######1:#######1/00001100
01#####1:#######1/00001100
##1##0#1:#######1/00001100
##0####1:#######1/00001100
0######1:#######1/00001100
#####1#1:#######1/00001100
0##1###1:#######1/00001100
#101###1:#######1/00001100
#1#####1:#######1/00001100
#######1:#######1/00001100
000####1:#######1/00001100
#0#####1:#######1/00001100
##00###1:#######1/00001100

genetic algorithm is that much better than without. In fact, by the end of the learning run IOLCS.2 with GA, the LCS had learned rules sufficient for the regular restoration of the object to the goal position.

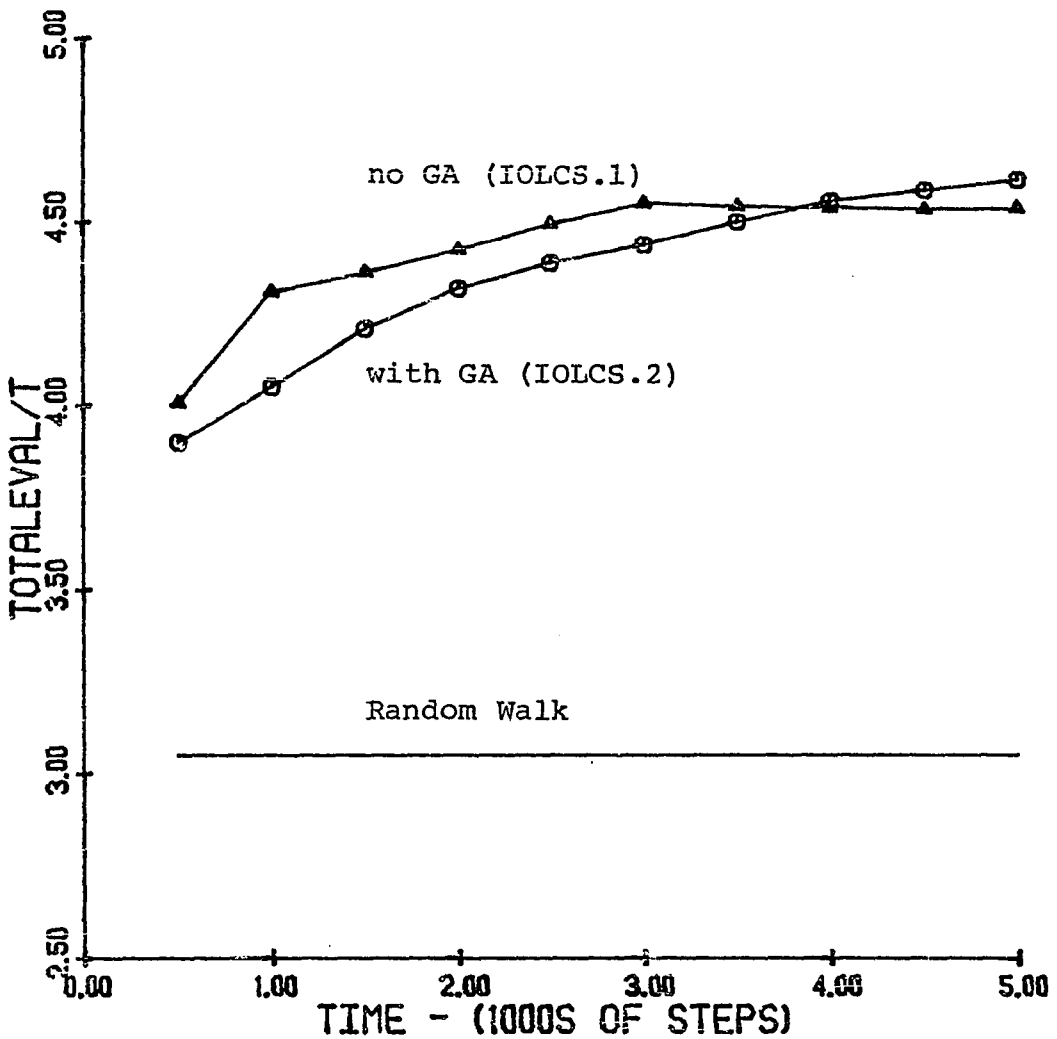


Fig. 5-11. Time-averaged TOTALEVAL vs. Time - Random Rule Set - Runs IOLCS.1 and IOLCS.2

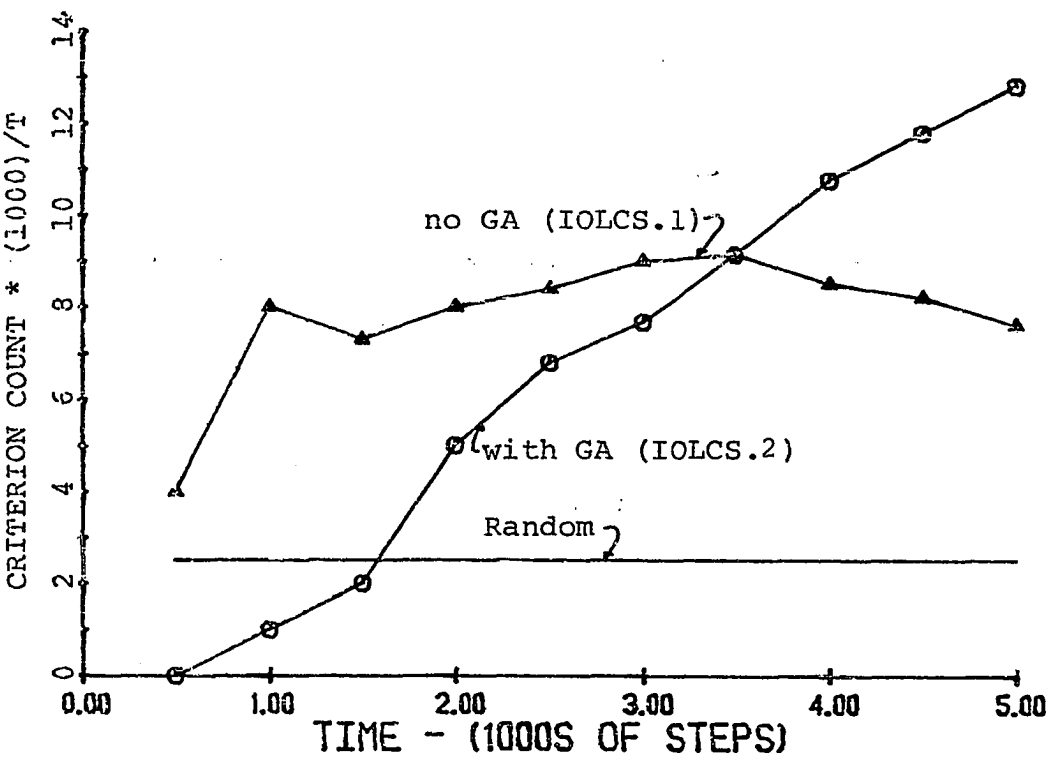


Fig. 5-12. Time-averaged Goal Count vs. Time
Random Rule Set - Runs IOLCS.1 and IOLCS.2

To get a better feel for the type of rules selected and formed by the two learning mechanisms, we examine the above average rule sets generated by each of the runs. Earlier, we introduced the time-optimal solution where we recognized the need for two types of action: restoring action and braking action. Roughly speaking, we accelerate toward the target with maximum force and suddenly apply the maximum force in the opposite direction to place the object at the desired location. Both types of actions are necessary for effective control in a frictionless system. Similarly, in a rule-based system, we expect to see 2 types of rules, restoration and braking. Previously, in our tuning tests, the specified rule set consisted of 4 rules, two restoring and two braking rules as follows:

```
if <x=L> then <F+>                          :  Restoration
if <x=R> then <F->

if <x=1> & <u=+> then <F->                  :  Braking
if <x=2> & <u=-> then <F+>
```

The restoring rules are fairly general. If the object is in the left or right half plane a restoring force is applied. The braking rules are more specific, only applying a braking force when the object is adjacent to the goal location and moving quickly toward the goal. Removal of any of these rules is detrimental to performance. Restoration without braking results in a perpetual oscillation. Braking without restoration is ineffective because no work is directed toward the desired goal.

In the two trial runs, we see evidence, as we must, of

both types of rules. In Table 5-4 we list the above average rules of run IOLCS.1 (no GA) at the end of the run (T=10000). Some of the more clear cut rules are interpreted in the table. In fact, we see some of the rules designed (by an intellect superior to the LCS) for the tuning tests, although the population was selected initially at random with no implants of helpful seeds. Clearly, the population has sufficient restoration rules, as we might expect, because the model restoration rules require the setting of a single bit. The braking rules are not quite as plentiful, nor are the ones selected as effective as we desire.

Contrasting these results to the above average rules of run IOLCS.2 (with GA), we see that by exploring new rules we can obtain more efficient braking as evidenced by the higher scores and the final rule set shown in Table 5-4. In this rule set, the presence of better braking rules is clearly responsible for better performance. This is precisely the kind of learning we had hoped to achieve. The apportionment of credit mechanism rates extant rules and decides among competitors, while the genetic algorithm contributes new rules to the fray.

Deprivation Cases

Further evidence of this desirable type of learning behavior is evidenced in runs IOLCS.3 and IOLCS.4. In these cases, we make life even more difficult for the LCS by starting from an otherwise randomly generated population with the best rules removed. As before, we generate a

Table 5-4

Learning Tests IOLCS.1 and IOLCS.2
Above Average Rule Sets at T=5000

Rule	S(5000)	Shorthand Description	Rule Purpose
IOLCS.1 (no GA)			
1) #1#####1:#######1/00001000	54.5	(x=R) -> f-	pure restoration
2) #0#####1:#######1/00001100	30.3	(x=L) -> f+	pure restoration
3) ##00###1:#######1/00001100	26.2	(u=-2) -> f+	Hi speed brake
4) ##0####1:#######1/00001100	8.1	(u=E) -> f+	mixed rule
5) #######1:#######1/00001000	6.8	(#) -> f-	default action
6) 0######1:#######1/00001000	5.5	(x=E) -> f-	mixed rule
7) 01#####1:#######1/00001000	4.6	(x=2) -> f-	partial restoration
Population Average	4.4		
IOLCS.2 (with GA)			
1) #1#1##1:#######1/00001000	46.3	(x=R,u=+) -> f-	direction specific restoration
2) #0#0##1:#######1/00001100	43.0	(x=L,u=-) -> f+	direction specific restoration
3) #1#####1:#######1/00001000	41.7	(x=R) -> f-	pure restoration
4) #######1:#######1/00001100	40.9	(#) -> f+	default action
5) ##0####1:#######1/00001100	39.6	(u=E) -> f+	mixed rule
6) 1######1:#######1/00001000	35.5	(x=0) -> f-	brake and restore
7) #0#####1:#######1/00001100	20.2	(x=L) -> f+	pure restoration
8) #0#0##1:#######1/00001100	19.4	(x=L,u=-) -> f+	#2 rule backup
9) #0#0##1:#######1/00001000	17.0	(x=L,u=-) -> f-	#2 rule opposite
10) ###0###1:#######1/00001100	15.4	(u=-) -> f+	speed brake and restore
Population Average	13.5		

population at random using the same rule templates, this time with $n_{class} = 40$. We permit the system to select its best rules by running the LCS with AOC on and GA off. At the end of this initial trial, we remove the 10 best rules and restore the remaining deprived population of $n_{class} = 30$ to the normal initial strength value. This deprived population is used as the starting point for the two learning deprivation runs, IOLCS.3 (no GA) and IOLCS.4 (w/ GA). The deprived initial population is displayed as Table 5-5.

Once again we compare performance on the basis of time-averaged accumulated evaluation, TOTALEVAL/T and contrast this to the performance of the random walk. These results are shown in Figure 5-13. As before, the case without GA outperforms the random walk and is outperformed by the case with genetic action. More dramatically, we see the physical performance as measured by the time-averaged criterion count in Figure 5-14. Here the effect of deprivation is most striking. Without genetic action, the deprived rule set never achieves criterion. With genetic action the rule set quickly outperforms the other cases moving toward the results of the previous simulations without deprivation.

As before, a comparison of the above average rules is instructive. Looking at Table 5-6, without the GA, we clearly see the reason for not achieving criterion: there are no effective braking rules. In the run IOLCS.4 the genetic algorithm has discovered some effective brakes

Table 5-5

Initial Deprived Rule Set
Runs IOLCS.3 and IOLCS.4

<CONDITION$_1$>:<CONDITION$_2$>/<MESSAGE>
###0###1:#######1/00001000
#####1#1:#######1/00001000
0#00#0#1:#######1/00001000
#00####1:#######1/00001000
1#0##1#1:#######1/00001000
0#1###1:#######1/00001000
##10#0#1:#######1/00001000
#1#0###1:#######1/00001000
##0####1:#######1/00001000
000##0#1:#######1/00001000
#1#####1:#######1/00001000
#0#0#0#1:#######1/00001000
##10###1:#######1/00001000
#1#0###1:#######1/00001000
#0###1#1:#######1/00001000
#######1:#######1/00001100
#1###0#1:#######1/00001100
#1#####1:#######1/00001100
#####0#1:#######1/00001100
1####1#1:#######1/00001100
#0#####1:#######1/00001100
#1#1###1:#######1/00001100
01#####1:#######1/00001100
0##1#1#1:#######1/00001100
###1###1:#######1/00001100
#####0#1:#######1/00001100
##1####1:#######1/00001100
11#####1:#######1/00001100
1#1##1#1:#######1/00001100
#1#1#1#1:#######1/00001100

thereby permitting consistent criterion achievement.

5.6 Summary

In this chapter, we have explored the history,

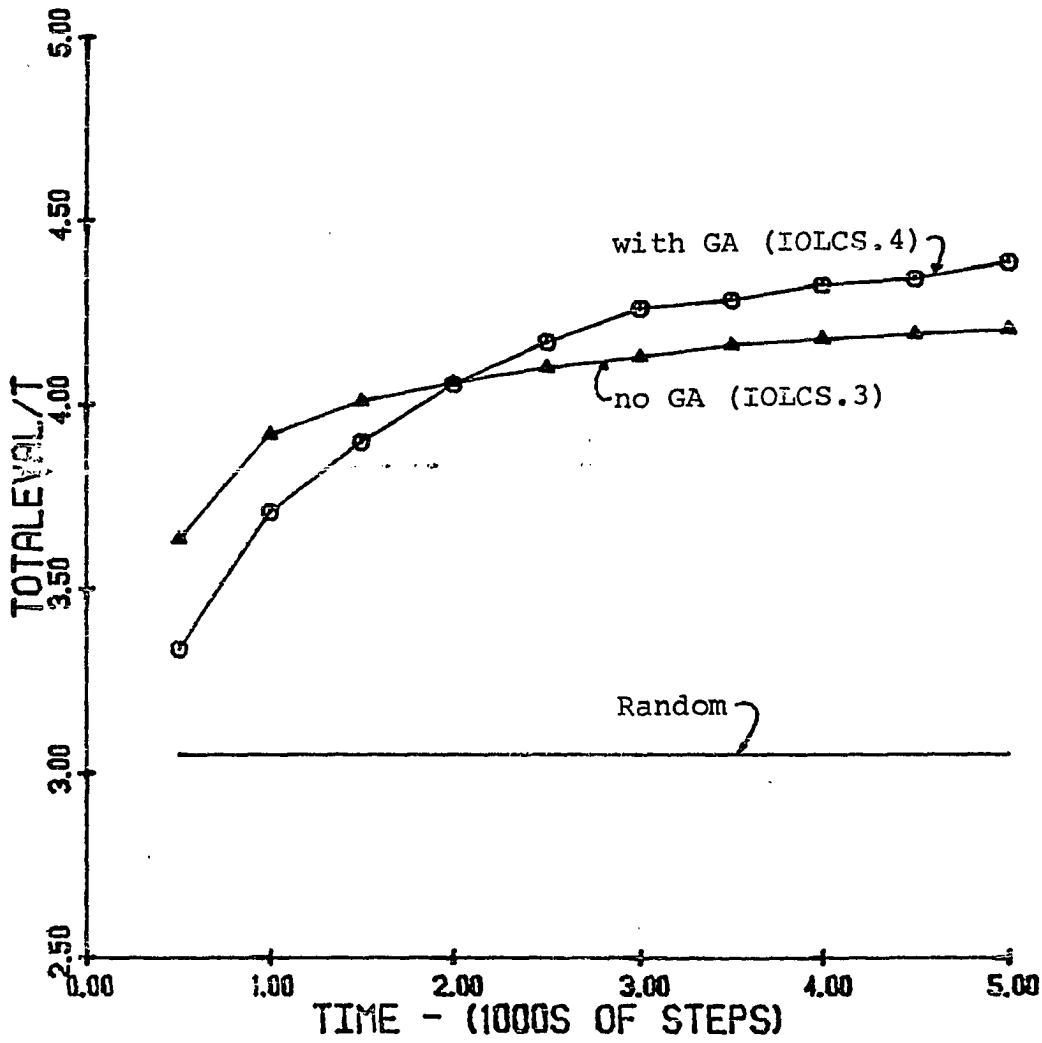


Fig. 5-13. Time-averaged TOTALEVAL vs. Time - Deprived Rule Set - Runs IOLCS.3 and IOLCS.4

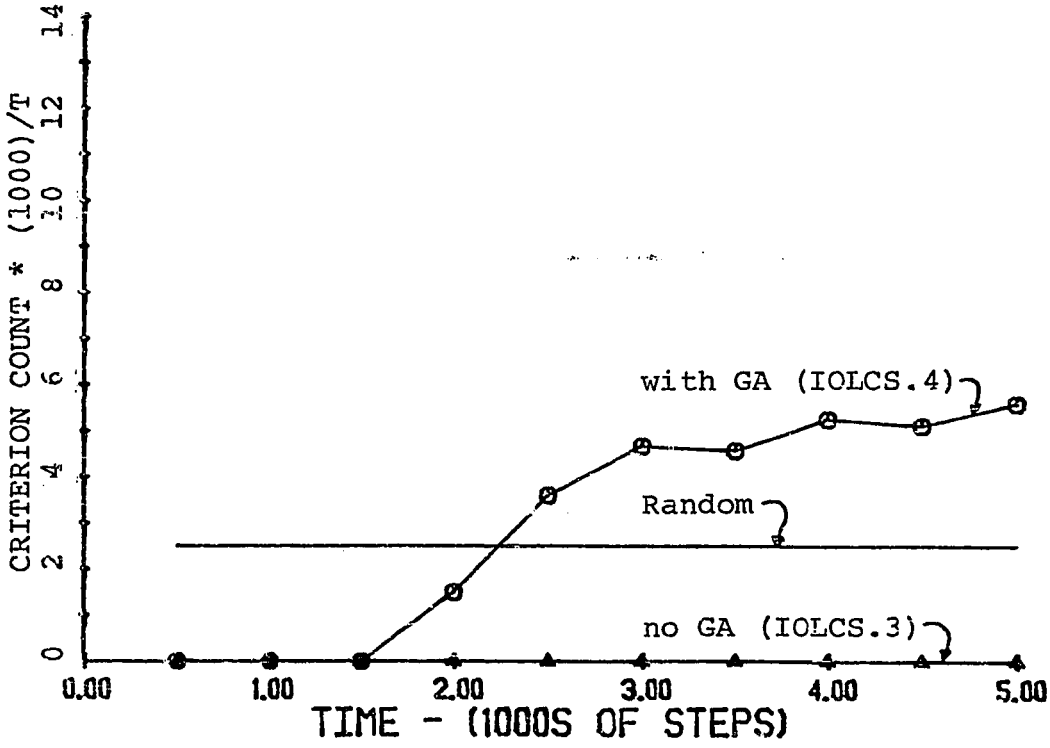


Fig. 5-14. Time-averaged Goal Count vs. Time Deprived Rule Set - Runs IOLCS.3 and IOLCS.4

Table 5-6

Above Average Rule Sets at T=5000
Deprivation Runs IOLCS.3 & IOLCS.4

Rule	S(5000)	Shorthand Description	Rule Purpose
IOLCS.3 (no GA)			
1) #1#####1:#######1/00001000	40.8	(x=R) -> f-	pure restoration
2) #0#####1:#######1/00001100	33.3	(x=L) -> f+	pure restoration
3) #######1:#######1/00001100	21.1	(#) -> f+	default rule
4) #####0#1:#######1/00001100	7.6	(f=-) -> f+	counterbalance
5) 0##1#1#1:#######1/00001100	5.0	(x=E,u=+,f=+) -> f+	restore and ???
Population Average	3.8		
IOLCS.4 (with GA)			
1) ###1###1:#######1/00001000	53.4	(u=+) -> f-	brake and restore
2) #0#####1:#######1/00001100	48.6	(x=L) -> f+	pure restoration
3) #0#####1:#######1/00001100	43.8	(x=L) -> f+	rule #2 backup
4) #0#####1:#######1/00001100	41.6	(x=L) -> f+	rule #2 backup
5) 01#1###1:#######1/00001000	36.4	(x=2,u=+) -> f-	direction specific restoration
6) 01#####1:#######1/00001100	29.5	(x=2) -> f+	weak brake
7) ##10#0#1:#######1/00001000	25.9	(u=-1,f=-) -> f-	directional restoration
8) #0#####1:#######1/00001000	21.8	(x=L) -> f-	weak brake
9) #0#####1:#######1/00001100	17.5	(x=L) -> f+	rule #2 backup
Population Average	13.6		

principles of operation, and application of a learning classifier system (LCS).

Learning classifier systems are the product of Holland's continuing work on adaptive systems. A variety of researchers have applied and enhanced these ideas; however, the application in engineering-related domains has been limited.

We have seen how the LCS starts from the concept of a rule and message system, a type of production or rule-based system. Production systems are useful in operations domains because human operators seem to store their knowledge in rule-of-thumb form. As with other production systems, the rules are of the form, if <conditions> then <action>; however, in the LCS, conditions and messages are restricted to a fixed length string. Explicit pattern recognition is provided by extending the binary alphabet by a single, don't care symbol, #. Though simple in form, this system is both computationally complete and convenient.

Classifiers (rules) send messages which may be placed on the message list, potentially activating other rules or directing external action by setting an action trigger, an effector. The presence of a central message list is a distinguishing and important feature providing a universal communication channel much like a bulletin board.

Since space is limited on the message list, some method must exist for choosing among competing messages. An apportionment of credit algorithm modeled after a

competitive service economy insures that rules are properly evaluated and selected. Rules bid for the right to send their messages; winning bids are paid to classifiers which previously sent activating messages. In this way, a chain of middlemen forms from the environment to ultimate action. Competition keeps the system honest; useful classifiers live and prosper, while the unsuccessful lose the means to engage in commerce.

The payment made to and from a rule increases and decreases its net worth called its Strength. Strength is used to help determine a rule's bid; it also serves as a rule's fitness in a genetic algorithm search for new rules. The GA adopted is similar to the one described earlier in the optimization chapters; however, we only reproduce a portion of the rule population at any one GA invocation. Differences in the reproduction and mutation methods are relatively minor and have been discussed in this chapter.

An LCS has been implemented and interfaced to a simulated environment: an inertial object traveling in a frictionless, one-dimensional space bounded by inelastic walls. The LCS decides whether to apply a force in the positive or negative direction, and it is rewarded if the force tends to center the object and bring it to rest. We choose this test problem to gain experience with the LCS in a simpler environment than the typical pipeline system. Yet, we recognize that this problem is no pussy cat; a variety of rules working together are necessary for high

performance.

In a variety of studies, the LCS has learned effective rule sets for control in this domain. In studies with the genetic algorithm enabled and disabled, the LCS has always outperformed a random walk. Cases with genetic algorithm have consistently outperformed those without; the difference is accentuated if we look at the number of times the centering goal is accomplished. Even in cases where a random population has been deprived of its best rules, the apportionment of credit mechanism rates the remaining rules and the genetic algorithm searches for new, better rules.

These results build our confidence in the learning classifier system as an artificial learning and decision-making device. In the next chapter, we further test its capability by applying the LCS method to the control of a highly variable pipeline environment.

CHAPTER 6

PIPELINE CONTROL WITH A LEARNING CLASSIFIER SYSTEM

> The idea of tying computers to control equipment to produce an 'automatic distribution system'. . . does not seem impossible to me. Present use of telemetric and remote control equipment has already gone a long way to produce just that. How much farther it will go in the future, and just how fast, is entirely a question of economics. - E. F. Trunk, Gas Engineer.

The words sound fresh, as if uttered only yesterday, but, in fact, they appeared in Gas magazine [7] in 1955 as part of a symposium on computers in the gas industry. The meteoric ascent of the digital computer at that time increased expectations for rapid automation of pipelining and network operations to the point where total control of a pipeline by computer seemed imminent:

> It would appear to me that the future of computers in the gas industry is unlimited. . . . The only objection to having an automatic distribution system is money. - J. P. Clennon, Gas Engineer.

While optimistic enthusiasm for new technology is laudable, in this case, it proved a bit premature. Though computers have played an increasing role in communications, simulation, and optimization in pipelining practice, the dream of an automatic system is not much closer to fruition today than it was in 1955.[1] Why is this so? Were our

---

[1]Some might argue that we are further away from this

writers correct and we have simply not invested enough money to go all the way to human-free, closed-loop systems? Money was and has not been the only obstacle as was recognized even back in 1955 by another symposium participant:

> Within the foreseeable future, no computer will be able to perform any operation which has not, first, been anticipated by its designer and taught to the computer. Since unforeseen emergencies do occur, we cannot yet hope to build into a computer the ability to respond correctly to any situation that may arise. The possible consequences of computer response to conditions arising through some unpredictable accident are too serious to accept. While continuous automatic indication of ideal distribution dispatching may well be commonplace in 20 or 10 or even 5 years, there must be retained at least one element of human judgement with full power to veto any operating order which the computer may produce.
> - J. H. Starr, Gas Engineer.

As this author realizes, conventional computer systems are doomed for autonomous control applications because of their brittle nature.[2] If we must anticipate all possible changes in the future before we can implement automatic systems, we cannot succeed because, surely, we will miss

---

goal because we have lost our enthusiasm for new computer discoveries and applications; everyone knows the way to apply computers in their fields, and people do not work on the innovations required for smarter machines. Even in AI (artificial intelligence) research--where we should expect a high level of innovation in this direction--one is overwhelmed by the prevalence of convention over invention, as evidenced by the burgeoning numbers of straight expert system applications.

[2]While this author recognizes the brittleness of conventional computer systems, he fails to recognize or admit the possibility of adaptive computers. He is also guilty of an implied double standard toward man and machine. For his automatic distribution system he requires the "ability to respond correctly to any situation." Does he require such perfection from his human dispatchers? We must overcome this double standard, one which is widely held, if we are to judge our computer efforts fairly.

something or anticipate the future incorrectly. This truth places a premium on learning and adaptation; the need for effective adaptation is crucial to have any hope for autonomous pipeline system control.

In this chapter, we strike at the heart of these issues directly by applying a learning classifier system to the control of a highly variable pipeline environment. Not only do we require the system to perform under normal conditions, we also expect the system to detect the presence or absence of leaks on the pipeline. Furthermore, the test is made more difficult by our choice of starting conditions. Unlike Mr. Starr's computer system where everything must be anticipated beforehand, in our tests, nothing is anticipated; we start from a randomly generated initial state of mind. This places our focus right where it should be, on the system's ability to learn and adapt.

We start our journey down this promising path by describing the simulated pipeline environment and LCS interface particulars. While the environment is a simulated abstraction of a real environment, we are careful to preserve important characteristics which make good performance a challenge. After this description, we present results from a variety of simulations. Specifically, we look at normal operations--summer and winter--and leak upset operations. We compare results with and without the genetic algorithm to a random walk as we did in the previous chapter.

6.1 Environmental Description

We develop a fairly complete, though simplified, pipeline environment to test the breadth of LCS techniques in pipeline control. In this section, we describe the pipeline model, load model, and supply alternatives, as well as the upset conditions which together comprise the pipeline environment.

Pipeline Model

For this portion of our study, we adopt a simplified model of the pipeline dynamics as compared to the transient optimization study in Chapter 4. We ignore gas inertia and use a simple volume, nonlinear resistance model of pipeline behavior. This permits simpler algebraic computations and larger time steps, thereby promoting more efficient computation. The LCS computations are demanding enough without burdening the little Apple II with a detailed pipeline model.

The model may be viewed simply as a balloon and pipe as shown in Figure 6-1. We write a simplified continuity equation for the balloon as follows:

$$Q_i - Q_o = dV/dt$$

where: Q_i - inflow
Q_o - outflow
V - line pack
t - time

We combine this with a no-inertia equation of motion:

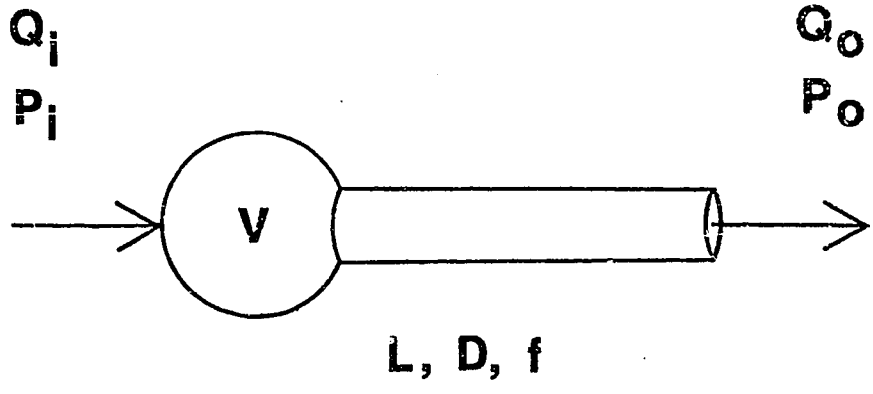


Fig. 6-1. Simplified Pipeline Model Schematic

$$P^2_i - P^2_o = K*Q_o*|Q_o|$$

where: P_i - inlet pressure
P_o - outlet pressure
K - resistance

The two are linked by an isothermal equation of state:

$$V = c_1*P_i$$

where: $c_1 = T_s*A*L/(P_s*T)$

Together, these equations define our simplified pipeline dynamics. If we assume a linear variation of the quantity $Q_i - Q_o$ from one time step to the next, we may integrate the set exactly using a trapezoidal rule integration. The detail of this formulation is straightforward; however, we do not pursue it further as this might be seen as an endorsement of this type of modeling for normal engineering work. More complete models such as the one developed in Chapter 4 and those referenced in Chapter 2 should normally be used. We adopt simpler models here because of the constraints of our computing environment.

Load Model

In our system, we have two types of load pattern which may occur: summer and winter. Both patterns are cyclical, repeating on a daily basis every T_{daily} time steps. The seasons change every T_{season} time steps and each major season change is preceded by a minor seasonal change--roughly equivalent to spring and fall--where the ambient temperature changes but the load pattern does not yet shift.

We adopt a pattern of loading shown in Figure 6-2. The daily variation roughly corresponds to some results presented by Boyer [78] who used cluster analysis to identify typical daily usage patterns for a California gas utility.

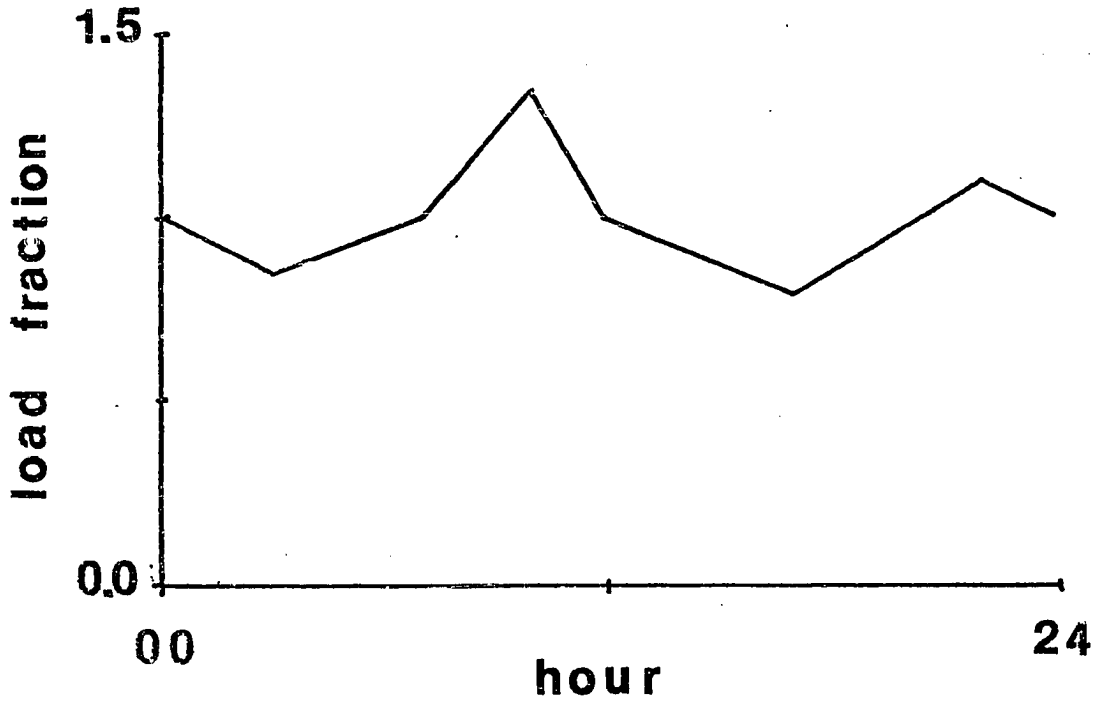
Supply Alternatives

As in our earlier study of transient optimization, our major control variable here is the pipeline inflow. We may supply different levels of flow between Q_{min} and Q_{max} depending upon the discretization of the effector. For this study, we permit 4 levels of flow to be specified by the LCS.

Upset Conditions

In addition to normal summer and winter conditions, the pipeline may be subjected to a leak upset. During any given time step, a leak may occur with probability p_{leak}. If a leak occurs, the leak flow, Q_{leak}, is removed from the upstream junction; however, this is not directly reflected in any system measurements. The leak persists for T_{leak} time steps.

Together, the pipeline, loading, supply alternatives, and upset conditions specify the pipeline environment. While we have simplified it a great deal, we have not eliminated the high load variability normally encountered in practice; nor have we removed the unpredictable to make the task easier. The LCS must learn to deal with both the



summer — 25 mmcfd

winter — 50 mmcfd

Fig. 6-2. Daily Loading Patterns

expected and the unexpected if it is to be successful in our environmental abstraction. In the next section, we examine the LCS-environmental interface to see what kind of cues the learning system can get from the environment.

6.2 LCS-Environmental Interface

The LCS is presented with a fairly complete, yet fairly crude, picture of its environment. In this section, we examine that picture. We also specify the reward mechanism adopted to evaluate the LCS's decisions and actions.

Environmental Message

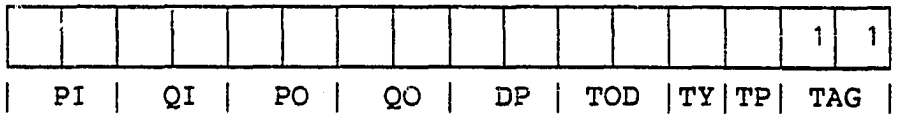
The environmental message template for this problem is shown in Table 6-1 along with the interpretations of the various codings. The system has complete, albeit imperfect and discrete, knowledge of its state including inflow, outflow, inlet pressure, outlet pressure, pressure rate change, season, time of day, time of year and current temperature reading.

Reward Mechanism

As in the inertial object study, we install an ever-vigilant computer procedure to consistently administer reward to the LCS. We describe the procedure in pseudo-code as follows:

Table 6-1

Pipeline LCS
Environmental Message Template



Variable	Description	min	max	# of positions
PI	inlet pressure	0	2000	2
QI	inlet flow	0	80	2
PO	outlet pressure	0	2000	2
QO	outlet flow	0	80	2
DP	u. s. pressure rate	-200	200	2
TOD	time of day	0	24	2
TY	time of year	0	1	1
TP	temperature	0	1	1

```
Every T_eval th time step
 if there is a leak and it is detected then
     points=ptsleak else points=0
 if there is no leak and
  if it is winter and the pressure is ok then
     points=ptsnotover
  or if it is summer and the temperature is hi
                    and  (pressure is lo-ok or
                    lowering) then
     points=ptsnotover
  or if it is summer and temperature is lo
                    and pressure is hi-ok or rising
                    then
     points=ptsnotover
  or if pressure is below acceptable
                    but system is packing at max rate
                    then
     points=ptspressure
  or if pressure is above acceptable
                    but system is drafting at max rate
                    then
     points=ptspressure.
```

The pressure ranges described in this procedure are illustrated in Figure 6-3. In words, a variety of point rewards may be given to the LCS depending upon the action taken and the current state of the pipeline. If a leak is present and it is detected, a certain level of reward is given; no points are given for an undetected leak regardless of other actions. If no leak is present, points are awarded depending upon the season, pressure and temperature level. Additionally, if the pressure is out of range, but the system is packing or unpacking to return the system to an acceptable pressure, then another quantity of points is awarded.

The normal level of award, ptsnotover, is greater than the out-of-range value, ptspressure, because in the former, both the action and system state are correct, while in the

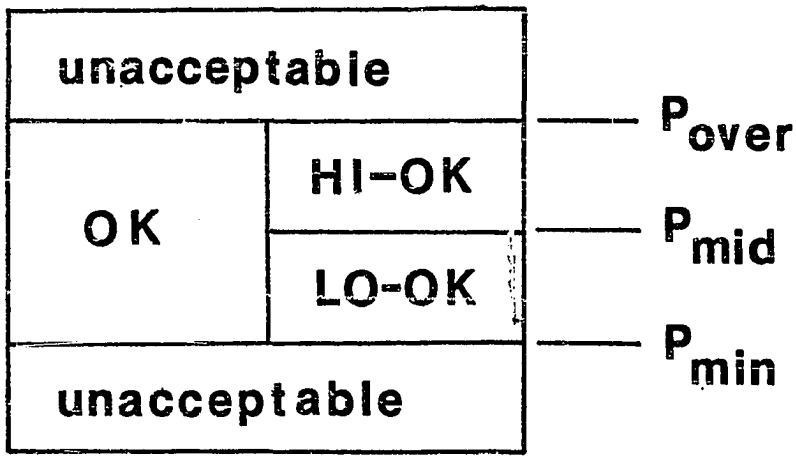


Fig. 6-3. Pressure Levels for Reward and Penalty

latter, only the action is correct. The leak award is generally higher than the other values because the occurrence of a leak is a relatively low probability event, thereby necessitating higher award for leak rule survival.

With a reward mechanism specified and LCS-environmental interface drawn, we proceed toward simulation results by first examining salient implementation details and system parameters.

6.3 Implementation Details

In this section, we attack some remaining details of implementation and summarize the LCS and environmental parameters used in the pipeline operations learning tests.

The pipeline environment has been coded in Pascal and hooked up with the learning classifier system of the previous chapter. Careful, modular programming eliminates the need for LCS modifications when arranging a new application. The skeletal description of the LCS, presented as Appendix B, is still an accurate representation of the LCS code, except that the environment is replaced by the pipeline operation environment also presented in Appendix B.

The learning system parameters of the previous chapter have been adapted for this study. C_{bid} must be scaled to reflect the longer message length and higher possible maximum matchscore. Other length dependent values have been adjusted to reflect the new message length. All the parameters are summarized in Table 6-2.

The environmental parameters are displayed in Table

Table 6-2

LCS Parameters
for
Pipeline Operation Tests

n_{class} = 40 (normal), 60 (leak runs)

l (message length) = 16

n_{mess} = 5

x_{lo} = 1

x_{hi} = 14

$p_{generality}$ = 0.75

$p_{mutation}$ = 0.001

C_{bid} = 0.0156

σ_{bid} = 1.0

$n_{replace}$ = 3

C_{tax} = 0.002

T_{ga} = 200

T_{eval} = 1

PROPORTION = 0.05 (normal), 0.0333 (leak runs)

6-3. The line is of moderate length, 94.7 miles, and diameter, 1.0 foot. As we saw previously, the load varies greatly from summer to winter. Furthermore, we note that the leak flow Q_{leak} is a significant multiple of the normal through flow. This flow rate is realistic as it represents the blowdown rate we might see if we were to punch a 4 inch diameter hole into the line at a pressure of 1000 psia.

Table 6-3

Environmental Parameters
for
Pipeline Operation Tests

pressure units = PSIA
flow units = MMCFD

dt = 1 hour
gas gravity = 0.6 (relative to air)
P_{std} = 14.73 PSIA
T_{std} = 520° R

P_{over} = 1500 PSIA
P_{mid} = 1000 PSIA
P_{min} = 500 PSIA

Pipe length = 500000 feet
Pipe diameter = 1.0 feet
f factor = 0.01
Gas temperature = 520° R

K1 = 300 (compressor coefficients)
K2 = 295
K3 = 0.25

T_{daily} = 24
$T_{seasons}$ = 480

LEAKQ = 250 MMCFD
P_{leak} = 0.2

ptsnotover = 6
ptspressure = 5
ptsleak = 12 (leak runs only)

Summer Load (time,load)

(0,25)	(3,20)	(5,25)
(10,35)	(12,25)	(17,20)
(20,25)	(22,30)	(24,25)

Winter Load

(0,50)	(3,40)	(5,50)
(10,70)	(12,50)	(17,40)
(20,50)	(22,60)	(24,50)

6.4 Normal Operating Simulations

With a well-specified learning system and test environment established, we proceed with several learning tests under normal operating conditions (no leaks). As in the inertial object tests, we look at learning with and without genetic algorithm and compare those results to a random walk through the decision space.

For this series of tests we generate an initial population of rules of size, nclass=40; ten rules are reserved for each of the four possible decisions corresponding to the four possible input flow rates. The rules are restricted to the following form:

RRRRRRRRRRRRR11:##############11/< ACTION MESSAGE >

As in the inertial object study, the R signifies a position chosen at random with the generality bias probability, $p_{generality}$.

In the first set of learning tests, we start from the initial population presented in Table 6-4. In run POLCS.1, the learning proceeds with apportionment of credit only (no GA). In run POLCS.2, both kinds of learning are enabled. Results from these tests are presented in Figure 6-4, a graph of time-averaged total evaluation vs. time. The pattern displayed in this figure is familiar; both learning tests outperform the random walk. Furthermore, while outperformed at first, the run with genetic algorithm soon beats the case without. This is not unexpected as we obtained similar results in the inertial object simulations.

Table 6-4

Initial Rule Population
Runs POLCS.1 & POLCS.2

```
#100##########11:##############11/1110000000000000
1###11#1##1###11:##############11/1110000000000000
#0#####0#1###111:##############11/1110000000000000
1#0#####10#1##11:##############11/1110000000000000
###0##0#1####011:##############11/1110000000000000
#0###1#0###0##11:##############11/1110000000000000
0#1#000#######11:##############11/1110000000000000
##0#11##10#1##11:##############11/1110000000000000
1#######1#0###11:##############11/1110000000000000
###0#0########11:##############11/1110000000000000
#0##1######1##11:##############11/1010000000000000
1#####1#######11:##############11/1010000000000000
#####110###11#11:##############11/1010000000000000
######1#######11:##############11/1010000000000000
11#######1####11:##############11/1010000000000000
##1#####0#####11:##############11/1010000000000000
##0##1###1100#11:##############11/1010000000000000
###0##########11:##############11/1010000000000000
#1001##11#####11:##############11/1010000000000000
#1######0#####11:##############11/1010000000000000
##0#########0#11:##############11/0110000000000000
1###1#####01##11:##############11/0110000000000000
1#######1#####11:##############11/0110000000000000
1####1#####11#11:##############11/0110000000000000
#######1###10#11:##############11/0110000000000000
##########01#111:##############11/0110000000000000
###001##1#####11:##############11/0110000000000000
0##1###100####11:##############11/0110000000000000
######1#######11:##############11/0110000000000000
###1##0####0##11:##############11/0110000000000000
###1#1########11:##############11/0010000000000000
##1##0##1###0011:##############11/0010000000000000
1#01######0##011:##############11/0010000000000000
######1#10###111:##############11/0010000000000000
0#110#####10#011:##############11/0010000000000000
#######1###1##11:##############11/0010000000000000
#1##0#####11#111:##############11/0010000000000000
##00##########11:##############11/0010000000000000
##11##11######11:##############11/0010000000000000
1####1###11##011:##############11/0010000000000000
```

To further explore why this occurs, we delve into the particular rules learned by the system. In Table 6-5, we

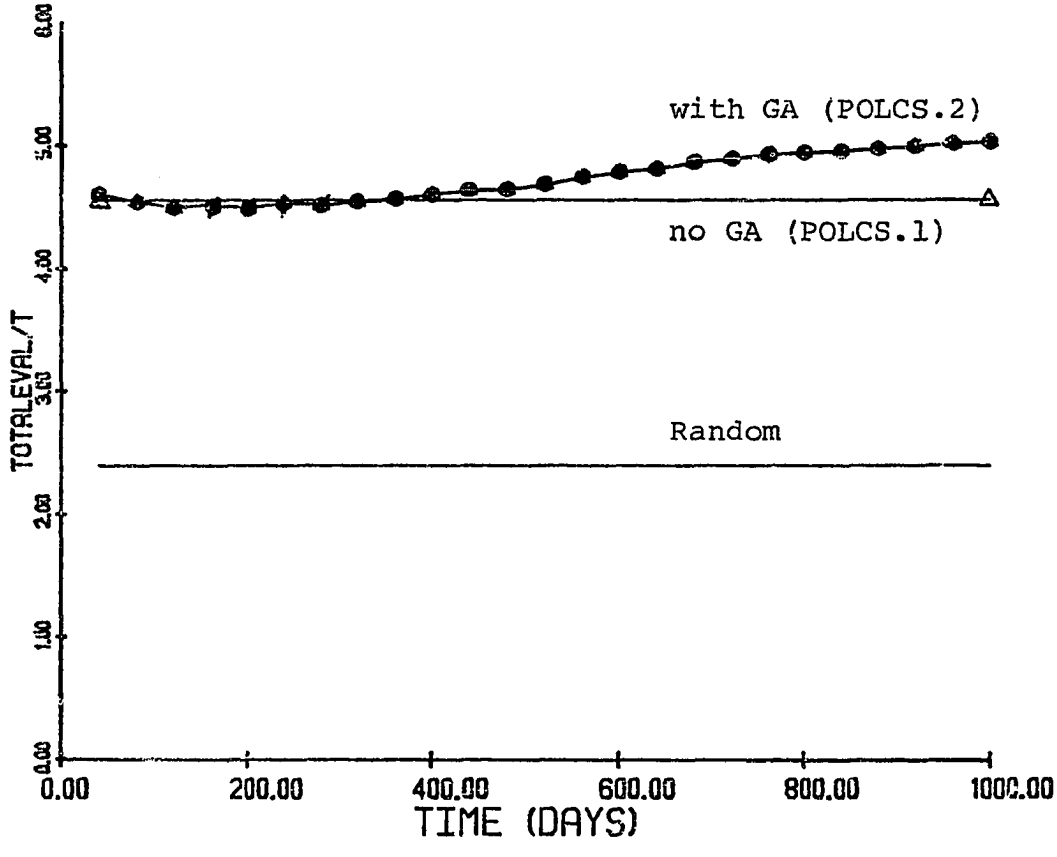


Fig. 6-4. Time-averaged TOTALEVAL vs. Time. Normal Operations. Runs POLCS.1 & POLCS.2

see the top rules, those exceeding 3 times the average strength at each run's end. Comparing the run POLCS.1 (no GA) rules to the environmental template, the rules seem to be a hodgepodge of not nonsensical, but not entirely clear, rules. For example, the two top rated rules translate loosely as follows:

If [(P_i is hi)
and (DP_i/dt is (very negative or moderately positive))]
then [set inflow=Q_1]

If [(P_i is low) and (Q_o is low) and (temp is hi)]
then [set inflow=Q_{max}]

Although the rules are not usually counterproductive, they are by themselves, not complete. This is the best we can expect when we start from a random state of mind, and we permit no refinement of the rules through genetic action; the system must choose the best of the bunch to try to cover the operating conditions it sees.

By contrast, when we permit genetic action, the best rules in run POLCS.2 start to approach our intuition of how a system should be controlled. For example, the two best rules of run POLCS.2 translate as follows:

Table 6-5

Top Rule Subset & Strengths (End of Run)
Runs POLCS.1 & POLCS.2

POLCS.1 (no GA)

#1######0#####11:##############11/1010000000000000	54.30
#0#####0#1###111:##############11/1110000000000000	35.20
###1#1########11:##############11/0010000000000000	28.95
#1##0#####11#111:##############11/0010000000000000	26.79

POLCS.2 (w/ GA)

#0############11:##############11/1110000000000000	65.44
########0#####11:##############11/0010000000000000	60.31
######1#10####11:##############11/0010000000000000	37.07

if [(P_i is low)] then [set inflow=Q_{max}]

if [(dP_i/dt is (extremely negative or moderately positive))]
then [set inflow=Q_{min}]

These simple rules, a simple pressure threshold and a multi-level rate threshold, seem more natural than the earlier mixture of complex rules (The pressure rate rule makes more sense than it first seems because the extremely negative DP/dt is not seen under the normal conditions of this run). Judging by the relative performances, these rules are also a good bit more effective than their no-GA counterparts.

A confirmation set of runs is started from a different, randomly generated set of rules, runs POLCS.3 and POLCS.4. Again, the comparison of time-averaged total evaluation

yields the same result (shown in Figure 6-5): the run with genetic algorithm outperforms the run without, and both outperform the random walk.

6.5 Leak Detection Simulations

To test the breadth of the LCS method, we see if the system can learn, not only normal operating rules, but also rules that detect leaks. Recall that the LCS has only a limited discretization of its environment to work with and that no rules have been implanted to help it find its way. Certainly, if the system can learn to detect leaks, we have reason to hope that we can teach it many other tasks required in a real operating environment.

In the leak runs, we again start from a randomly generated set of rules. In these tests, 40 are devoted to the four flow effectors, and 20 are allocated to the two external leak messages: leak is present and no leak is present. Once again, we compare performance with and without a genetic algorithm to a random walk. In runs POLCS.5 (no GA) and POLCS.6 (with GA), history repeats itself with the GA run outperforming the no-GA run, and both outperforming the random walk on the basis of time-averaged total evaluation shown in Figure 6-6. It is also instructive to look at the percentage of leaks alarmed correctly in Figure 6-7. Without the genetic algorithm, the percentage of leaks alarmed correctly starts out very high and remains stationary throughout the simulation. By contrast, the simulation with genetic algorithm starts out

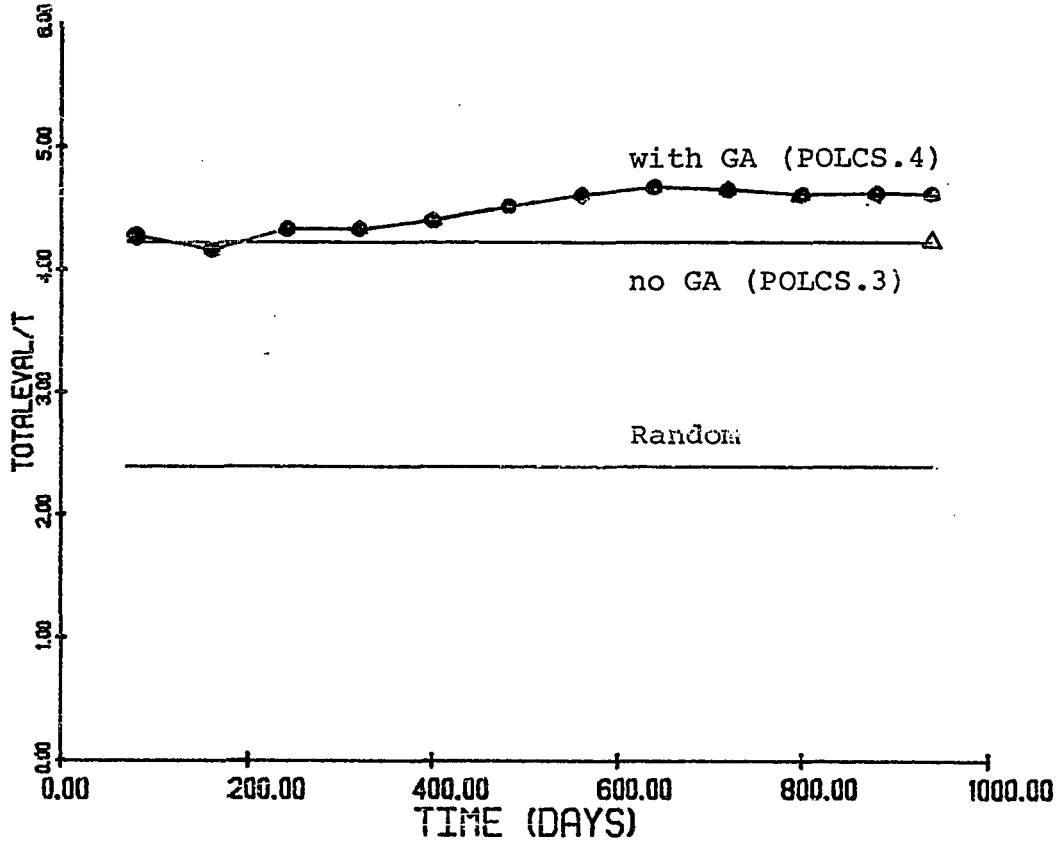


Fig. 6-5. Time-averaged TOTALEVAL vs. Time - Normal Operations - Runs POLCS.3 & POLCS.4

lower and approaches (and would eventually surpass) the no-GA results on this basis. This seems surprising until we examine the percentage of false alarms (shown in Figure 6-8). The high percentage of correct leak detections for run POLCS.5 (no GA) is bought at the expense of a very high (worse than random) percentage of false alarms. With the genetic algorithm enabled, the system learns to not alarm falsely at the same time it learns to detect correctly. This results in the large differential in point score between the two simulations.

If we examine the rule set at the end of run POLCS.6 (with GA) we see the reason for its high level of success in detecting leaks. Among the leak rules, those with a leak or no leak message effector, two rules predominate at the end of the run:

if [anything] then [send no leak message]

if [(P_i is low) and (P_o is low) and
(dP_i/dt is very negative)]
then [send leak message]

In other words, by default the system sends the no leak message; the leak message is only sent under very specific conditions. We note that the leak rule designed by the system is not among the original random population. Furthermore, the rule is the ideal leak detection rule for this system because the specified conditions are precisely those the system will see with the magnitude of leak

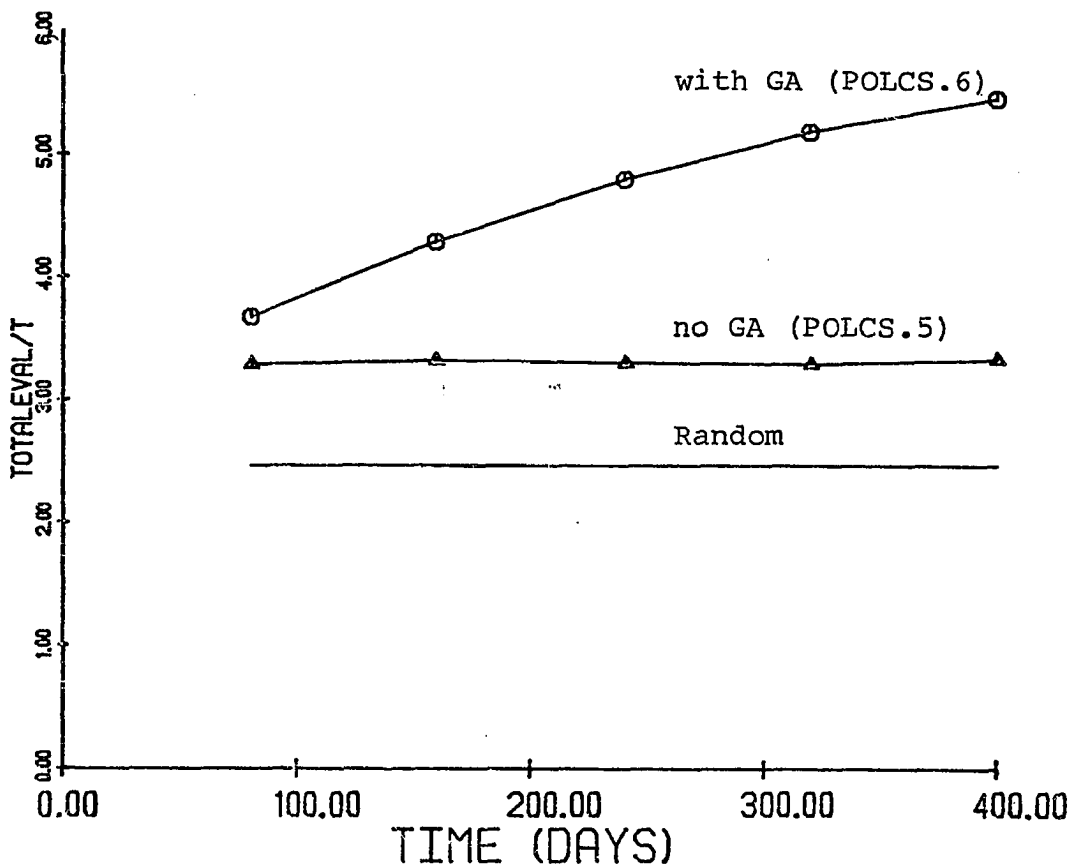


Fig. 6-6. Time-average TOTALEVAL vs. Time - Leak Runs - POLCS.5 & POLCS.6

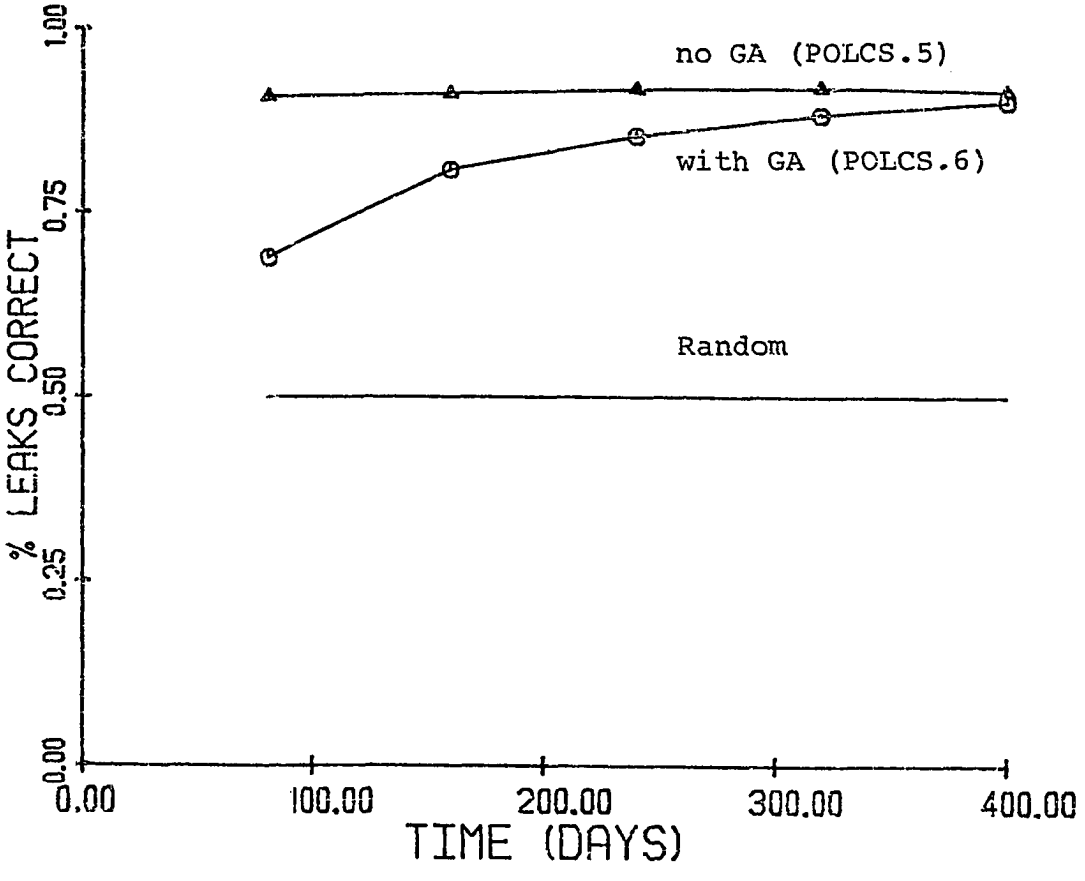


Fig. 6-7. Percentage of Leaks Correct vs. Time Runs POLCS.5 & POLCS.6

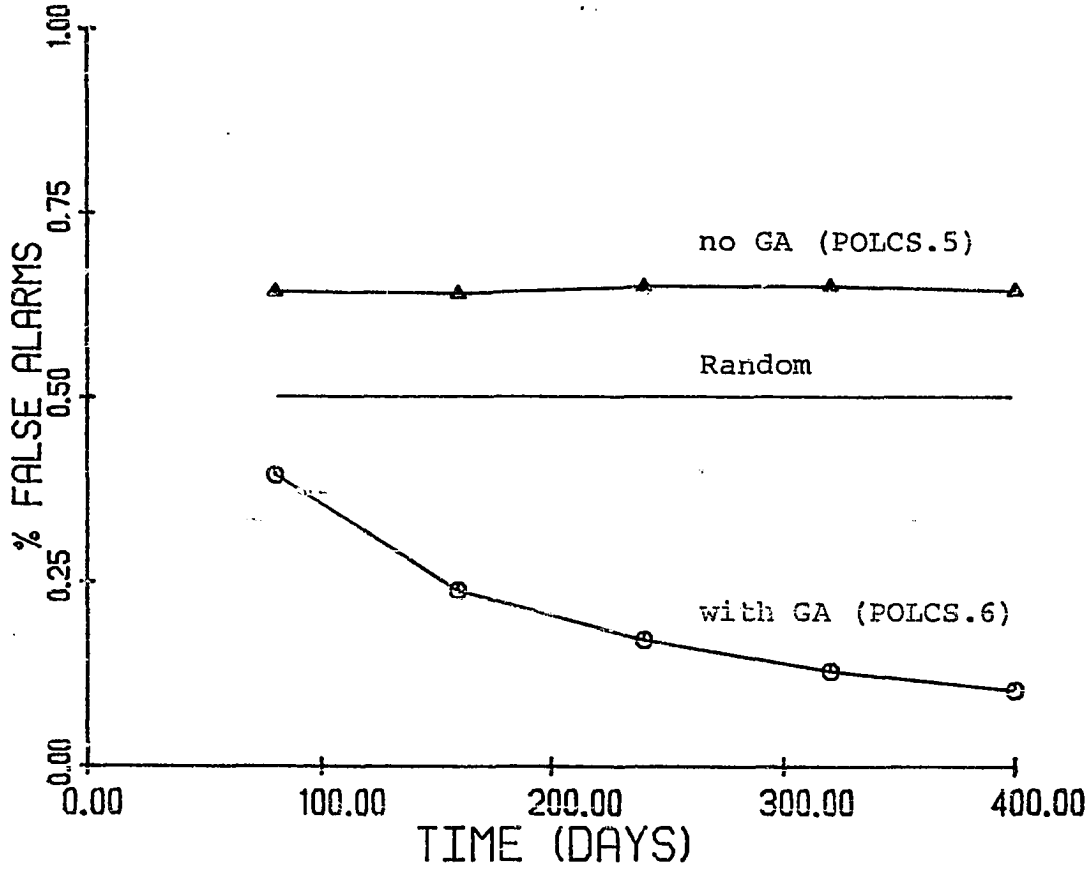


Fig. 6-8. Percentage of False Alarms vs. Time
Runs POLCS.5 & POLCS.6

imposed.

A confirmation set of runs is performed using a different, randomly generated, rule population. The results are presented in Figures 6-9 (time-averaged total evaluation), 6-10 (% leaks correct), and 6-11 (% false alarms). In these cases, the run with GA does not find the ideal leak detection rule; however, its GA-refined rule set still beats random and no-GA performance by a significant margin on the basis of time-averaged total evaluation. In the other measures, percentage leaks correct and percentage false alarms, run POLCS.8 (with GA) outperforms run POLCS.7 (no GA); by contrast, in the previous runs, the no-GA case outperformed the GA case in the percentage leaks detected measure, but this was countered by the high percentage of false alarms encountered. In the present cases, GA performance is consistently better than no-GA performance in all measures, although the differences are less dramatic than in the previous cases. Examination of the end-of-run high performance rules shows why the runs differ: in runs POLCS.7 and POLCS.8 there is no evidence of the crucial leak detection schema, "if [very negative pressure rate] then [send leak message]." The schema did not exist in the initial population (In fact, the schema "very negative pressure rate" did not exist in any rule in these runs). Furthermore, generation of this schema (via GA) during the run is a fairly low probability event because it requires a mutation at one or two specific locations or a cross at a

specific site between two specific alleles in different rules. The problem here (if there really is one--the GA run did outperform the no-GA run, and both outperformed the random walk.) is the small population size. Higher order schemata are not present in the starting population in sufficient quantities because of the small population sizes adopted. Larger populations should be used to provide sufficient expected numbers of higher order schemata in starting populations generated at random. Small populations have been used in this study to keep computational requirements (which go up as the product of rule population size and the message list length) to a minimum. The use of large populations in future applications will rectify this situation.

6.6 Summary

In this chapter, we have applied a learning classifier system (LCS) to the control of a varied and uncertain pipeline environment. Starting from randomly generated rules, the system has learned, not only to operate under normal summer and winter conditions, but also it has learned to detect the presence or absence of leaks with increasing certainty.

In two series of normal operating condition tests, the LCS has consistently outperformed a random walk. The simulations with GA have, once again, improved upon the initial rule set by outperforming the runs without GA. Examination and comparison of the best rules has

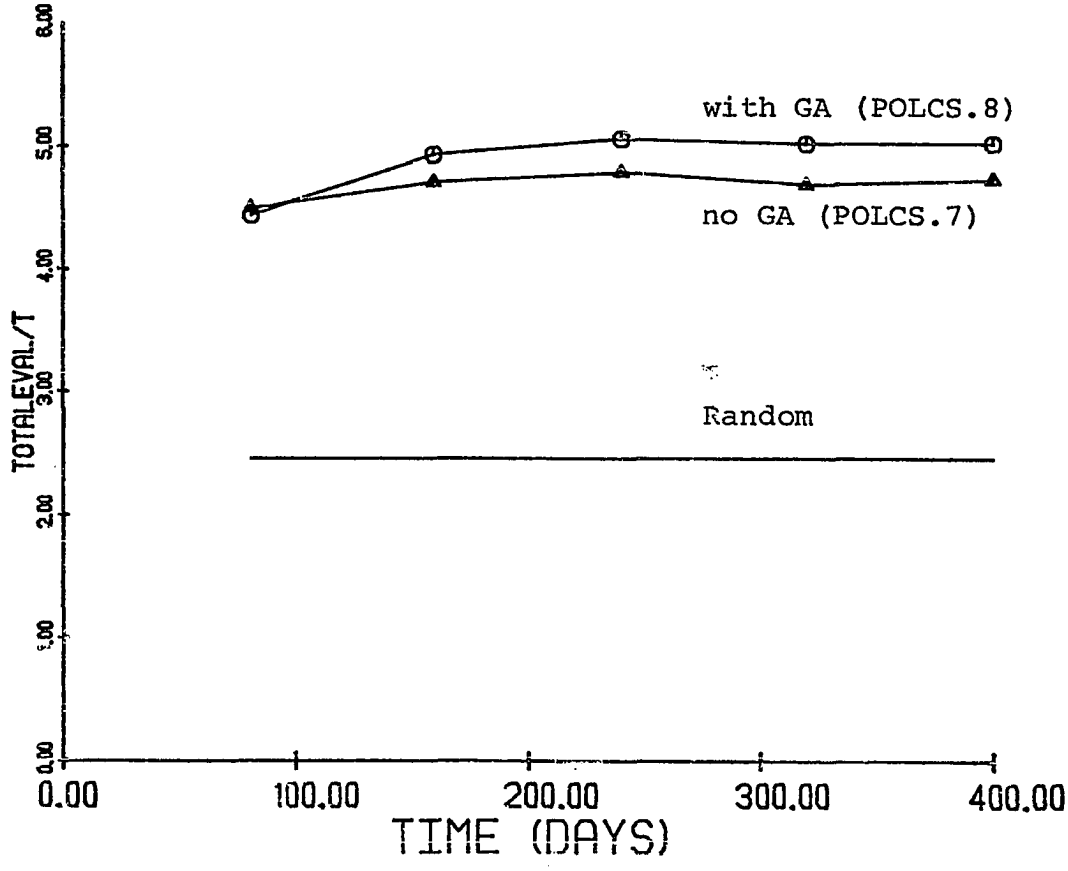


Fig. 6-9. Time-averaged TOTALEVAL vs. Time - Leak Runs POLCS.7 & POLCS.8

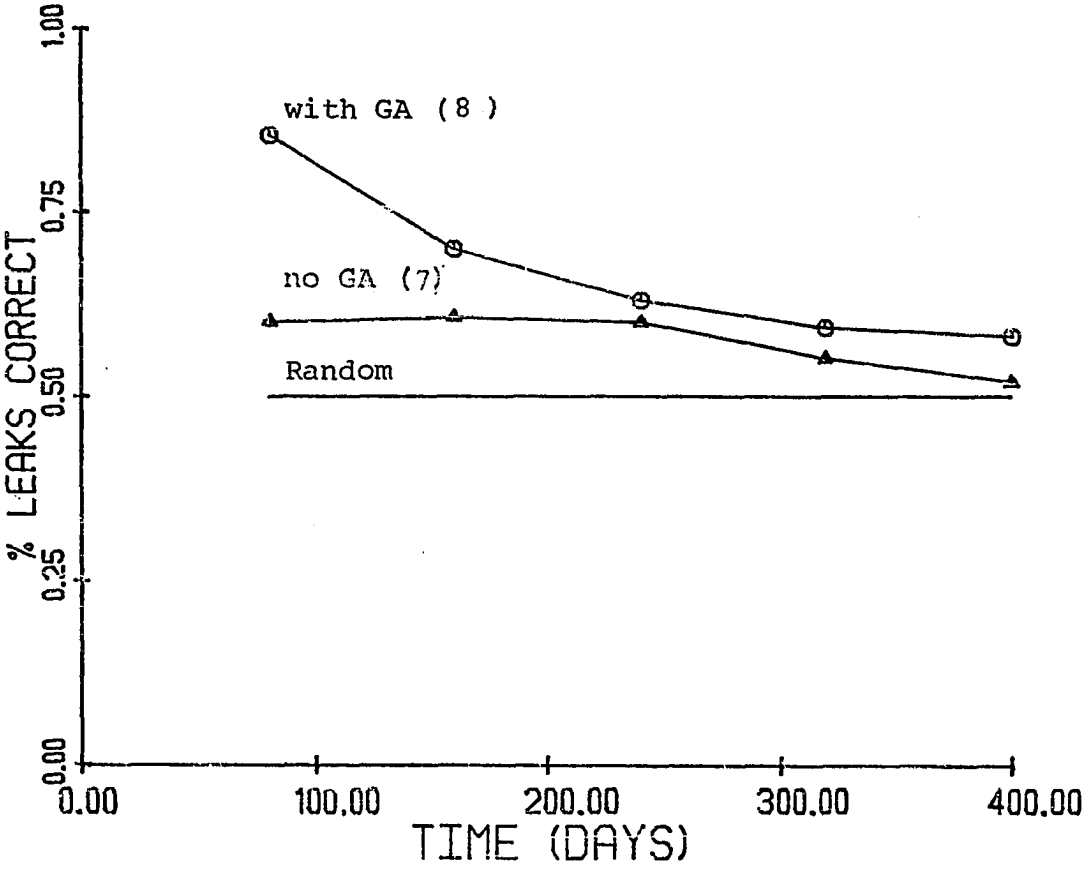


Fig. 6-10. Percentage of Leaks Correct vs. Time - Leak Runs POLCS.7 & POLCS.8

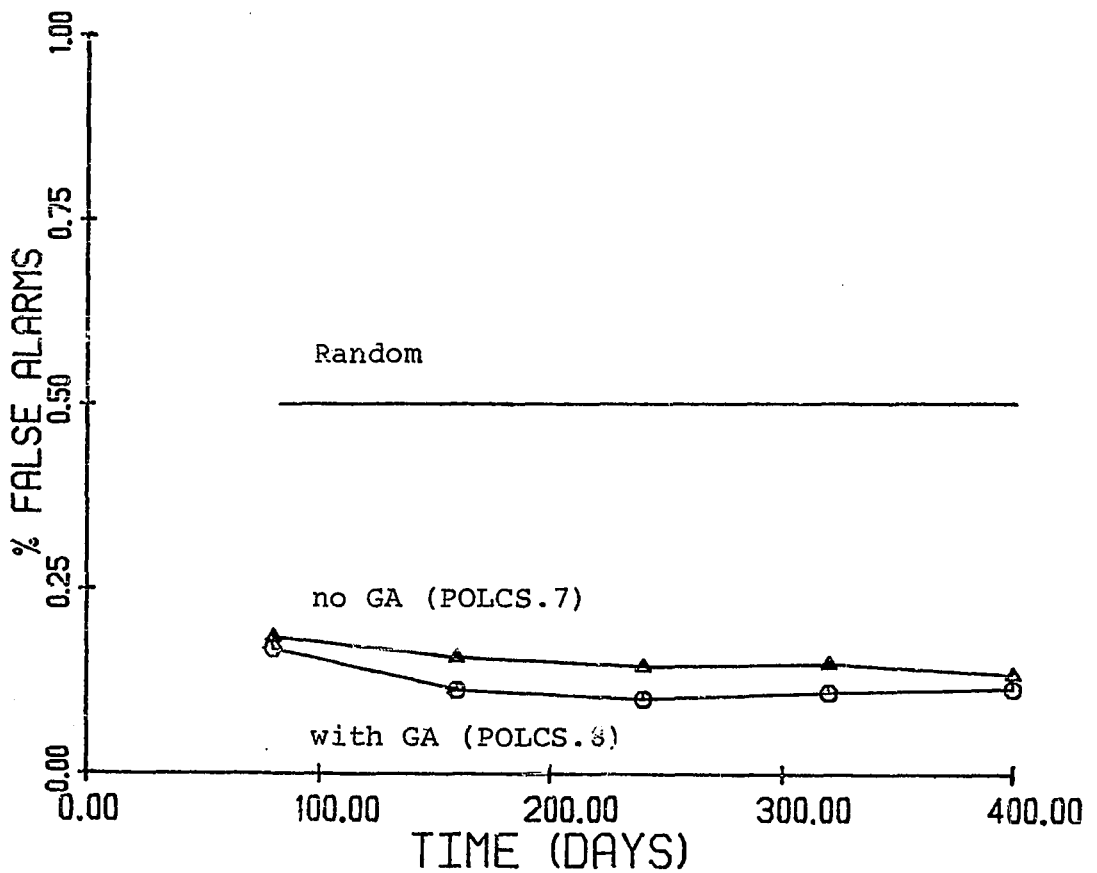


Fig. 6-11. Percentage of False Alarms vs. Time Runs POLCS.7 & POLCS.8

demonstrated the increased effectiveness of the GA-discovered rule set.

Two sets of leak detection runs have been performed. Starting from a random rule set, the learning system has outperformed the random walk. The GA runs have outperformed the no-GA runs. Examination of the leak detection rules has shown definite movement toward pressure rate leak detection as we should expect. Indeed, in one of the two leak runs the ideal leak detection rule has been discovered even though it did not exist in the original rule population.

CHAPTER 7

CONCLUSIONS

Our goals in this study have been clear: we have sought robust learning and decision algorithms for operating a gas pipeline. We have emphasized the need for robustness--the efficiency and breadth of performance we observe in human pipeline operators--because the varying pipeline environment is fraught with change and uncertainty; any pre-programming, modeling, or a priori decision making is doomed to failure when conditions change, thereby violating the assumptions contained in the programs, models, or decision rules we so carefully constructed at an earlier time.

Although it is easy to talk about algorithmic efficiency and breadth, finding examples of robust adaptation procedures is not so simple; a survey of methodologies has shown that most common artificial procedures have two shortcomings: locality and structural rigidity. Because of these shortcomings, we have abandoned the frontal attack--traditional optimizers and the learning systems that use them--and instead, have staged a two pronged assault on our objective.

First, we have tested the flanks by investigating a genetic algorithm in two pipeline optimization applications. This algorithm has demonstrated more of a global flavor than other search procedures we commonly encounter.

Second, we have struck at the heart of structural rigidity through the development and application of a learning classifier system, first in an inertial object environment and then in a pipeline environment. The learning classifier system avoids rigidity because it is a rule-based computational scheme that, in a sense, programs itself with better and better rules.

In this chapter, we review our progress in genetic algorithm and learning classifier system applications. We also recommend some important directions for the continuation of this work.

7.1 A Genetic Algorithm and Pipeline Optimization

In our bout with a genetic algorithm (one of a class of algorithms) we have detailed its mechanics and effect. Generally, genetic algorithms imitate the mechanics of natural genetics by combining a survival-of-the-fittest notion (reproduction) with a randomized, though structured, information exchange (crossover) among strings in a population. These operators involve nothing more complex than string copying and partial string swapping, yet, we have seen how this simple process effects a rapid search among alternatives by independently sampling building blocks, short high strength schemata, at near-optimal rates.

Intuitively, this process is appealing because it is a kind of innovative search where new ideas are boldly formed from the best pieces of our old ideas.

Our genetic algorithm also contains a strictly random operator, mutation, thrown in as an insurance policy against unrecoverable loss of information. Many are surprised that this operator only plays a secondary insurance role in the search process; genetic algorithms are not coin flipping by a fancy name. Crossover and reproduction carefully exploit existing information to search for improvement in future generations.

The three operator (reproduction, crossover, and mutation) genetic algorithm has been tested on two pipeline optimization problems, steady state operation of a serial line and transient operation of a single line. In both cases, in a variety of runs, near-optimal results have been found after examining an infinitesimal fraction (10^{-6} - 10^{-7}%) of the search space.

These results have established the genetic algorithm approach as a practical methodology in engineering optimization. They also have bolstered our confidence in using the genetic algorithm as a component in a more flexible learning system, a learning classifier system.

7.2 A Learning Classifier System Controls a Pipeline

A learning classifier system (LCS) has been developed to control a simulated pipeline system. An LCS is a learning system that creates, evaluates, and exploits string

rules for high performance interaction with some arbitrary environment.

Our LCS learns in two ways. First, existing rules are evaluated by an apportionment of credit algorithm modeled after a competitive service economy. Rules bid to become active and pay their bids to message-sending predecessors. In this way, rules gain or lose accumulated wealth depending u : their ability to set up reward from the environment.

Second, the system learns new rules by using a genetic algorithm similar to the one in the optimization studies. New rules are created by reproducing, crossing, and mutating rules in the current rule set. Thus, new rules are generated from the best pieces of the old; they are then inserted into the population and evaluated by the apportionment of credit mechanism.

In two different environments, an inertial object and a pipeline, an LCS has learned effective rules for high performance control. In the inertial object environment, both braking rules and restoration rules have been learned to center the inertial object after it is disturbed from rest. In the pipeline environment, rules have been learned to control the pipeline during normal summer and winter conditions; the system also learns how to detect leaks with increasing effectiveness. In all cases, the learning has proceeded from a random state of mind; no rules have been implanted to help the system learn its task more easily.

7.3 What Needs to be Done?

The coming application of genetic methods in tougher problem domains will demand fundamental study of a number of genetic operators and mechanisms. Crowding, locus rearrangement (inversion), and dominance mechanisms should be studied empirically to verify the theories advanced to date (ANAS). Crowding is particularly important in multi-modal problems (and learning systems) because it divides population slots among different peaks. Locus rearrangement operators like inversion should be studied to verify their usefulness in GA-hard problems; Bethke's methods may be used to design GA-hard codings of GA-easy functions. Dominance operators should be studied to naturally prevent allele loss without disruptive mutation rates.

The LCS system may be extended simply to achieve more computational convenience. At present, recognition of environmental patterns and data is easy; transformation of data is not. We suggest the extension of the LCS alphabet by a single shift and transfer character, the dollar sign $. In a condition a $ behaves like a #. In a message, it transfers data from the condition side (processing left to right) to the open $ slots in the message side. Such an operation is necessary for convenient formation of time histories and other transformed representations of incoming data.

Beyond this we should strive to unify messages, classifiers, and the algorithms that process them. If we can

extend the rule and message system simply, so the basic genetic operators (and AOC?) may themselves be written in rule form, we can achieve the ultimate in flexibility, a system that learns to learn as it learns to perform.

7.4 Are the GA and LCS Ready for Gas Pipeline Control and Vice Versa?

Yes, maybe, and probably not.

In this study, we have demonstrated the practicality of genetic algorithms in pipeline engineering optimization applications. If you have a model of your system, well-defined constraints, and some appropriate objective function, a genetic optimizer is a nice little black box you plug into. In addition to being easy to use, we have also seen how the GA does not rely upon the restrictive assumptions of other methods (unimodality, existence of derivatives, stage decomposability, piecewise linearity, etc.) Indeed, genetic algorithms stand ready, today, to perform practical production optimization studies on pipelines and other engineering systems.

Learning classifier systems are, perhaps, not quite as ready to start performing their roles as expert consultants and knowledge storehouses; however, the progressive pipeline manager may want to keep abreast of their progress and maybe start a pilot investigation into their applicability.

In a pilot study, we should keep both the application and our expectations down to a reasonable size. Implementation of an LCS on an isolated stub line or a

simple main line is feasible; however, as with an infant, we should not expect too much too soon. We should not expect an LCS to solve problems that we cannot solve (although this seems to be one popular success criterion for AI research). Instead, if the system learns to advise its operators in a consistent manner, we should be pleased. If the system starts to take on some of our intuitive knowledge in its rule set, we should be modestly ecstatic.

Is the pipeline community ready for genetic algorithms and learning classifier systems? As we have already speculated, probably not. The notion of using artificial genetics to optimize pipeline operations is sufficiently bizarre to raise a few eyebrows; however, the resistance to these techniques will probably be the same resistance that faces other optimizers. Pre-programmed models and objective functions have a difficult time representing the real world. As a result, human dispatchers tend to distrust optimizer recommendations and rely on their own intuition. Nonetheless, genetic optimizers can be a useful tool in the hands of the skilled dispatcher or engineer; furthermore, the innovative search process underlying the genetic algorithm may appeal to the operator's own sense of innovation.

The notion of a learning system that learns rules of thumb similar to the dispatcher will probably face a different, less justified, kind of resistance. Dispatchers may fear for their jobs, discomfited by the prospect of an

infallible silicon surrogate. Although we understand these fears, our goals run in a different direction. A learning system should replace no dispatchers; it should aid the decision-making process and act as a storehouse of pipelining knowledge. This should ease, not replace, the dispatcher's job, while it adds continuity of experience to the pipelining workplace.

Regardless of the time and place of application of this research, its pursuit can only help with our understanding of intuitive gas dispatching and the performance of other complex technical tasks. Our parting hope is twofold: First, we hope this work spurs other engineers to carry on both applied and basic research in this field. Second, we hope it encourages the trial of these techniques on an operating pipeline system, because in engineering, practical application is the certain touchstone of success.

APPENDICES

APPENDIX A

SKELETAL CODE FOR GENETIC OPTIMIZATION PROGRAMS GENESS AND GENETR

In this appendix, the hierarchy of coding is presented for the genetic optimization work of Chapter 4. Relationships are presented in Pascal-like pseudocode with procedures described in brief comments--(* comment enclosed like this *). We proceed, presenting code from general to particular, in the order of program flow.

Program GENESS

```
program geness; (*  SS Serial Problem *)
  begin (*  main *)
    input; (*  read GA parameters *)
    creation; (*  create a string population and initialize
                  problem *)
    generation; (*  perform subsequent generation
                    calculations *)
  end.


procedure creation;
  begin
    randomize; (*  shake up random number generator *)
    randomstart; (*  randomly initialize string population
                     *)
    initialreport; (*  parameter printout *)
    initmodel; (*  initialize the model *)
    fitnessevaluation; (*  evaluate gen=0 fitness *)
    greport; (*  generation report *)
    plotreport; (*  initial plot report *)
  end;
```

```
procedure generation;
  begin
    for ngen := 1 to maxgen do
      begin (*  generation loop *)
        reproduction; (*  fitness proportionate reproduction
                          and mutation *)
        crossover; (*  mate and cross reproduced strings *)
        fitnessevaluation; (*  evaluate fitness over
                               population *)
        greport; (*  generation report *)
        plotreport; (*  plot report *)
        advance; (*  oldpop := newpop *)
      end (*  end loop *)
  end;


procedure fitnessevaluation;
  begin
    for [all strings] do
      begin
        unpackchrom; (*  unpack chromosome (string) *)
        decodeparms; (*  decode string to sequence of real
                         parameters *)
        modelss; (*  steady state model calculations and
                     cost accounting *)
      end; (*  string loop *)
    sort; (*  order strings by fitness , hi to lo, for
              convenience not necessary *)
    countcalc; (*  calculate reproduction count with
                   probabilistic rounding *)
  end;


procedure modelss;
  begin
    flow; (*  calculate flow and pressure for each pipe-
              compressor *)
    horse; (*  calculate horsepower *)
    constraint; (*  calculate penalty cost *)
    [calculate fitness]
  end;


procedure countcalc;
  begin
    [calculate average fitness of population];
    [calculate fitness count with probabilistic  rounding
      scaling (max=2) until all slots filled]
  end;
```

Program GENETR

The transient program is similar in structure except for

the model. The following code replaces procedure modelss; otherwise the structures are identical.

```
procedure modeltr; (*  transient single line problem *)
  begin
    modelinit; (*  initialize model *)
    modelexec; (*  execute model *)
    costcalc; (*  cost accounting calculations *)
    summaryreport; (*  summary report, if flagged *)
  end;


procedure modelinit;
  begin
    initreset; (*  reset time and accumulator variables *)
    initsstate; (*  set initial steady state conditions *)
    report; (*  variables report *)
  end;


procedure modelexec;
  begin
    while [time is less than maximum]
      begin
        timecalc; (*  calculations in time *)
        report; (*  variables report *)
        advance; (*  advance pressure-flow vars *)
      end (*  time iterations *)
  end;


procedure timecalc;
  begin
    interior; (*  interior point calculations in pipe *)
    boundary; (*  boundary junction calculations *)
    statistics; (*  calculate run statistics *)
  end;
```

APPENDIX B

SKELETAL CODE FOR LEARNING CLASSIFIER SYSTEM (LCS) INERTIAL OBJECT AND PIPELINE OPERATIONS ENVIRONMENTS

In this appendix, the program hierarchy is presented for the learning classifier system work of Chapters 5 and 6. Relationships are presented in Pascal-like pseudocode with procedures described in brief comments--(* comment enclosed like this *). We proceed from general (main program) to particular (lower level procedures) in the the normal order of program execution.

Program LCSIO (Inertial Object)

```
program lcsio; (*  Inertial Object LCS *)
  begin (*  main *)
    initialization; (*  initialize the LCS *)
    initenvironment; (*  initialize the environment *)
    report; (*  t=0 report *)
    repeat (*  thought iteration *)
      lcs; (*  LCS computations *)
      efftocontrol; (*  set actions from effectors *)
      timekeeper; (*  keep time and flags for ga, report,
                      eval *)
      environment; (*  environmental calculations and
                       display *)
      statetomessage; (*  place new environmental state in
                          env. message *)
      reinforcement; (*  reward active e-classifiers *)
      report; (*  report on iteration *)
    until haltflag (*  end of run *)
  end.


procedure initenvironment;
  begin
    initenvdata; (*  read environmental data *)
    initenvreport; (*  initial env. report *)
    initdisplay; (*  set up screen display *)
  end;
```

```
procedure initialization;
  begin
    randomize; (*  shake up random number generator *)
    initlcs; (*  initialize the lcs  *)
  end;


procedure initlcs;
  begin
    readdata; (*  read in LCS parms and data *)
    initreport; (*  printout initial LCS parms *)
  end;


procedure lcs;
  begin
    performance; (*  rule and message system and aoc *)
    if gaflag then ga; (*  genetic algorithm if enabled *)
  end;


procedure performance;
  begin
    matchclass; (*  match classifiers to messages and build
                    pointer list *)
    bid; (*  construct active list and hold noisy bidding *)
    sort; (*  pick best active classifiers by bid *)
    effector; (*  check best actives for e-classifiers and
                  arbitrate among mutually exclusives *)
    payment; (*  payment to previous message senders -
                 clearinghouse and taxation *)
    statistics; (*  update avg, max, min stats *)
  end;


procedure ga;
  begin
    pickmofon; (*  pick potential replacement candidates *)
    repeat
      mating; (*  selection, crossover, mutation  *)
    until [ enough mates ]
  end;


procedure mating;
  begin
    select; (*  2 mates *)
    crossover; (*  crossover and mutation *)
    replacement; (*  the new children *)
  end;


procedure environment; (*  inertial object version *)
  begin
```

```
    evalkeyboard; (*  keyboard commands and evaluation *)
    model; (*  inertial object model calculations - f=ma *)
    evaluation; (*  automatic evaluation procedure at
                    specified intervals *)
    criteval; (*  criterion counting mechanism *)
    display; (*  screen display update *)
  end;
```

Pipeline Operations Environment

The pipeline LCS differs from the inertial object LCS in the environment installed. The following is a description of those differences.

```
procedure environment; (*  pipeline operations version *)
  begin
    modelhandler; (*  seasons, leaks, time of day, year  *)
    evalkeyboard; (*  keyboard commands and evaluation *)
    loadclac; (*  load calculation *)
    pipe1; (*  pipe model calculations *)
    evaluation; (*  automatic evaluation procedure at
                    specified intervals *)
    display; (*  update display *)
  end;
```

REFERENCES

REFERENCES

1. Holland, J. H., Adaptation in Natural and Artficial Systems, University of Michigan Press, Ann Arbor, 1975.

2. Lafferty, H. B., "Function of the Gas Dispatching Department," Petroleum Engineer, vol. 30, no. 12, pp. D34-35,38-39, November, 1958.

3. Ebdon, J. F., "Digital Computers-Their Application to the Gas Industry's Problems," Gas, vol. 31, no. 1, pp. 109-115, January, 1955.

4. _____, "Personnel and Organizational Requirements for Engineering Applications of Digital Computers," Gas, vol. 31, no. 3, pp. 43-49, March, 1955.

5. _____, "The Second Gas Computer Symposium," Gas, vol. 31, no. 5, pp. 44-50, May, 1955.

6. _____, "Digital Computers Part 3. Where Gas Companies Use Them; Storage System Developments," Gas, vol. 31, no. 7, pp. 68-72, July, 1955.

7. _____, "The Place of Computers in Gas Piping Network Analysis," Gas, vol. 31, no.9, pp. 40-45, September, 1955.

8. _____, "Electronic Computers in Gas Industry Technology," Gas, vol. 31, no. 11, pp. 51-57, November, 1955.

9. Yonker, T., "Computer Control of a Large Gas Pipeline System," Pipeline and Gas J., vol. 201, no. 10, pp. 26-27, 72, August, 1974.

10. Kloer, F. H., "Automation Helps Meet Growing Demands of Pipeline System," Oil and Gas J., vol. 76, no. 24, pp. 62, 67-70, June 12, 1978.

11. Turner, E. B., "Computer Based Supervisory Control Systems," IEEE Trans. on Ind. Applications, vol. IA-10, no. 2, pp. 305-315, March-April, 1974.

12. Wilson, G. C., "Unattended Terminal Station is Controlled Remotely by Telemetering," Instruments, vol. 26, no. 1, pp. 127-129, January, 1953.

13. Orlofsky, S., "First Pushbutton Gas Pipeline Successful," Oil and Gas J., vol. 56, no. 28, pp. 114-119, July 14, 1958.

14. Armstrong, T., "Computer Monitors Gas Flow," Oil and Gas J., vol. 63, no. 51, pp. 64-67, December 20, 1965.

15. Pai, M. A. and R. A. Mugele, "Rapid Solution of Gas Pipeline Flow Problems-GASFLOW," ISA Trans., vol. 3, no. 2, pp. 149-157, April, 1964.

16. Heath, M. J. and J. C. Blunt, "Dynamic Simulation Applied to the Design and Control of a Pipeline Network," Gas Council Research Communication, GC149, pp. 1-14, November, 1968.

17. Rachford, H. H., "Care Can Reduce Gas Pipeline Use," Oil and Gas J., vol. 71, no. 29, pp. 93-96, July 16, 1973.

18. Covington, M. T., "Transient Models Permit Quick Leak Identification," Pipe Line Industry, vol. 49, no. 8, pp. 71-73, August, 1979.

19. Wylie, E. B. and V. L. Streeter, Fluid Transients, FEB Press, Ann Arbor, 1983.

20. Kroegner, C. V., "Role of the Computer in Distribution Design," American Gas J., vol. 182, no. 3, pp. 14-16, 34, March, 1955.

21. Nelson, J. M. and J. E. Powers, "Unsteady State Flow of Natural Gas in Long Pipelines," Oil and Gas J., vol. 56, no. 26, pp. 86, 88-89, 91, June 30, 1958.

22. Taylor, T. D., N. E. Wood and J. E. Powers, "A Computer Simulation of Gas Flow in Long Pipes," Soc. Pet. Eng. J., vol. 2, no. 4, pp. 297-302, December, 1962.

23. Wilkinson, J. F., D. V. Holliday, E. H. Batey and K. W. Hannah, Transient Flow in Natural Gas Transmission Systems, American Gas Association, New York, January, 1965.

24. Guy, J. J., "Computation of Unsteady Gas Flow in Pipe Networks," in Proc. of a Symposium on Efficient Computer Methods for the Practising Chemical Engineer, pp. 139-145, Institution of Chemical Engineers, London, 1967.

25. Stoner, M. A., "Analysis and Control of Unsteady Flows in Natural Gas Piping Systems," Ph.D. dissertation (Civil Engrg.), University of Michigan, Ann Arbor, 1968.

26. Yow, W., "Analysis and Control of Transient Flow in Natural Gas Piping Systems," Ph.D. dissertation (Civil Engrg.), University of Michigan, Ann Arbor, 1971.

27. Wylie, E. B., V. L. Streeter and M. A. Stoner, "Unsteady Natural Gas Calculations in Complex Piping Systems," Soc. of Pet. Eng. J., vol. 14, no. 1, pp. 35-43, February, 1974.

28. Rachford, H. H. and T. Dupont, "A Fast Highly Accurate Means of Modeling Transient Flow in Gas Pipeline Systems by Variational Methods," Soc. Pet. Eng. J., vol. 14, no. 2, pp. 165-178, April, 1974.

29. Wong, P. J. and R. E. Larson, "Optimization of Natural Gas Pipeline Systems via Dynamic Programming," IEEE Trans. Auto. Control, vol. AC-13, no. 5, pp. 475-481, October, 1968.

30. Larson, R. E., T. L. Humphrey and P. J. Wong, "Short-Term Optimization of a Single Pipeline," SRI Project 5975-Interim Report, Stanford Research Institute, Menlo Park, February, 1967.

31. Ade, C. W., "Optimal Management of a Natural Gas Transmission System," Sc.D. dissertation (Chem. Engrg.), Washington University, St. Louis, 1969.

32. Sood, A. K., G. L. Funk and A. C. Delmastro, "Dynamic Optimization of a Natural Gas Pipeline Using a Gradient Search Technique," Int. J. Control, vol. 14, no. 6, pp. 1149-1157, November-December, 1971.

33. Wienecke, D. R., "Computerized Optimization of Dispatching on a Gas Pipeline System," ASME 72-Pet-16, presented at the Petroleum Mechanical Engineering and Pressure Vessels and Piping Conf., New Orleans, September, 1972.

34. Flanigan, O., "Constrained Derivatives in Natural Gas Pipeline System Optimization," Soc. Pet. Eng. 3621, presented at 46th annual Fall meeting, New Orleans, October, 1971.

35. Rothfarb, B., et al., "Optimal Design of Offshore Natural Gas Pipeline Systems," Operations Research, vol. 18, no. 6, pp. 992-1020, November-December, 1970.

36. Bhaskaran, S. and F. J. M. Salzborn, "Optimal Design of Gas Pipeline Networks," J. of the Operational Res. Soc., vol. 30, no. 12, pp. 1047-1060, December, 1979.

37. Edgar, T. F., D. M. Himmelblau, and T. C. Bickel, "Optimal Design of Gas Transmission Networks," Soc. Pet. Eng. J., vol. 18, no. 2, pp. 96-104, April, 1978.

38. Larson, R. E. and D. A. Wismer, "Hierarchical Control of Transient Flow in Natural Gas Pipeline Networks," in Proc. of the IFAC Symposium on the Control of Distributed Parameter Systems, paper 6-1, 1971.

39. Holland, J. H., "Nonlinear Environments Permitting Efficient Adaptation," in Computer and Information Sciences II, Tou, J. T. (ed.), pp. 147-164, Academic Press, New York, 1967.

40. _____, "A New Kind of Turnpike Theorem," Bull. Am. Math. Soc., vol. 75, pp. 1311-1317, November, 1969.

41. _____, "Adaptive Plans Optimal for Payoff-Only Environments," Proc. of the 2nd Hawaii International Conference on System Sciences, pp. 917-920, Western Periodicals, 1969.

42. _____, "Genetic Algorithms and the Optimal Allocation of Trials," SIAM J. Computing, vol. 2, no. 2, pp. 88-105, June, 1973.

43. _____, "Adaptation," in Progress in Theoretical Biology, vol. 4, Rosen, R. and F. M. Snell (eds.), pp. 263-293, Academic Press, New York, 1976.

44. Bosworth, J., N. Foo and B. P. Zeigler, "Comparison of Genetic Algorithms with Conjugate Gradient Methods," NASA Contractor Report CR-2093, Langley Research Center, NASA, August, 1972.

45. Foo, N. Y. and J. L. Bosworth, "Algebraic, Geometric and Stochastic Aspects of Genetic Operators," NASA Contractor Report CR-2099, Langley Research Center, NASA, August, 1972.

46. Burks, A. W., et al., "Biologically Motivated Automaton Theory and Automaton Motivated Biological Research," in Proc. of the 1974 Conf. on Biologically Motivated Automata Theory, pp. 1-12, 1974.

47. De Jong, K., "Adaptive System Design: A Genetic Approach," IEEE Trans. on Systems, Man, and Cybernetics, vol. SMC-10, no. 9, pp. 566-574, September, 1980.

48. Bagley, J. D., "The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms," Ph. D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1967.

49. Cavicchio, D. J., "Adaptive Search Using Simulated Evolution," Ph.D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1970.

50. Frantz, D. R., "Non-linearities in Genetic Adaptive Search," Ph.D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1971.

51. Hollstein, R. B., "Artificial Genetic Adaptation in Computer Control Systems," Ph. D. Dissertation (C. I. C. E.), University of Michigan, Ann Arbor, 1971.

52. Martin, N., "Convergence Properties of a Class of Probabilistic Adaptive Schemes Called Sequential Reproductive Plans," Ph.D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1973.

53. De Jong, K. A., "Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. Dissertation (C. C. S.), University of Michigan, 1975.

54. Bethke, A. D., "Genetic Algorithms as Function Optimizers," Ph.D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1981.

55. Bellman, R., Adaptive Control Processes: A Guided Tour, Princeton Universty Press, Princeton, 1961.

56. Saridis, G. N., Self-Organizing Control of Stochastic Systems, Marcel Dekker, New York, 1977.

57. Beightler, C. S., D. T. Phillips and D. J. Wilde, Foundations of Optimization, Prentice-Hall, Englewood Cliffs, 1979.

58. Holland, J. H., "Adaptive Knowledge Acquisition," unpublished research proposal to the National Science Foundation, 1980.

59. Fiacco, A. V. and G. P. McCormick, Nonlinear Programming: Sequential Unconstrained Minimization Techniques, Wiley, New York, 1968.

60. Holland, J. H., "Outline for a Logical Theory of Adaptive Systems," J. Assoc. Computing Machinery, vol. 3, pp. 297-314, July, 1962.

61. _____, "Processing and Processors for Schemata," in Associative Information Techniques, Jacks, E. L. (ed.), pp. 127-146, American Elsevier, New York, 1971.

62. _____ and J. S. Reitman, "Cognitive Systems Based on Adaptive Algorithms," in Pattern-Directed Inference Systems, Waterman, D. A. and F. Hayes-Roth (eds.), pp.313-329, Academic Press, New York, 1978.

63. _____, "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge-Bases,"

Int. J. Policy Analysis and Information Systems, vol. 4, 1980.

64. _____, "Genetic Algorithms and Adaptation," Tech. Report #34, Cognitive Science Series, Univ. of Michigan and Univ. of Chicago, December, 1981.

65. Booker, L. B., "Intelligent Behavior as an Adaptation to the Task Environment," Ph.D. dissertation (C. C. S.), University of Michigan, Ann Arbor, 1982.

66. Wilson, S., "Adaptive 'Cortical' Pattern Recognition," unpublished manuscript, Rowland Institute for Science, Cambridge, MA, 1983.

67. Minsky, M. L., Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, 1967.

68. Davis, R. and J. King, "An Overview of Production Systems," in Machine Intelligence 8, Elcock, E. W. and D. Michie (eds.), pp. 300-332, Wiley, New York, 1976.

69. Anderson, J. R., Language, Memory and Thought, Lawrence Erlbaum Associates, Hillsdale, N. J., 1976.

70. Buchanan, B., G. Sutherland and E. A. Feigenbaum, "HEURISTIC DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in Machine Intelligence 4, Meltzer, B., D. Michie and M. Swann (eds.), pp. 209-254, American Elsevier, New York, 1969.

71. Davis, R., B. Buchanan and E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence, vol. 8, pp. 15-45, February, 1977.

72. Duda, R., J. Gaschnig and P. Hart, "Model Design in the Prospector Consultant System for Mineral Exploration," in Expert Systems in the Microelectronic Age, Michie, D. (ed.), pp. 153-167, Edinburgh University Press, Edinburgh, 1979.

73. Hollander, C. R., et al., "The Drilling Advisor," in Proc. Trends and Applications 1983 Automating Intelligent Behavior Applications and Frontiers, pp. 28-32, IEEE Computer Society Press, Silver Spring, 1983.

74. Feigenbaum, E. A. and P. McCorduck, The Fifth Generation, Addison-Wesley, Reading, 1983.

75. DeGroot, M. H., Optimal Statistical Decisions, McGraw-Hill, New York, 1970.

76. Takahashi, Y., M. J. Rabins and D. M. Auslander, Control and Dynamic Systems, Addison-Wesley, Reading, 1970.

77. Tsypkin, Ya. Z., Foundations of the Theory of Learning Systems, translated by Z. J. Nikolic, Academic Press, New York, 1973.

78. Boyer, H. M. "Application of Cluster Analysis in Determining Transient Flow Loading Patterns," Pipeline Simulation Interest Group, October, 1975.