Dynamic Scale Genetic Algorithm: An Enhanced Genetic Search for Discrete Optimization

Bela Dange Joshi
Old Dominion University

DYNAMIC SCALE GENETIC ALGORITHM:

AN ENHANCED GENETIC SEARCH FOR DISCRETE OPTIMIZATION

by

Bela Dange Joshi
M.S. December 1992, University of Tennessee

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

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY
December 1996

Approved by:

Resit Unal (Director)

Laurence D. Richards (Member)

Billie M. Reed (Member)

Mark Fleischer (Member)

James Schwing (Member)

ABSTRACT

DYNAMIC SCALE GENETIC ALGORITHM: AN ENHANCED GENETIC SEARCH FOR DISCRETE OPTIMIZATION

Bela Dange Joshi
Old Dominion University, 1996
Director: Dr. Resit Unal

The minimization of operations and support resources of reusable launch vehicles is a complex task, involving discrete optimization and the simulation domain. Genetic algorithms, offering a robust search strategy suitable for integer variables and the simulation domain, can be applied to minimize these resources. This research developed an enhanced genetic algorithm for problems with a linear objective function, the most common class of discrete optimization problems. The dynamic scale genetic algorithm developed here incorporates concepts of implicit enumeration to enhance search. This is achieved by utilizing problem specific information to refine the solution space over successive generations. The utility of the proposed algorithm was demonstrated by comparing its performance, in terms of quality of solutions produced, to that of the simple genetic algorithm. For all test problems, the dynamic scale genetic algorithm consistently produced *better* solutions in *fewer* generations. The proposed algorithm was successfully applied to optimize the operation and support resources of reusable launch vehicles, through a discrete event simulation model. The least cost solution so obtained represents

an improvement over both the simple genetic algorithm, and the previous manual approach of minimizing operation and support resources.

To my first teachers, my parents

ACKNOWLEDGMENTS

I express sincere appreciation and gratitude to Dr. Resit Unal, whose unswerving enthusiasm, patience, and guidance helped me throughout the research and graduate program. I am grateful that he supported me as a research assistant while working on this project. I also thank Dr. Laurence Richards, Dr. Billie Reed, and Dr. Mark Fleischer for their interest and insightful suggestions. Sincere thanks to Dr. James Schwing for helpful comments.

This research was sponsored by NASA Langley Research Center. I would like to especially thank Doug Morris and Nancy White of the Vehicle Analysis Branch, NASA, for the support they extended during this research. All computations were performed on the SUN SPARC® Station of the Vehicle Analysis Branch.

Finally, I thank my husband Ravi Joshi for making the dream possible. His undying faith, patience and belief through the course of my graduate program, were a tremendous source of inspiration and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

BACKGROUND AND MOTIVATION

1.1 Introduction

The design of complex systems, such as aerospace systems, has historically emphasized the performance requirements aspect. However, global competition and declining budgets of recent years has prompted the need for developing economically competitive systems, without compromising the design objectives of quality, produceability, operability and supportability (Unal et al. 1990). This can only be achieved by a thorough understanding and inception of life cycle economic impacts in the early design phase. Studies show that Operations and Support (O&S) activities can account for 60% to 80% of life cycle costs of reusable space systems (Griffin 1988, Fabrycky and Blanchard 1991). Therefore, in order to design affordable spacecraft and minimize life cycle costs, it is essential to study and optimize the operation and support resources and activities early in the design phase itself.

The interactions between the various operation and support activities of a complex system such as a reusable launch vehicle, are also complex. There is uncertainty due to the stochastic nature of failure rates and maintenance activities. Therefore, a closed form analytical formulation realistically describing the complex activities does not exist. Furthermore, operational data is generally not available in the early design stages. Due to these reasons, studying the operation and support considerations during conceptual design remains a difficult and challenging task. As a result, O&S models and optimization approaches for the early design phase of launch vehicles have been generally lacking in the literature.

Recently, the operation and support requirements of reusable launch vehicles have begun to be modeled and examined at Langley Research Center (LaRC) of the National Aeronautics and Space Administration (NASA), by employing stochastic discrete event simulation (Morris et al. 1983, 1995; Ebeling and Donohoe 1994). Simulation models can be thought of as theoretical mechanisms or functions that translate feasible input parameter sets into probabilistic output performance measures. Simulation provides an effective means of studying complex, non-linear systems characterized by random processes, which cannot be described analytically, or whose explicit form is unknown. Simulation is therefore an efficient and cost effective tool, for studying the impact of various parameters on the system response, without having to actually build the system and perform expensive experiments. The O&S discrete event models used at NASA Langley Research Center, simulate the mission and the pre- and post- flight maintenance activities of a fleet of vehicles in a particular space program (Morris et al. 1983, 1995; Ebeling and Donohoe 1994). The simulation can be run for as many alternative designs as desired, to gain an insight into and obtain projections of the maintenance resource requirements for a proposed space program. These estimates can then be used to compare the acquisition and O&S costs for various alternate conceptual designs.

In order to effectively compare candidate designs, one has to ensure that the overall maintenance resources are minimized for a particular O&S scenario and space program.

This O&S problem, involving the determination of the minimum resources to meet a given mission rate constitutes an optimization problem. The decision variables for the above optimization problem, are restricted to be integer values, within a certain "user-specified" range. Examples of the integer input variables include: launch vehicles, launch pads, facility bays, scheduled and unscheduled maintenance crew. These decision variables are usually combined in the form of a linear additive cost function. The constraints, specified in terms of the performance measures, such as meeting a given mission rate in a timely manner, are non-linear and evaluated through stochastic simulation.

Such operation and support studies can be viewed as simulation optimization problems, characterized by integer decision variables, a linear objective function and non-linear constraints. Simulation optimization problems are known to be difficult to solve (Jacobson and Schruben 1989). Although the particular case of minimizing operation and support resources of space vehicles has been considered here, problems belonging to this general class commonly occur in industry. Constrained optimization problems involving integer variables and a linear objective function, widely occur in the management and efficient use of scarce resources to increase productivity (Nemhauser and Wolsey 1988, Parker and Rardin 1988). For example, the transportation industry, such as the airlines and the car rental agencies, face similar problems. The airline industry needs to maintain a fleet of airplanes in order to meet a specific schedule of flights. Similarly, a car rental agency maintains a fleet of cars to meet customer demand. The primary objective for the airline or the car rental agency, as in any industry, is to make money, which can be achieved through minimum acquisition cost of equipment (planes and cars) and efficient allocation of operational resources, so as to minimize the overall operating expenses.

1.2 The Operations and Support Problem

Two distinguishing features characterize the present launch vehicle operations and support optimization problem. First, it deals with determining the optimal levels of the deterministic input parameters which minimize the required resources of the overall system, subject to constraints and performance criteria that are computed through stochastic simulation. Simulation becomes necessary since constraint violations and measures of the performance cannot be obtained or predicted through closed form analytical formulations. This is often a result of the non-deterministic nature of the system under study. For instance, an O&S model simulates the processes of component and system failure, repair and replacement times, and maintenance delays. These underlying processes are non-deterministic in nature and hence the simulation model itself and its outputs are stochastic. Such simulation optimization problems are traditionally solved by techniques borrowed from nonlinear programming. However, these techniques, such as gradient estimation and pattern search, originally developed for deterministic optimization, are in many cases impractical for the computationally intensive non-deterministic simulation domain.

Secondly, the problem involves integer variables, and hence like other discrete domain problems is difficult to solve. Optimization problems with integer variables, such as integer programming problems, are in the class of so called *NP-hard* problems

(Papadimitriou and Steiglitz 1982, Garey and Johnson 1983). It is conjectured that such problems cannot be exactly solved by polynomial-time algorithms, i.e., algorithms that are guaranteed to terminate in a finite time. For such problems one is generally willing to settle for less ambitious goals, such as an improved or near-global solution rather than a globally optimum solution.

1.3 Simulation Optimization in the Discrete Domain

Simulation models enable one to observe the effect of a set of input parameters (such as the maintenance crew size), on the output parameters (such as the mission rate). However, simulation models in general do not provide a way to directly minimize (maximize) the input parameters. To achieve this, the following steps are usually carried out:

1. The simulation is run at a particular set of input parameter levels,
2. The output results so obtained are analyzed, and
3. The input parameters are modified in accordance with an existing simulation optimization scheme to obtain a desired change in the output parameters.

The above steps have to be repeated until either the optimal value is reached or some stopping criteria is met.

It is evident from a study of the literature that solving an optimization problem through a simulation involving integer variables is difficult -- in terms of the quality of the solution and the computational burden (Fu 1994). The optimization methods that are traditionally used for the discrete domain are the pattern search methods, statistical methods, complete enumeration and the random method. Pattern search methods are local optimum seeking approaches that start with an initial randomly selected point in the solution space. The search proceeds by applying suitable transformations to the initial point to other solutions in the problem landscape.

For instance, the pattern search method of Hooke and Jeeves starts at an initial base point and increments the input variables, by a fixed value, one at a time, if doing so improves the solution (Hooke and Jeeves 1961, Friedman and Savage 1972). Input variables are incremented in this manner until no more improvement in the solution is obtained. Next, the incremental values are decreased and the entire process is repeated, starting from the last point reached. The search terminates when a pre-determined incremental value is reached. Another pattern search method, Nelder and Mead's simplex search, similarly starts with an initial set of factor settings (Nelder and Mead 1965, Spendley et al. 1962). In case of maximization, it successively replaces the factor with the least value with the centroid of all current factor settings. The procedure is repeated until no more improvement is possible. It is therefore evident that the performance of these search methods is extremely sensitive to the initial point chosen. Furthermore, the pattern search strategies are local-optimum seeking techniques. Consequently, in the simulation domain, typically characterized by a vast solution space of unknown topology, there is a risk of sub-optimization. (Tabu search is another promising technique applicable for discrete optimization, that combines local search with other more advanced search mechanisms (Glover 1989). However, its applicability to the simulation domain remains to be explored.)

When the solution space is finite, statistical methods such as multiple comparison, and ranking and selection may be used. Multiple comparison uses certain pair-wise comparisons to make inferences in the form of confidence intervals. Ranking and selection specifies some criterion, such as choosing the best alternative with some pre-specified confidence level. The procedure selects a combination that guarantees with a user-specified probability, that the response will be within a certain range of the optimal value. Although these statistical methods yield the global optimum, their applicability is limited to problems with a very small solution space due to their high computation burden.

As the name suggests, complete enumeration performs an exhaustive search of the entire solution space and yields the global optimum (Farrell 1977). This method is computationally very intensive, and once again, applicability is severely restricted to small search spaces. The random method evaluates random points in the search space and terminates when a pre-specified number of evaluations are reached (Smith 1973).

Recent years have seen the emergence of directed random searches such as genetic algorithms and simulated annealing. Genetic algorithms (GAs) have been proven, both theoretically and empirically, to provide a robust search in complex spaces (Holland 1975, Goldberg 1989). Genetic algorithms do not impose constraints such as continuity and differentiability and hence can be used in the integer valued discrete domain. They have shown promise as simulation optimizers in preliminary studies (Yunker and Tew 1994, Elketroussi and Fan 1994, Tompkins and Azadivar 1995). Inspired by natural selection and genetics, a genetic algorithm uses crossover and mutation to form a generation of candidate solutions from an initial randomly selected population. Candidates with above average fitness are mated to produce offspring in the successive generations. In this manner, new candidates with improving fitness are formed.

A summary of the various optimization approaches available in the literature for simulation studies involving integer variables is provided in Table 1.1 below.

Technique	Domain	Optimum	Solution space
Pattern Search			
Hooke-Jeeves	C,D	L	B,S
Simplex, Constrained Simplex	C,D	L	B,S
Statistical Methods			
Multiple Comparison Approach	D	G	S
Ranking and Selection	D	G	S
Complete Enumeration	D	G	S
Random Method	D	L	S
Genetic Algorithm	C,D	L	B,S

Key: **C** Continuous **D** Discrete **L** Local **G** Global
B Big **S** Small

Table 1.1 Simulation optimization techniques applicable for the discrete domain

The literature review indicates that the approaches guaranteeing the global optimum, such as the statistical methods and the complete enumeration, are too computationally intensive to be practically useful. Of the practical approaches, such as pattern search, random search and the genetic search, the genetic algorithm seems a promising heuristic. Unlike pattern search, genetic algorithms explore several areas of the solution space simultaneously, and do not terminate upon finding a local optimum.

1.4 Purpose of the Study

Thus, of the optimization techniques that can be implemented in practical situations, the search strategy of the genetic algorithm appears to be more effective compared to the pattern search approaches. The simple genetic algorithm utilizes the operations of random crossover of genetic material and mutation to obtain solutions with above average fitness. However, in doing so, it ignores any information that might be contained in the problem under consideration. In this research it is hypothesized that combining problem-specific information into the genetic search, by intelligently pruning the search space, makes for a more efficient search strategy.

Specifically, the research aims to improve the performance of the genetic algorithm for constrained optimization problems involving integer variables and a linear objective function. A conventional genetic algorithm maintains a set of candidate strings representing solutions in the search space. The search space itself, and the mapping of the candidate strings to the search space, is defined at run-time by the user and is unchanging over the run of the genetic algorithm. Thus, even when the population has converged to an optimal or sub-optimal region, the entire original search space defined by the user at run-time is used to map the strings. In this research, it is hypothesized that employing a fixed search space reduces the effectiveness with which the genetic algorithm finds a solution. It is hypothesized that by suitably shrinking or refining the original solution space, and hence changing the mapping through successive generations, the performance of the genetic algorithm can be improved. The proposed dynamic scale genetic algorithm (DyScGA) utilizes problem-specific information to successively refine the search space.

There is also a built-in memory feature, which retains the boundaries of the refined search space from one application of the algorithm to the other for a particular problem. Due to the memory, consequent searches are started from solution spaces that have been previously refined by the proposed modified algorithm. This feature enables a more effective exploration of the portion of the original solution space that is most likely to contain the optimum. Furthermore, it also reduces the computational requirements to perform the exploration from one application to another.

It is therefore hypothesized that the DyScGA improves the performance of the simple genetic algorithm, for discrete optimization with a linear objective, twofold: it produces *better solutions* with *lower computational requirements*. This is a contribution to the genetic algorithm literature. Most of the search-space-refining improvements suggested for GAs in the literature (such as dynamic parameter encoding, delta coding and adaptive representation genetic optimizer), employ population convergence measurements and do not have a 'memory'. Genetic algorithms with learning that have been suggested in the past (such as classifier systems and GAs combined with expert systems) employ rule-based systems. However, these are limited in that they either do not preserve memory, or are restricted to machine learning and cannot be used for optimization. A multi-leveled environment for learning that preserves memory and can be used for optimization has been proposed earlier (Nutter and Ding 1992). However, this expert system based genetic search is extremely complicated and consists of ten different modules consisting of three levels of representation, two transformations and three levels of learning.

In contrast, the dynamic scale genetic algorithm proposed in this research is very simple to implement. It consists of an add-on module that contains code to dynamically assign boundaries of the solution space, by exploiting the information provided by the current best objective function. It does not need any additional parameters to be set by the user at run-time. The enhanced search strategy proposed here is applicable for constrained optimization problems with a linear objective function and discrete integer valued variables. There is no restriction on the constraints, which can be linear or non-linear. This genetic algorithm modified for discrete optimization can be applied to simulation as well as non-simulation situations. On a practical level, this research also contributes to the life cycle cost analysis of launch vehicles. The operations model can be integrated with other disciplinary models to achieve a systems level design optimization for a reusable launch vehicle.

1.5 Contribution

This research makes a contribution to the genetic algorithm literature by proposing an enhanced genetic search for discrete optimization. Specifically, the dynamic scale genetic algorithm differs from earlier search-space-refining modifications in the following manner:

1. It uses problem specific information and not population convergence to refine the search space.
2. The user does not have to set any additional control parameters during run-time.
3. The search space is refined if and only if mathematical evidence indicates that the discarded portion does not contain the optimum.
4. Refining the search space in this manner negates the necessity for an 'inverse' pruning operator to recover the discarded portion that is employed by the other modified genetic search strategies.
5. It retains the new boundaries of the refined search space over subsequent applications of the genetic algorithm. This provides for a memory feature that can significantly improve the performance of the DyScGA from one run to another.

The research also indirectly contributes to the life cycle cost analysis of launch vehicles. An O&S model in conjunction with the optimization methodology developed here can be integrated with other disciplinary models. By enabling the consideration of operations and support costs early in the design phase, a total life cycle cost approach to design can be used.

1.6 Outline

The general outline of this dissertation is as follows. A review of literature devoted to the relevant improvements and enhancements made to the basic genetic algorithm is summarized in Chapter 2. The foundations of the proposed dynamic scale genetic algorithm are then described in Chapter 3. In Chapter 4, the DyScGA is tested and validated. The DyScGA is used in conjunction with a NASA LaRC discrete event simulation model to optimize the operations and support resources for reusable launch vehicles. Conclusions based on the results obtained and suggestions for future research are outlined in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

This chapter provides a brief description of the basic genetic algorithm. It also contains a summary of the relevant enhancements and modifications to the genetic algorithm, as proposed in the literature.

2.1 The Genetic Algorithm

Genetic Algorithms (GAs), first introduced by Holland (1975), are stochastic search algorithms inspired by the mechanics of natural selection and natural genetics. GAs combine the principle of *survival of the fittest* among string structures with a structured yet randomized information exchange. GAs have been used primarily in the fields of search, optimization and machine learning. Traditional calculus based methods and mathematical programming techniques impose constraints on the search space, such as continuity, convexity and differentiability. In practical situations with large unknown solution spaces, these local optimum search methods are susceptible to getting trapped in a local minimum (Goldberg 1989). Biased random search algorithms, such as genetic algorithms and simulated annealing, have gained popularity as researchers have recognized the shortcomings of the traditional optimization techniques. GAs are attractive in application due to the following reasons:

1. Robust over a broad spectrum of problems.
2. Require no auxiliary information such as derivatives, as they use the performance metric itself to guide towards better and better solutions.
3. Easy to implement GAs and to interface to simulation and other models.

A simple genetic algorithm maintains a population consisting of candidate solutions, made of binary strings representing the parameters of the optimization problem. Strings with above average fitness are selected to form a mating pool. Two such strings are randomly mated to produce offspring by exchanging parts of their binary string. The mutation operator acts on the offspring by flipping a bit from 0 to 1 and vice versa with a certain probability. The mutation feature inserts diversity into the current population and helps the genetic algorithm escape from local optima. The reproduction and mutation cycle is repeated until a desired termination criterion is reached (for example, a predefined number of generations are processed). Figure 2.1 depicts a graphical schematic of the simple genetic algorithm.

Encoding Mechanism

Fundamental to the GA structure is the encoding mechanism for representing the variables of the optimization problem. The encoding mechanism depends upon the nature of the problem variables. Integer variables are encoded using a fixed number of binary bits within a user-specified range. In the case of continuous variables, each variable is first linearly mapped to an integer defined in the specified range, and then encoded using binary bits.

Fitness Function

Each population member has an associated fitness function which represents a solution to the optimization problem. To maintain uniformity over various problem domains, a fitness function that normalizes the objective function of a problem between 0 and 1, is used. For example, the objective function x^{10} where x is coded with 30 bits is normalized as $\left(\frac{x}{2^{30}-1}\right)^{10}$. The normalized value of the objective function is the *fitness* of the string, which the selection mechanism uses to evaluate members of the population.

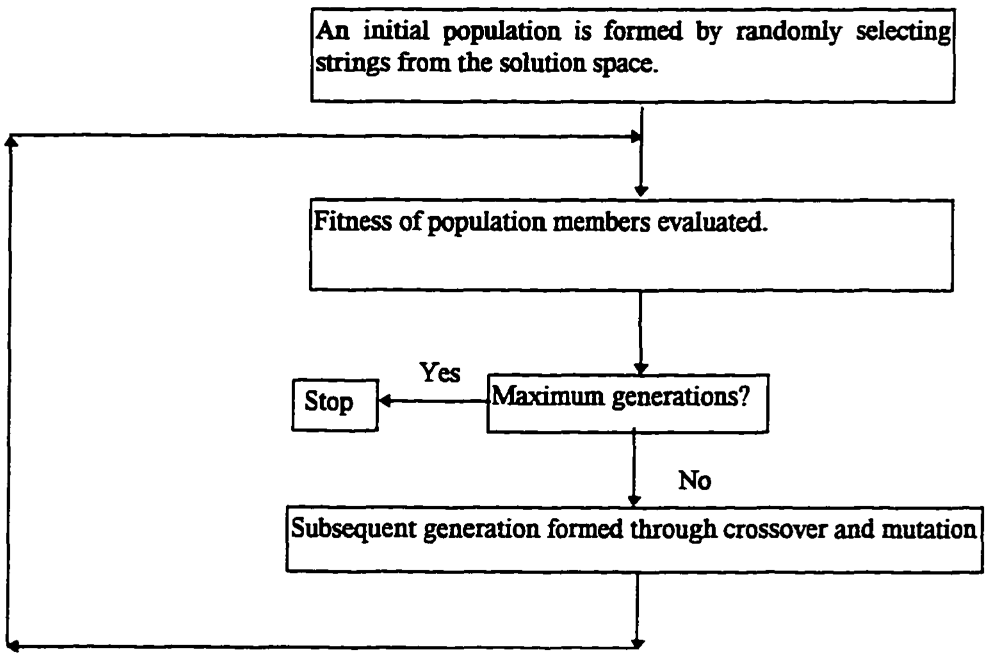

An initial population is formed by randomly selecting strings from the solution space.
Fitness of population members evaluated.
Maximum generations?
Yes
Stop
No
Subsequent generation formed through crossover and mutation

Figure 2.1 Flowchart: Simple Genetic Algorithm

Selection Schemes

A selection scheme chooses the members of the population that will reproduce. A number of different selection schemes have been proposed over the years. In a simple genetic algorithm, a string with higher fitness function receives a higher number of offspring and has a higher chance of surviving in subsequent generations.

Crossover

Pairs of strings are picked from the population based on the selection scheme being used, to be subjected to a single-point crossover. Assuming l is the string length, it randomly chooses a crossover point greater than 1 and less than l. An offspring is created by the portion of the first string up to the crossover point and the portion of the second string after the crossover point. After choosing a pair of strings the genetic algorithm invokes crossover only if a randomly generated number in the range 0 and 1 is greater than p_c, the crossover probability. The crossover probability influences the outcome of the genetic algorithm and is generally selected by the user.

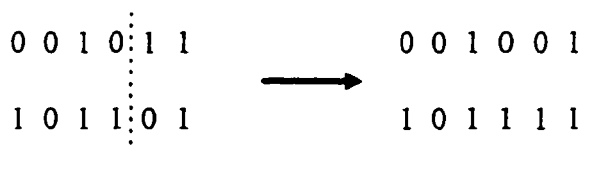


Figure 2.2 Crossover

Mutation

After crossover, strings are subjected to mutation. Mutation of a bit involves flipping it, i.e. changing a 0 to 1 and vice versa. The mutation rate p_m controls the probability that a bit will be flipped, and is set by the user. The bits of a string are mutated independently of one another. Usually, the mutation rate is set to a small value, to avoid excessive mutation. Mutation provides an effective mechanism for introducing diversity into the genetic pool, exploring new regions of the problem landscape, and escaping local optima.

2.2 Review of Literature

Over the past decade, several modifications and enhancements have been proposed to the simple genetic algorithm with a view to improving its performance. The following sections contain suggested modifications that are pertinent to GA learning and optimization. Accordingly, the literature review is divided into three sections: GA optimization without memory, GA learning without optimization, and GA optimization with memory. (Learning implies knowledge acquired over successive generations of a *single* application of a GA. Memory implies retention of this acquired knowledge across *successive* applications of the GA.)

2.2.1 GA optimization without memory

Several improvements have been suggested in the literature to improve the performance of genetic algorithms for function optimization. It was evident from the literature, that most of such GA modifications have focused on continuous variables. Encoding real valued parameters onto a discrete domain consisting of binary string representations is usually a complex task. Typically, the higher the number of bits utilized to encode a parameter, the better is the resolution of the genetic search. However, as the number of bits in the encoding increases, the effectiveness of the genetic search suffers. Consequently, a large body of literature is devoted to improving the encoding and the corresponding mapping strategy and search space for continuous variables. This section describes these improved genetic search strategies.

Delta Coding

Delta coding employs a novel encoding structure to achieve efficient optimization (Whitley et al. 1991). In a simple genetic algorithm, a population member takes the form of binary strings representing the various parameters involved. In delta coding, the encoding represents a particular distance delta 'δ' away from some previous solution. The first run of the GA is like a conventional GA. However, subsequent runs are made by using the best solution obtained in the most recent run as a partial solution. The genetic algorithm is restarted with the substring coding for each parameter representing a distance δ away from the value of the corresponding 'best' parameter. The delta values represented by the encoding are added to the partial solution to evaluate fitness. Therefore, a neighborhood about the current best solution is explored. With each delta iteration, the number of bits used for encoding is typically reduced, and the solution space is made smaller. There is also provision for an inverse operator to increase the number of bits if required.

Delta coding preserves diversity in the population by having an entirely new and random population for each generation. Each individual iteration can be viewed as a single run of a genetic algorithm; with the only change being in the mapping strategy. Hence, the theoretical foundations of genetic algorithms still apply.

Delta coding results in an efficient optimization strategy. However, there is no memory or retention of learning across successive applications of the genetic algorithm. Furthermore, implementing delta coding requires additional effort by the user at run time. Reduction and expansion strategies have to be tested and incorporated. Also, the user needs to set additional parameters such as the smallest number of bits for a parameter, and the number of bits by which to reduce and expand each parameter during each new iteration. The performance of the delta coding strategy is greatly influenced by these strategies and parameters that the user selects, often in an ad-hoc manner.

Distributed Genetic Algorithm

A distributed genetic algorithm attempts to improve search and maintain diversity by using distributed populations (Whitley and Starkweather 1990). Small sub-populations represent an independent search except that the sub-populations exchange information by swapping copies of their best strings at fixed intervals. This provides for an effective exploration of the parameter space for optimization. The sub-population size, number of sub-populations, and number of strings exchanged are user defined. However, there is no memory or retention of learning within independent runs of the distributed genetic algorithm.

Adaptive Representation Genetic Optimizer

Adaptive Representation Genetic Optimizer Technique (ARGOT) 'learns' a strategy for solving a particular optimization problem (Shaefer 1987). Intermediate mappings are introduced between the strings representing candidate solutions and the search space. Several population based operators alter these intermediate mappings during the search. These operators are based on population measurements such as parameter convergence (uniformity of parameter or substring), parameter variance (spread of the parameter or substring distribution), and parameter positioning (relative average position of the parameter within a permissible range of parameter values). ARGOT uses these measurements to dynamically adjust parameter resolution by changing the number of bits, and to adjust the location of parameter boundaries.

If the parameter or substring population has converged (i.e., if a user-defined proportion of the population contains a fixed parameter or substring value), the resolution of a parameter is increased by adding bits to perform a finer search of the parameter space. If the parameter population has not converged, the resolution is decreased by reducing the number of bits, leading to a coarser evaluation of the search space.

As the parameter value approaches the moving boundary, it is shifted in an attempt to better center the parameter. (There is a rigid parameter boundary within which the moving boundaries must lie). Shifting the parameter boundary causes the search space to either contract or expand. The moving boundaries are 'dithered' or shifted by random small increments when the parameter has neither converged nor been completely randomly distributed. When the distribution of a parameter is narrow, the moving boundary interval is contracted. Similarly, when this distribution is wide the roving boundaries are expanded.

Besides these primary operators, there are several secondary operators. The Metropolis operator accepts a bit mutation based upon the change in fitness. A homotopy operator is switched on when a parameter has converged, and a local search to locate the solution within a promising region, is initiated.

Although ARGOT has shown good results for optimization purposes, the strategy and its implementation is very complex. Furthermore, threshold levels for triggering all the above operators, such as parameter convergence, need to be set by the user at run-time. It has been pointed out that it is difficult to establish these trigger threshold levels for each problem (Schraudolph and Belew 1992). The performance of the ARGOT strategy depends significantly on these settings. Additionally, ARGOT does not preserve the learning strategy for future applications within a problem domain, i.e. it does not have a memory feature.

Adaptive search space scaling algorithm

An adaptive search space scaling algorithm has been developed for medical image registration (Mandava et al. 1989). It searches a real-valued domain of transformations for the optimum transformation. Adaptive search space scaling dynamically estimates a sub-space to focus the investigation from the allowable search space. Distributions of the best solutions are used to contract and expand the sub-space in a manner similar to the ARGOT roving boundaries. A histogram of best solutions is formed after every generation. The user determines the maximum permissible number of best solutions to include in the histogram. A separate histogram is constructed for each parameter. The smallest number of consecutive bins in the histogram that contain 80% of the best structures are used to assign the new boundaries. By setting these boundaries to be larger than the theoretical assignment, a previously contracted sub-space can be expanded if future parameter values fall near the boundaries.

Adaptive search space scaling performs an effective search by zooming in on a sub-space most likely to contain the optimum. However, it requires the user to specify the number of best solutions for histogram generation during run-time, which has an impact on the performance of the algorithm. Furthermore, there is no memory or retention across successive applications of the genetic algorithm.

Dynamic Parameter Encoding

Dynamic parameter encoding (DPE) dynamically adjusts the accuracy of the encoded parameters to increase the resolution of the solution and to zoom in on the most promising area of the search space (Schraudolph and Belew 1992). It uses concepts similar to ARGOT. The heuristic DPE employed for triggering the zoom operator is based on populatic\n convergence. A histogram over the current search interval formed by the two most significant bits of the parameter is constructed. By summing over two neighboring quarters, population counts for the three overlapping target intervals are computed. The largest of these counts is used as a basis for indicating population convergence. When this convergence indicator exceeds a trigger threshold level, set by the user during run-time, the population is considered to have converged, and the zoom operator is invoked. The zoom operator restricts the GA search to target intervals.

DPE does not add bits to increase the parameter or search resolution. It keeps the number of bits constant. However, it drops the significant bits as the population converges and the search progresses. In the beginning of the search the binary string representation encodes only the most significant bits of the parameter, representing a coarse grain partitioning of the search space. As the genetic algorithm begins to converge, the most significant bit is recorded and dropped from the encoding, and a new bit is introduced. The new bit adds to the precision and creates a finer grain partitioning of the search space. While the number of bits remains constant, the optimization function is searched using an increasing level of detail.

Dynamic parameter encoding improves the optimization performance of the genetic algorithm. However, since DPE does not employ an inverse zoom operator, there exists a possibility, for highly complex landscapes, that the region of the search space that contains the optimal gets permanently discarded. Furthermore, the user is required to set a trigger threshold, which influences the performance of the algorithm. Finally, DPE does not provide for learning across successive applications of the GA or for knowledge retention.

2.2.2. GA learning without optimization

The genetic algorithm has also been modified in the literature to include learning aspects. This section describes enhancements to the simple genetic algorithm that provide for a learning feature.

Machine Learning

Classifier systems are a special class of rule-based systems (Holland 1986). A classifier system is a machine learning system that learns simple rules to guide its performance in an arbitrary environment. Knowledge is stored in "If-Then" rules. Each rule is associated with a real number representing a measure of its performance. Genetic algorithms explore the space of permissible rules and provide the learning algorithm in classifier systems. The genetic algorithm used in classifier systems is slightly different from the simple GA used for optimization, although it is still largely based on reproduction, crossover and mutation. New rules are created and placed in the population and processed to evaluate their role in the system. The knowledge in classifier systems is not retained across individual GA runs, thus jeopardizing their ability to improve their performance. Additionally, classifier systems cannot perform optimization, because instead of storing candidate solutions in the populations as required for optimization, they store 'If-Then' rules.

To overcome the lack of knowledge retention, Zhou (1990) developed a rule-based learning system called Classifier System with Memory (CSM). This kind of classifier system preserves problem solving expertise and tailors it to fit a new situation, so that learned knowledge can be transferred within a domain. However, since CSM is still fundamentally a classifier system, which stores 'If-Then' rules as individuals in the population, it cannot be used for optimization.

Strategy Acquisition Method Using Empirical Learning (SAMUEL) is based on the classifier system (Grefenstette et al. 1990), and learns expert system rules and control strategies with GAs. Unlike other machine learning systems, SAMUEL learns rules in a high-level language by adapting high-level genetic operators for that language from basic genetic algorithms. However, SAMUEL does not preserve memory across successive applications of the GA, and cannot be applied for optimization.

Expert Systems and Genetic Algorithms

Powell et al. (1989) developed an optimization system, EnGENEous, that combines an expert system and genetic algorithm, to exploit domain-specific knowledge for the design of aircraft turbine engines. In this approach, the rule base for the expert system is first created by the engineer. The expert system starts from a single design point specified by the engineer and uses selective rules from the rule base to achieve a desired change in the fitness function. Additional specialized control methods are built in to augment the rules provided by the expert system. The expert system continues to change input parameter levels in this fashion. Eventually, the expert system may get stuck in local minima or at constraint boundaries. When this occurs, the genetic algorithm is used to escape the minima or avoid constraints. The initial population is formed from promising design points already explored by the expert system and past designs. The solution found by the genetic algorithm is fed back to the expert system and the entire process is repeated until some stopping criterion is satisfied.

This hybrid model provides for a more efficient optimization procedure. However, it does not address knowledge retention across future applications of the genetic algorithm system.

2.2.3 GA optimization with memory

Nutter and Ding (1992) have combined expert system learning with GAs to achieve optimization and retain domain specific knowledge across successive applications. The specific application domain they have considered is a computer network system. The proposed Multi-Leveled Environment for Learning (MEL), acts as a bridge between the different data and knowledge representational formats required by the genetic algorithm (typically binary strings) and the expert system (typically 'If-Then' rules). MEL consists of ten modules operating on a layered knowledge base. Knowledge can be projected from one layer to another through the use of appropriate transformations. Three levels of representation and two transformations, along with three levels of learning are used. The low-level representation takes the form of binary strings that encode parameters to form candidate solutions for the optimization problem. Mid-level representations use an expert system to store information about background knowledge, interpretation of system parameters that comprise individuals and experiences, and generalization of hierarchies. High level representation consists of 'If-Then' types of rules. The two transformations help to convert knowledge from one representation to another. These are the low-to-middle-level transformation and the middle-to-high-level transformation. The three levels of learning include: low-level genetic search, mid-level accumulation and abstraction, and high-level inductive generalization. The low-level genetic search is provided by the GA. Mid-level learning comes from the analysis of past experiences. An individual's fitness is compared with the past history, and if it lies at the limits of or outside that range, it is stored permanently in the middle-level representation. These best and worst experiences accumulated in the mid-level are analyzed. The analysis is used to generate new 'If-Then' rules which provide the high-level learning.

Thus MEL, designed for application in the computer network system, provides for a learning mechanism and can be used for optimization. However, with its ten modules operating on layers, three levels of representation, two transformations and three levels of learning, it is extremely complex to build and implement. Furthermore, as it was developed for the computer network system, its applicability is restricted.

2.3 Summary of Literature Review Results

A number of enhancements have been proposed to the basic genetic algorithm over the past years, with a view to improving its performance. In this chapter some of those enhancements pertinent to the algorithm proposed in this research (i.e., involving learning and dynamic search space) have been reviewed. The following table summarizes the literature review presented in the preceding sections.

Name	*Improvement or modification*	*Optimization?*	*Memory ?*	*Comment*
Delta coding	Encodes parameters as δ distance	Yes	No	
Distributed GA	Sub-GAs performing searches	Yes	No	
ARGOT	Learns problem solving strategy based on population convergence	Yes	No	Complex
Adaptive scaling	Adapts search space based on population convergence	Yes	No	
DPE	Contracts search space based on population convergence	Yes	No	
Classifier	Machine learning	No	No	
CSM	Machine learning with memory	No	Yes	
SAMUEL	High level GA based machine learning	No	No	
EnGENEous	Hybrid expert system and GA	Yes	No	
MEL	Combines GA and expert system learning to provide for memory	Yes	Yes	Complex.

Table 2.1 Summary of literature review

Modifications to the simple genetic algorithm have been proposed for incorporating learning or 'intelligence'. Systems with learning, such as classifier, SAMUEL, and CSM are limited in that they are restricted to machine learning and cannot be used for optimization. EnGENEous, performs optimization through knowledge contained in an expert system rule base, and uses GA for escaping local minima and constraint boundaries. However, it does not retain learning across successive applications. Furthermore, creating the rule base for a problem domain, for example design of aircraft turbine engines, requires an additional effort. MEL is a problem specific optimization system that preserves memory. However, this expert system based genetic search is extremely complicated in structure and implementation. It consists of ten different modules that operate on a knowledge base with three levels of representation, and employs two transformations and three levels of learning. Thus it requires tremendous additional effort in implementation, for each new problem domain.

The idea of search space scaling or refining has been tested in several modifications, including delta coding, adaptive representation genetic optimizer, adaptive search space scaling, and dynamic parameter encoding. However, all these methods use population convergence as a basis for trimming the search space. This dependence on population convergence makes the above techniques independent of the problem domain. Consequently, they can be applied to a wide range of problems, without imposing constraints on the nature of the problem that can be solved. However, ignoring problem-specific information also gives rise to the possibility of trimming off the area containing the optimal point, and hence of sub-optimization. (This concern is of special importance in the simulation domain, where the simulation is typically a 'black box' with a complex and unknown landscape). Most of these techniques consequently incorporate an 'inverse trim' operator to overcome such situations. However, incorporation of an inverse trim operator necessarily increases complexity of the algorithm. Moreover, there still is no guarantee that the optimal area will not be discarded and sub-optimization will not occur.

Furthermore, all these techniques utilize algorithm parameters that have to be set by the user at run-time. The performance of the improved algorithms is sensitive to the values of these parameters, which often have to be determined in an ad-hoc manner.

Additionally, most modifications suggested to improve the optimization performance of GAs (dynamic parameter encoding, delta coding, adaptive search space scaling, and adaptive representation genetic optimizer) do not 'learn' over successive applications.

In summary, although there have been several modifications proposed to improve the performance of GAs, the results of practical implementation have been decidedly mixed. It seems there has not been any single accepted strategy to deal with constrained optimization problems. The reason for this might be experimental evidence that incorporation of problem-specific knowledge into the evolutionary algorithm enhances its performance (Michalewicz 1993).

In this research, an improved genetic algorithm that is applicable for the class of discrete constrained optimization problems with linear objective functions, is proposed. It exploits problem-specific information to improve the performance of genetic algorithms, by refining the search spaces over successive generations. The proposed dynamic scale genetic algorithm overcomes the deficiencies of the prevalent methods in the literature in the following manner:

1. Search space is trimmed conservatively, if and only if there is mathematical evidence that the eliminated portion does not contain the optimum.
2. Does not require an inverse zoom operator, hence is simpler to use and implement.
3. Does not employ parameters that have to be arbitrarily set by user at run-time, hence its performance is consistent.
4. Has a built-in memory feature that retains the knowledge acquired over successive applications.

CHAPTER 3

METHODOLOGY: DYNAMIC SCALE GENETIC ALGORITHM

3.1 Introduction

The simple genetic algorithm allocates reproductive opportunities to members of a population based on their relative fitness. By so doing, the search is directed towards regions that contain solutions with above average fitness. The fitness of the binary coded strings is evaluated by mapping them onto a fixed search space. The search space is unchanging through the run of the genetic algorithm, and is defined in accordance with the problem specification. This fixed mapping aspect has been recognized, at some times, as placing an unnecessary burden on the performance of a simple genetic algorithm. For instance, when the genetic algorithm has located a sub-space that contains the optimum, it might be more efficient to concentrate on this region alone.

The idea of refining the solution space by altering its boundaries as the genetic search progresses has consequently received attention in the literature. All these strategies (dynamic parameter encoding, adaptive representation genetic optimizer, and adaptive search space scaling) utilize population convergence measurements to dynamically change the mapping strategy. Michalewicz and Arabas (1994) observe that no single genetic algorithm strategy seems to have been accepted in practice for the general constrained optimization problem. The reason for this, they speculate, might be that the biased random search strategy considers only relative fitness and ignores problem specific information contained in its objective function, constraints and bounds. Experimental evidence suggests that incorporation of problem specific information into the genetic algorithm (representation and genetic operators), enhances its performance in a significant manner (Michalewicz 1993).

As a step in this direction, this research proposes to incorporate into the genetic algorithm, problem specific information contained in the objective function and constraints, so as to dynamically allocate tighter boundaries by changing the mapping strategy. The proposed dynamic scale genetic algorithm is applicable for a broad class of discrete optimization problems, z_{IP}, defined as

$$z_{IP} = \min\{cx : x \in S\} \tag{3.1}$$

The linear objective function denoted by cx is of the form $c_1x_1 + c_2x_2 + \ldots + c_ix_i + \ldots + c_nx_n$ where c_i represents the cost coefficient associated with the i^{th} variable x_i. The objective function is assumed to comprise of n such variables x_1 through x_n. The solution space S is defined by (linear or non-linear) inequality constraints on the non-negative, discrete variables. Thus, the only restriction placed on the general discrete optimization problem is that of linearity of the objective function. This class of problems, z_{IP}, commonly occurs in the management and efficient use of scarce resources to increase productivity (Nemhauser and Wolsey 1988, Parker and Rardin 1988). It includes the following well-known types of discrete optimization problems (all of which have linear constraints): traveling salesman, postman's, knapsack, parallel machine scheduling, vertex coloring, spanning tree, shortest path, bin packing, matching, set covering, maximum flow, p-median, and fixed charge

(Nemhauser and Wolsey 1988, Parker and Rardin 1988). Thus, it can be seen that the proposed strategy is applicable to a wide range of discrete problems.

3.2 Theoretical Basis

The dynamic scale genetic algorithm proposed in this research is based on partial enumeration, a discrete optimization technique also known as branch and bound. The partial enumeration scheme employs tests of feasibility and value dominance to eliminate from consideration subsets of the solution space. Generally in commercial codes, subsets are formed by adding constraints that divide the original problem, and solved as linear programs with the integrality constraints relaxed. In this section, the theoretical foundations of partial enumeration, used for the dynamic scale genetic algorithm proposed in this research, are presented.

3.2.1 Partial Enumeration Concepts for Integer Programming

Total enumeration is not a viable strategy for most practical discrete optimization problems, however *partial* or *implicit* enumeration can be used with better results. Partial enumeration is usually performed by eliminating entire subsets of the search space, without actually enumerating their individual candidate solutions. Consider the discrete linear integer programming problem given by $z_{IP} = \min\{cx{:}x \in S\}$. The solution space S can be considered to be made up of i divisions or mutually exclusive sets S^i, such that $\cup_{i=1}^{k} S^i = S$ where $\{S^i{:}i = 1,\ldots,k\}$. The following proposition holds true:

Proposition 1 (Nemhauser and Wolsey 1988):

Let $z_{IP}^{i} = \min\{cx : x \in S^{i}\}$, where $\{S^{i}\}_{i=1}^{k}$ is a division of S. Then $z_{IP} = \min_{i=1,\dots,k} z_{IP}^{i}$.

Therefore, a large optimization problem can be solved by attacking several smaller and consequently more manageable sub-problems, which are formed by dividing the original solution space into mutually exclusive subsets. Generally these subsets are formed through the addition of mutually exclusive constraints so that the original problem is divided into smaller subsets. Subsets are also formed by dividing the permissible range of a variable, into two or more ranges. Figure 3.1 depicts such subset divisions.

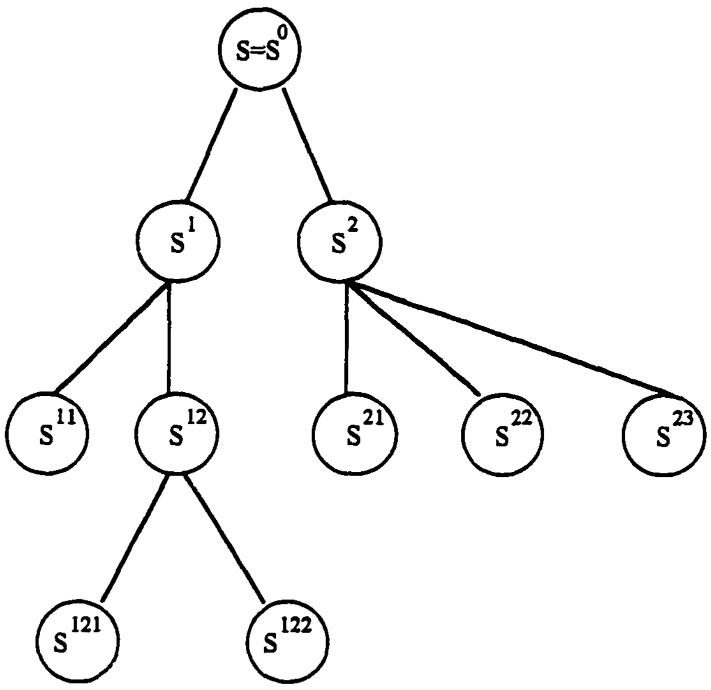


Figure 3.1 An enumeration tree showing subset divisions

(An extreme case of achieving such divisions would be by means of a complete enumeration tree). Further, suppose S has been divided into subsets $\{S^1,..S^i..,S^k\}$. If we can establish by the methods given in Proposition 2 below, that S^i does not contain the optimal value and hence need not be divided further, we say that the enumeration tree is 'pruned' at S^i. A subset is *pruned* if any of the conditions stated in Proposition 2 occur.

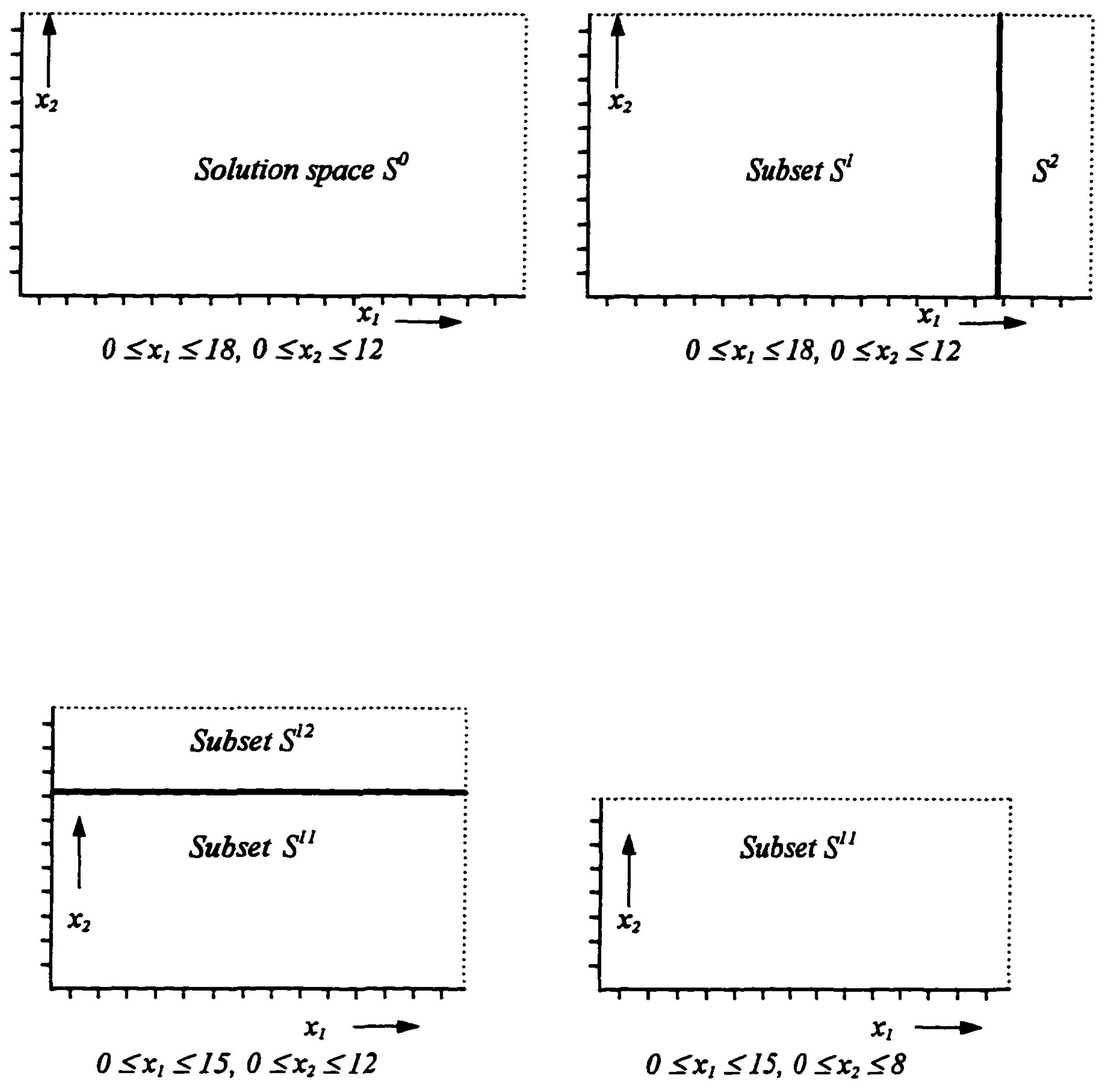
Proposition 2 (Nemhauser and Wolsey 1988):

No further division of a subset S^i is needed (and the enumeration tree can be pruned at the node corresponding to S^i), if any one of the following conditions hold:

1. Infeasibility: If subset S^i contains no feasible solutions.
2. Optimality: If an optimal solution z^i_{IP} is known.
3. Value dominance: $z^i_{IP} \geq \underline{z}_{IP}$, where $\underline{z}_{IP}$ is the value of some feasible solution.

By using the infeasibility, optimality and value dominance conditions of proposition 2, a subset can be *fathomed*, or eliminated from consideration. For example, if it can be established that a subset S^i cannot produce a feasible solution, then it need not be considered further. If the optimal solution is known, subsets can be eliminated as they do not contain a solution better than the known optimal. Similarly, a subset can be fathomed using the condition of value dominance. If it can be established that the best solution contained in a subset is inferior compared to the current best solution, the subset can be fathomed. Thus, the current best feasible solution of a subset $\underline{z}_{IP}$, obtained during a previous partial enumeration can be used to determine whether a subset needs to be investigated further.

Fathoming subsets that are formed by dividing the permissible range of variables, yields tighter upper and lower limits on the concerned variable. This concept demonstrated in Figure 3.2, is known as *pegging* and aids in compacting the solution space further (Parker and Rardin 1988). Pegging is achieved through value dominance. If it can be established that for a certain range of a variable, the solutions produced are inferior to the best feasible solution so far, tighter limits can be placed on the variable by eliminating the range under consideration. In Figure 3.2, the original solution space S^0 is bounded by the variables x_1 and x_2 with permissible ranges of $0 \leq x_1 \leq 18$ and $0 \leq x_2 \leq 12$ respectively. This solution space can be compacted by restricting the upper limits or permissible scale ranges of the variables, as shown below. Subsets S^1 and S^2, are formed by dividing the scale range of variable x_1 at 15, i.e. $0 \leq x_1 \leq 15$ and $16 \leq x_1 \leq 18$. If subset S^2 can be fathomed, a tighter upper bound $0 \leq x_1 \leq 15$ can be placed on variable x_1. Similarly, subset S^1 can be further divided into subsets S^{11} and S^{12}, which are formed by dividing the permissible scale range of variable x_2 at 12. Once again, if subset S^{12} can be fathomed, then a tighter upper bound can be placed on variable x_2, as $0 \leq x_1 \leq 12$. The remaining subspace S^{11} , bounded by tighter permissible scale ranges, can be solved by enumeration, or any other optimization strategy.



x_2
Solution space S^0
x_1
$0 \leq x_1 \leq 18,\ 0 \leq x_2 \leq 12$
x_2
Subset S^1
S^2
x_1
$0 \leq x_1 \leq 18,\ 0 \leq x_2 \leq 12$
Subset S^{12}
Subset S^{11}
x_2
x_1
$0 \leq x_1 \leq 15,\ 0 \leq x_2 \leq 12$
Subset S^{11}
x_2
x_1
$0 \leq x_1 \leq 15,\ 0 \leq x_2 \leq 8$

Figure 3.2 Tightening scale ranges of variables

3.3 Incorporating Partial Enumeration Concepts

In terms of the foregoing discussion, a genetic algorithm can be viewed as a ***biased random partial enumeration*** scheme, where relative fitness information is used to partially enumerate the solution space. As such there are similarities between partial enumeration and the genetic search schemes. Both strategies attempt to produce an optimal solution by using fitness or objective function (not indirect information such as derivatives), without enumerating the entire search space. However, there is an *important difference*. The partial enumeration scheme works by shrinking its target solution space as it proceeds, whereas the genetic algorithm works with the entire original solution space.

This research attempts to build some intelligence into genetic search, by using partial enumeration concepts to shrink the target solution space. Specifically, constraint and objective function information of the current best feasible solution in the population is used to form mutually exclusive subsets, by dividing the permissible ranges of variables. A rule involving elementary mathematical operations was developed in this research to achieve this subdivision. Subsets so obtained are subsequently fathomed through value dominance, thus tightening permissible ranges and refining the solution space. It is hypothesized that eliminating subsets from consideration in this manner enables the genetic algorithm to concentrate on the more promising areas of the solution space, thus improving its performance.

3.4 Dynamic Scale Genetic Algorithm (DyScGA)

The concepts of value dominance and pegging described above were incorporated into the simple genetic algorithm. A rule utilizing the elementary operations of division and multiplication was developed for forming and fathoming subsets. The rule is based on the following proposition:

Proposition 3:

Let a be the current best feasible solution to the constrained optimization problem $z_{IP} = \min\{cx : x \in S\}$, where c_i. are non-negative cost coefficients associated with the non-negative decision variables x_i. Let the subsets formed by dividing the permissible scale of variable x_i $(0 \leq x_i \leq z_i)$ be S^1 $(0 \leq x_i \leq |a/c_i|)$ and S^2 $(|a/c_i|+1 \leq x_i \leq z_i)$, where the symbol $|y|$ represents the integer part of a real number y. Then subset S^2 can be fathomed.

Proof:

Let $x_l = |a/c_l|+1$ thereby guaranteeing membership in the subset S^2. In order to compute the least objective function value associated with $x_l = |a/c_l|+1$, set all other variables $x_i = 0,\ i \neq l$. Since c_l are non-negative, the lowest objective function value can only be attained if all other variables $x_i = 0,\ i \neq l$. In this case, the objective function value is $(c_l\ |a/c_l|+1) = a+c_l > a$. Therefore, through value dominance the subset S^2 can be fathomed.

The above rule was incorporated into the simple genetic algorithm. The resulting dynamic scale genetic algorithm is presented in the form of a flow chart in Figure 3.3. It is seen that the DyScGA adds some steps to the simple genetic algorithm. The following sections describe these additional steps that achieve fathoming through value dominance.

Step 1. Forming an initial population

As shown in the flow chart, an initial population of randomly selected solutions from the search space is formed in a manner identical to the simple genetic algorithm.

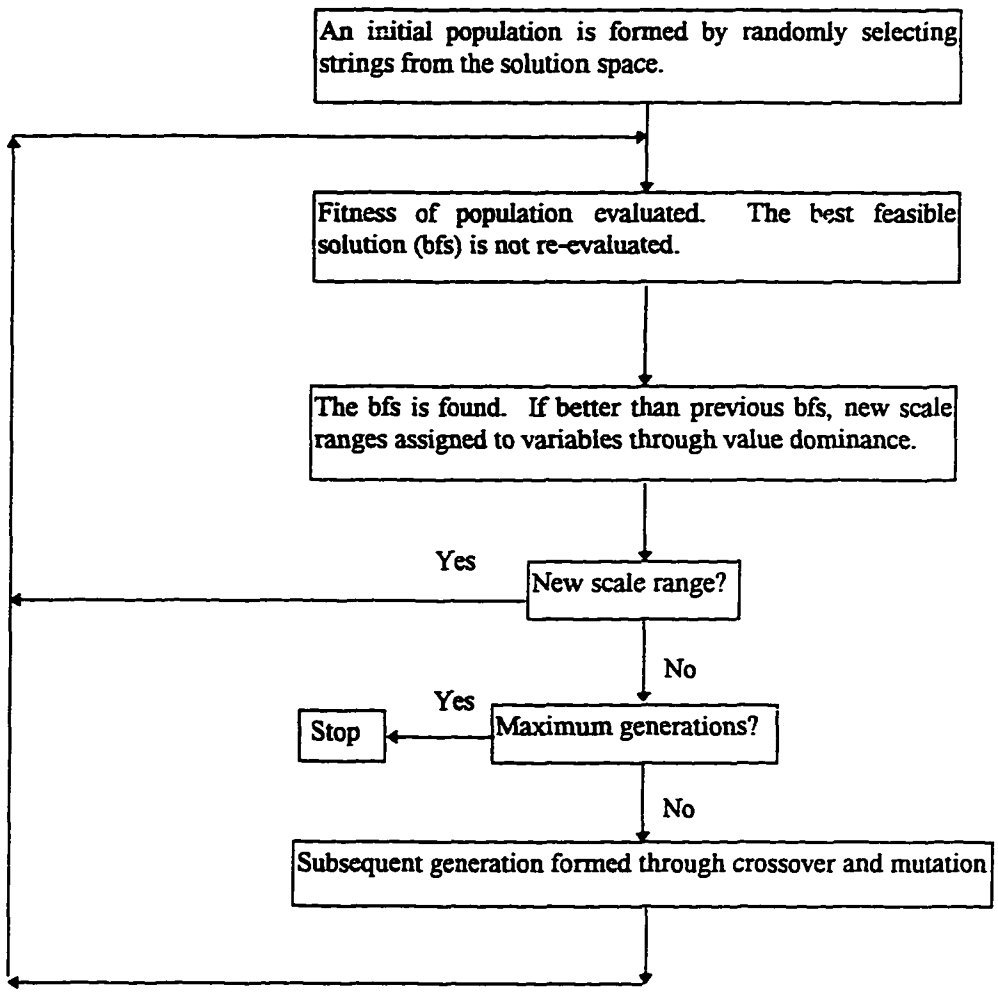

An initial population is formed by randomly selecting strings from the solution space.
Fitness of population evaluated. The best feasible solution (bfs) is not re-evaluated.
The bfs is found. If better than previous bfs, new scale ranges assigned to variables through value dominance.
Yes
New scale range?
No
Yes
Stop
Maximum generations?
No
Subsequent generation formed through crossover and mutation

Figure 3.3 Flowchart: Dynamic Scale Genetic Algorithm

Step 3. Assigning new scale ranges based on the best feasible solution

After these initial steps which are similar to the simple genetic algorithm, the DyScGA requires some additional steps for forming subsets based on scale ranges and pegging through value dominance. The best feasible solution (bfs) among the current population is first determined. The bfs is used to divide the permissible scale ranges of decision variables by the formation of subsets, in accordance with the rule developed above. The rule is explained here through a numerical example. Consider the objective function of a constrained optimization problem (Gass 1985):

$$\begin{array}{llllllllll} \textit{Minimize} & 4x_1 + & 12x_2 + & 2x_3 & & + 5x_5 + & 10x_6 & & & \\ \textit{such that} & 2x_1 + & x_2 + & x_3 & & & & & \geq & 30 \\ & & x_2 & & + 3x_4 + & 2x_5 + & x_6 & & \geq & 60 \\ & & & 2x_3 & & + x_5 + & 2x_6 + & 4x_7 & \geq & 48 \end{array} \qquad (3.2)$$

$$0 \leq x_i \leq 99, \quad i = 1, \ldots, 7$$
$$x_i \text{ are non-negative integers}$$

The original scale range for all seven variables is 0 to 99, i.e., each variable can take any of the values 0, 1, 2, ..., 99. Therefore, the search space is made up of 10^{14} discrete combinations. (Since this is a constrained optimization problem, several of these combinations may be infeasible). Let the *best feasible* objective function or solution in the randomly chosen initial population of the DyScGA be evaluated as 989 ($x_1 = 32, x_2 = 38, x_3 = 40, x_4 = 10, x_5 = 37, x_6 = 14, x_7 = 90$). The contribution of each variable to the best feasible solution is computed in order to determine whether its permissible range can be subdivided. This is done simply by dividing the best feasible solution, 989, by the cost coefficient or weightage of each variable. Therefore, for the first variable x_1 with coefficient 4, this yields a contribution of 247.25 (989/4). Since this value is higher than the upper bound of the original scale range (99), it does not divide the scale range and no subset can be formed.

Next the contribution by variable x_2 to the best feasible solution is determined as 82.4 (989/12). Since the contribution 82.4 divides the original permissible scale range of 99, subsets can be formed at its absolute value. (The absolute value is used, since all variables are restricted to be integers). The subsets are defined by changing the permissible scale range $0 \leq x_2 \leq 99$ of the original problem. The first subset is formed by restricting the original scale range of variable x_2 to the absolute value of the contribution 82, i.e., $0 \leq x_2 \leq 82$. The second mutually exclusive subset is formed by modifying the original scale range so that variable x_2 exceeds 82, i.e., $83 \leq x_2 \leq 99$. Next, value dominance based on the best feasible solution (989) is used to fathom one of these subsets. It is established that when the variable x_2 exceeds 83, i.e., x_2 lies in the second subset, its contribution to the linear objective function is greater than or equal to 996 (=12*83). Therefore the best objective function associated with the second subset is at least 996. Since we already have a best feasible solution that is better (989), the second subset can be fathomed through value dominance. The permissible scale range of variable x_2 is tightened by limiting it to 82 ($0 \leq x_2 \leq 82$) and the original search space is reduced.

The above process of forming subsets and fathoming through value dominance is repeated for all the variables. The scale ranges for variables x_3, x_4, x_5, x_7 cannot be divided based on the value of the current best feasible solution. (Variables x_4 and x_7 do not contribute to the objective function at all, and hence their respective scale ranges will not be divided through the entire run.) Variable x_6 contributes 98.9 (989/10=98.9) to the best feasible solution and hence its original scale range is divided at its absolute value, 98. The first subset contains the scale range $0 \leq x_6 \leq 98$, and the second subset is formed by fixing the variable x_6 at 99. Using value dominance, the subset with variable $x_6 = 99$ can be fathomed, as the contribution of this variable to the objective function is at least 990 (=99*10), which is greater than the best feasible solution of 989.

By forming and fathoming the above two subsets, the search space is reduced to $80.36*10^{12}$ combinations. Since the permissible scale ranges of the variables x_2 and x_6 have been reduced, new boundaries for these variables are assigned within the dynamic scale genetic algorithm. It may be worth noting that, as the permissible scale ranges for variables shrink, the corresponding number of bits required to encode the range of variables may also shrink. Dropping bits has the added benefit of improving the performance of the GA, as fewer bits prove more effective during the genetic search. (In keeping with the elitist strategy which preserves the current best solution through the subsequent generation, the number of bits are dictated by the current best feasible solution. Therefore, although new scale ranges are immediately assigned for the entire population, reducing the number of bits is delayed until the current bfs demands it.)

Step 4. Determining if new scale ranges are assigned

If no new scale ranges are assigned in Step 3 above, the dynamic scale genetic algorithm proceeds to Step 5. However, if new scale ranges are assigned in Step 3, dynamic scale genetic algorithm loops back to Step 2. The population is re-evaluated with the new mapping strategy. This feature injects diversity into the population, a key requirement for an effective genetic search, as the binary encoding is mapped onto different points in the solution space. Thus, the DyScGA has a built-in mechanism for enhancing diversity.

Step 5. Determining if maximum generations are reached

If the maximum number of generations specified by the user at run-time has been reached, the program terminates. If not, the program advances to Step 6, for the formation of the subsequent generation.

Step 6. Forming subsequent generation through crossover and mutation

Through the genetic operators of crossover and mutation, the next generation is formed in a manner identical to the simple genetic algorithm. After a new generation is formed through reproduction, DyScGA loops back to Step 2, and the entire cycle of population evaluation, new scale range assignments and reproduction repeats.

3.5 Features of the Dynamic Scale Genetic Algorithm

3.5.1 Theoretical Basis

The dynamic scale genetic algorithm applies a simple genetic algorithm to successively refined search spaces. The genetic operators are not modified and the basic scheme of the simple genetic algorithm remains unchanged. Therefore, the theoretical basis of the simple genetic algorithm still holds over successive adjustments to the scale ranges and hence the solution space (Whitley et al. 1991, Schraudolph and Belew 1992).

3.5.2 Diversity

The DyScGA re-evaluates the members of a population with progressively tighter scale ranges. This improves the sampling of the solution space by mapping the candidates onto different points in the search space. This feature injects diversity, a key component in the quality of the solution produced by a genetic algorithm (Goldberg 1989).

3.5.3 Learning and Memory

Unlike the genetic algorithm, the dynamic scale genetic algorithm learns problem specific information and retains it in the form of a memory for future use. During a single run, the dynamic scale genetic algorithm progressively refines the solution space, by trimming areas that do not contain the optimum. Therefore, the algorithm 'learns' about the solution space over successive generations. This knowledge is retained for future use across subsequent runs of the DyScGA to avoid having to re-learn. This constitutes a permanent memory, a feature which improves the performance of the DyScGA. It has been established in the genetic algorithm literature that if problem specific knowledge from past experience is retained, it can help seed the initial population of a subsequent run to improve the performance in a significant manner (Davis 1987).

3.6 Advantages of DyScGA

Most of the enhancements to the simple genetic algorithm suggested in the literature pertain to continuous variables. The dynamic scale genetic algorithm proposed in this research is the first enhancement specifically applicable for discrete variables. In addition to the unique application area, it has several advantages over other GA-modifications suggested in the literature.

1. Unlike other permissible scale range adjusting improvements suggested in the literature (dynamic parameter encoding, adaptive search space scaling, adaptive representation genetic optimizer technique, and delta coding), DyScGA does not require additional control parameters that have to be arbitrarily set by the user. For example, Schraudolph and Belew's dynamic parameter encoding (DPE) (1992) requires the user to set a trigger threshold in order to adjust the resolution of a parameter. The authors note that setting the trigger threshold is complicated and no rules are available to guide the selection process. If the trigger threshold is too high, the DPE will not be able to eliminate suboptimal areas of the search space from consideration. On the other hand, if the threshold is too low the algorithm may easily be triggered by noise. The performance of the DPE algorithm is thus dependent on the value of the trigger threshold. The proposed DyScGA does not employ such parameters, thus eliminating subjectivity and guess-work and simplifying the implementation of the algorithm.
2. DyScGA does not require inverse operators, such as those employed by the adaptive search space scaling, the adaptive representation genetic optimizer technique, and the delta coding techniques. Unlike these methods, the search space is trimmed conservatively if and only if there is mathematical evidence that the optimal does not lie in the eliminated search space. This simplifies the implementation of the dynamic scale genetic algorithm.
3. The DyScGA has a built in learning *and* memory feature. Problem specific learning occurs over successive generations and is stored for use with future runs.

There is no such learning or memory in a simple genetic algorithm. Nutter and Ding's (1992) multi-leveled environment for learning provides for learning and memory in a computer network system application domain. However, with its ten modules operating on layers, three levels of representation, two transformations and three levels of learning, it is extremely complex to build and implement.

4. By re-evaluating a population with different mapping strategies, the diversity of the population is significantly enhanced. This promotes the exploration of the search space, an essential ingredient to effective genetic search.
5. DyScGA does not alter the structural properties of the genetic operators of a standard genetic algorithm, and hence retains all the features of the latter. DyScGA can be characterized by successive GAs applied iteratively to search spaces refined through pruning. Therefore, the theoretical foundations and analysis for GAs still applies.

3.7 Limitations of DyScGA

The main limitation of the dynamic scale genetic algorithm proposed is that its applicability is not independent of the problem domain. The DyScGA can only be applied to discrete optimization problems employing a linear objective function. Therefore, generality is sacrificed for performance. However, this can be justified for two reasons. First, discrete optimization problems with a linear objective function include most of the commonly occurring discrete optimization problems in industry and management settings, and encompass the integer programming problems. Second, GA-modifications that are independent of the problem domain, such as DPE, ARGOT, adaptive search space scaling, and delta coding, have a performance that varies vastly from problem to problem. This implies that, while these strategies may give good results for a particular problem domain, its performance on another problem domain may be poor (Michalewicz and Arabas 1994). This occurs mainly due to their dependence on population convergence measurements, which can sometimes produce mis-leading indications for triggering the trimming action.

A second limitation of the DyScGA is that the enhanced diversity comes at a price -- an increased computational burden associated with re-evaluation of the population. In the case of a simple genetic algorithm, diversity is enhanced by employing a large population. However, a large population also implies greater computational requirements associated with reproduction (crossover and mutation), in addition to population evaluation. There is, however, an important difference between the nature of computational requirements of the two algorithms. An increased population size in the simple genetic algorithm demands higher computational requirements, regardless of the state of the population. Thus, even when the population has converged, reproduction and population evaluation are performed until the pre-specified number of generations is reached. On the other hand, the dynamic scale range genetic algorithm re-evaluates populations and hence requires more computations, if and only if there is a change in the scale range of a variable. As a result, if the population has converged, there will be no extra population evaluations and the computational burden does not increase.

3.8 Implementation of DyScGA

DyScGA was implemented by incorporating a module into an existing simple genetic algorithm, written in FORTRAN 77® and run on a SUN SPARC® workstation. The simple genetic algorithm has been validated and tested, and is used extensively at NASA Langley Research Center (Gage 1995). It uses the elitist strategy, i.e., it carries the best solution unchanged into the next generation, thus preserving it. It uses a tournament selection scheme, which determines parents by picking the best of two potential mates.

3.9 Verification and Validation

Verification was performed to ensure that the computer code, written to translate the DyScGA concepts developed in this research into machine language, was appropriate. This was done by solving a problem with a known optimal value, and verifying the outputs of the computer program as it progressed. Once it was ascertained that the computer code was satisfactory, validation was performed to establish that the DyScGA does indeed have a superior performance compared to the simple genetic algorithm. By using identical values for the initial random number seed and control parameters, the performance of both algorithms can be compared directly.

3.9.1 Verification

Verification was performed to ensure that the software code written for DyScGA is accurate. This was done by examining and verifying each step of the algorithm as it proceeded. The problem given by equation (3.2), with a known objective function of 60 (Gass 1985), was selected as a test case for the purpose of verification. A genetic algorithm (and the dynamic scale genetic algorithm) does not make provisions for explicit constraint formulation. Instead, a constraint is stated in terms of a penalty for violating it. The objective function of problem (3.2) was therefore modified to accommodate constraint violation through appropriate penalty functions. The penalty function, in case of constraint violation, was selected by trial and error on the simple genetic algorithm as follows:

$$\begin{aligned} &100((2x_1 + x_2 + x_3)-30)^2 \\ &100((x_2 + 3x_4 + 2x_5 + x_6)-60)^2 \\ &100((2x_3 + x_5 + 2x_6 + 4x_7)-48)^2 \end{aligned} \qquad (3.3)$$

The following values were used for the random number generator seed and control parameters: initial random number seed 100, population size 100, crossover probability 0.95, mutation probability 0.2 (De Jong 1975, Grefenstette 1986, Schaffer et al. 1989).

The DyScGA was run for ten generations with the above configuration. The results and outputs generated as the dynamic scale genetic algorithm stepped through the generations are presented in Table 3.1. The following steps were executed by the DyScGA.

1. *Step 1. Forming an initial population*

An initial population was randomly selected. This step is carried out by the simple genetic algorithm code.

Gen	*Bfs*	*Scale ranges*		*Best feasible solution / coefficient*					*Re-eval*
		x_1 $\lvert bfs/4 \rvert$	x_2 $\lvert bfs/12 \rvert$	x_3 $\lvert bfs/2 \rvert$	x_4	x_5 $\lvert bfs/5 \rvert$	x_6 $\lvert bfs/10 \rvert$	x_7	
Gen 1	2593	-	-	-	-	-	-	-	
	1307	-	-	-	-	-	-	-	
	721	-	60	-	-	-	72	-	
	359	89	29	-	-	71	35	-	Yes
Gen 2	153	38	12	76	-	30	15	-	Yes
	151	37	12	75	-	30	15	-	Yes
	149	37	12	74	-	29	14	-	
	69	17	5	34	-	13	6	-	Yes
	64	16	5	32	-	12	6	-	
	62	15	5	31	-	12	6	-	Yes
	60	15	5	30	-	12	6	-	Yes
Gen 3	60	15	5	30	-	12	6	-	
Gen 4	60	15	5	30	-	12	6	-	
Gen 5	60	15	5	30	-	12	6	-	
Gen 6	60	15	5	30	-	12	6	-	
Gen 7	60	15	5	30	-	12	6	-	
Gen 8	60	15	5	30	-	12	6	-	
Gen 9	60	15	5	30	-	12	6	-	
Gen 10	60	15	5	30	-	12	6	-	

(Underlined values indicate intermediate scale ranges that were not actually implemented)

Table 3.1 Stepping through the DyScGA

2. *Step 2. Evaluating fitness of population*

The fitness, in terms of objective function and constraints, was evaluated for the entire population. This step was also performed by the simple genetic algorithm.

3. *Step 3. Assigning new scale ranges based on the best feasible solution*

This step involves the dynamic scale genetic algorithm module. The DyScGA determined the best feasible solution in the initial population by comparing the fitness values of each member determined in Step 2 above. The best feasible solutions are listed in Table 3.1 in the order in which they were found. The first best feasible solution found was 2593. New scale ranges, if any, were determined by computing the contributions of each variable to the bfs of 2593. As the table indicates, no new scale ranges were required at this best feasible solution. This occurred because the contribution of each variable to the bfs was greater than the original scale range of 99. For example, the contribution of variable x_1 to this bfs, determined as 648 ($|2593/4|$), does not divide the original scale range of 99. Hence, subsets of the original search cannot be formed, and no new scale ranges were assigned.

Next, a best feasible solution of 1307 was found. Once again no new scale ranges were required. Following this, a best feasible solution was found at 721. Based on the contributions of the variables to this bfs value, the scale ranges of variables x_2 and x_6 can be divided at 60 ($=|721/12|$) and 72 ($=|721/10|$) respectively. Subsets can be formed by supplying the following permissible scale ranges instead of the original: Subset^{11} : $0 \leq x_2 \leq 60$, Subset^{12} : $61 \leq x_1 \leq 99$. Subset^{12} can be fathomed through value dominance, as it contains solutions with fitness of at least 732 ($=61*12$), which are greater than the best feasible solution of 721. The new permissible scale range of variable x_2 was now limited to 60, i.e., $0 \leq x_2 \leq 60$.

Similarly, for variable x_6 the following subsets can be formed: Subset^{21} : $0 \leq x_6 \leq 72$, and Subset^{22} : $73 \leq x_1 \leq 99$. Once again, Subset^{22} can be fathomed through value dominance, as it produces solutions that have fitness of at least 730 (=73*10), which are greater than the best feasible solution of 721. Therefore, the new permissible scale ranges for variable x_6 were limited to 72, i.e., $0 \leq x_6 \leq 72$. Next, a best feasible solution of 359 was found. In a similar manner, variables x_1 , x_2,, x_5 , and x_6 were assigned tighter scale ranges of 89, 29, 71 and 35, respectively. There was no scale change for the variables x_3 , x_4, and x_7 as their respective contributions were higher than their original scale ranges of 99. For example, for variable x_3 , the contribution was 179 (=|359/2|), which is higher than 99. Variables x_4 and x_7 do not figure into the objective function, and hence no scale change was required for them.

4. *Step 4. Determining if new scale ranges were assigned*

The scale ranges for the problem were now refined as follows: $0 \leq x_1 \leq 89$, $0 \leq x_2 \leq 29$, $0 \leq x_3 \leq 99$, $0 \leq x_4 \leq 99$, $0 \leq x_5 \leq 71$, $0 \leq x_6 \leq 35$, and $0 \leq x_7 \leq 99$. Since new scale ranges were assigned, the program looped back to Step 2 for re-evaluation of the population.

5. *Step 2. Evaluating fitness of population*

Since scale ranges were tightened, the population members now represented different points in the solution space. The population was re-evaluated with the new mapping strategy (scale ranges 89, 29, 99, 99, 71, 35, 99). The population member representing the best feasible solution of 359 was left unchanged without re-evaluation, since the elitist strategy which preserves the bfs was used.

6. *Step 3. Assigning new scale ranges based on the best feasible solution*

The best feasible solution was found to be 359 again, since the re-mapped population did not produce a better solution. Therefore, no new scale ranges were required.

7. *Step 4. Determining if new scale ranges were assigned*

Since the scale ranges were unchanged (89, 29, 99, 99, 71, 35, 99) the program advanced to Step 5.

8. *Step 5. Determining if maximum generations were reached*

The current generation was the first generation, which is less than the maximum allowable generations of 10. Therefore, the computer code progressed to Step 6.

9. *Step 6. Forming subsequent generation through crossover and mutation*

The next generation was formed through the genetic operators of crossover and mutation. The program then moved to Step 2 for evaluation of the new population.

10. *Step 2. Evaluating fitness of population*

The new population was re-evaluated. The best feasible solution which was preserved by the elitist strategy was left unchanged.

11. *Step 3. Assigning new scale ranges based on the best feasible solution*

The best feasible solution from among the newly produced population was found as 153. New scale ranges were assigned based on this bfs as: $0 \leq x_1 \leq 38$, $0 \leq x_2 \leq 12$, $0 \leq x_3 \leq 76$, $0 \leq x_4 \leq 99$, $0 \leq x_5 \leq 30$, $0 \leq x_6 \leq 15$, and $0 \leq x_7 \leq 99$.

12. *Step 4. Determining if new scale ranges were assigned*

Since new scale ranges were assigned, the program looped back to Step 2 for population re-evaluation.

13. *Step 2. Evaluating fitness of population*

The population was re-evaluated with the new mapping strategy, except for the best feasible solution.

14. *Step 3. Assigning new scale ranges based on the best feasible solution*

The best feasible solution from among the newly produced population was found as 151. New scale ranges were assigned based on this bfs as: $0 \leq x_1 \leq 37$, $0 \leq x_2 \leq 12$, $0 \leq x_3 \leq 75$, $0 \leq x_4 \leq 99$, $0 \leq x_5 \leq 30$, $0 \leq x_6 \leq 15$, and $0 \leq x_7 \leq 99$.

As Table 3.1 indicates, the program continued in this manner until there was no more improvement in the best feasible solution. This occurred at fitness value 60, and new scale ranges were set as follows: 15, 5, 30, 99, 12, 6, 99. At this point the third generation was formed through reproduction and crossover. Since this third generation did not contain a better feasible solution, the DyScGA proceeded in the manner of a simple genetic algorithm through generation 10.

In this way, the computer code written for the DyScGA was fully verified.

3.9.2. Validation

The verified dynamic scale genetic algorithm code was validated to ensure that it performs better than the simple genetic algorithm. For the purpose of validation, the performance of both algorithms was compared by solving problem 3.2 above. To enable a direct comparison, the same initial random number seed (100) and algorithm parameters (population size 100, crossover probability 0.95, mutation probability 0.2) were used for both cases. Doing so enables an exact comparison between the results obtained by the simple GA and the DyScGA. It can be seen that the performance of the DyScGA at these settings has been examined in the preceding section.

The simple GA was run with the above control parameter settings and initial random number generator seed. It was run for a total of 25 generations, *greater* than the dynamic scale genetic algorithm generations (10). Table 3.2 presents the performance of the simple GA. It contains the best fitness and average fitness values found during each generation. The results demonstrate and underscore the utility of the DyScGA.

It can be seen, from Table 3.2, that the quality of the best fitness improved with each subsequent generation for the simple genetic algorithm. The average fitness value was observed to fluctuate through the generations. This can be attributed to the incidence of infeasible solutions in the population, for each generation. Occurrences of infeasible solutions are due to constraint violations that are penalized by increasing the objective function value. This increase contributes to a higher average fitness over the generation. The final solution corresponding to the best fitness was found by the simple genetic algorithm, in the 24th generation, as 83. This fitness value given in Table 3.2, turned out to be considerably higher than the known optimal value of 60.

Next the dynamic scale genetic algorithm with identical control parameters, which was applied to the same problem (3.2) for 10 generations, is considered. The results are tabulated in Table 3.3. Table 3.3 contains an additional column that depicts the number of

Gen	Best fitness	Avg fitness
1	359	1637
2	251	1426
3	227	1254
4	227	1092
5	213	901.2
6	213	922.5
7	193	1095
8	193	964.1
9	193	890.2
10	193	965.5
11	123	1520
12	123	1065
13	87	909.1
14	87	1303
15	87	1350
16	87	968.5
17	87	880.9
18	87	1441
19	87	1006
20	87	1070
21	87	2632
22	87	843.2
23	87	1190
24	83	880
25	83	1827

Table 3.2 Results obtained by simple genetic algorithm

times the population was re-evaluated due to a change in the scale range of a variable. Identical settings of the control parameters and the starting random number generator seed enable a direct comparison with the simple genetic algorithm. Care was taken to run the simple genetic algorithm for extra generations (25), due to the population re-evaluations performed by the dynamic scale genetic algorithm which was run for only 10 generations.

Gen	Best fitness	Average fitness	Population re-evaluation
1	359	1346	1
2	60	1.33E+04	5
3	60	6558	--
4	60	4738	--
5	60	5562	--
6	60	6249	--
7	60	9644	--
8	60	1.21E+04	--
9	60	1.10E+04	--
10	60	3362	--

Table 3.3 Results obtained by the DyScGA

The first generation for both the plain GA and the DyScGA was formed by the random selection of an initial population. This population, as expected, was identical for both algorithms, and the best fitness was 359. It can be seen that as with the simple GA, the best fitness improved with each generation. In the case of DyScGA, however, the rate of improvement in the best fitness can be observed to be more rapid. Furthermore, the quality of the final solution produced by the DyScGA at 60 was superior to that found by the simple genetic algorithm (83). This optimal solution of 60 was found in the second generation itself, after a total of six extra population evaluations as in section 3.9.1.

Average fitness values in Table 3.3 are noticeably higher than the corresponding values for the simple genetic algorithm. This can be explained by the higher percentage of infeasible solutions present in the search space refined by the DyScGA. Typically, the optimal for constrained optimization problems is located near the boundary of the feasible and infeasible regions. As the DyScGA tightens bounds, the search space is constricted and progressively contains a larger percentage of infeasible solutions in comparison to feasible solutions. Therefore, chances of generating infeasible solutions in a constrained optimization problem increase, driving the average fitness value for DyScGA higher.

A graphical depiction of the improvement associated with the DyScGA, based on the results in Tables 3.2 and 3.3, is given in Figure 3.4. From the above comparison, it can be concluded that the overall performance of the DyScGA, in terms of final solution (60 versus 83) and computational requirements (2 generations and 6 extra population evaluations, versus 24 generations with no extra population evaluations), is considerably superior to the simple genetic algorithm. This validates the dynamic scale genetic algorithm. It demonstrates the learning feature of the DyScGA which results in a noticeable improvement in its performance.

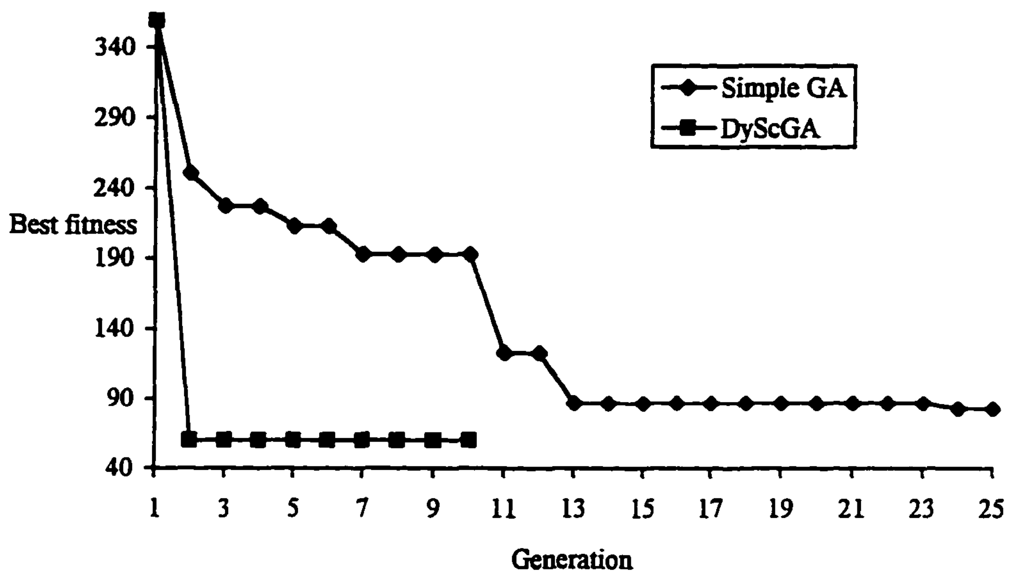


Figure 3.4 Graphical comparison of the simple GA and the DyScGA

Next, we turn to the memory feature of the dynamic scale genetic algorithm which enables the permanent storage of the information learned during each independent run. To validate the memory feature and its benefit, the DyScGA was compared to the simple genetic algorithm and to the DyScGA with memory feature disabled for validation purposes. These three strategies were used to solve the problem given by equation (3.2). As before the dynamic scale genetic algorithm was run for 10 generations, and the simple genetic algorithm for 25 generations. Each independent run of each algorithm was initialized by identical random number generator seeds. That is, the first run of all the three strategies started with seed 100, the second run started with seed 200, and so on. In addition, the control environment for all three strategies was maintained at: population size 100, crossover probability 0.95, and mutation probability 0.2.

These two measures promote a direct comparison between the DyScGA with memory disabled and the simple GA, in the sense that they start out with an identical control environment and an identical population. However, in the case of DyScGA with memory, although the control environment is the same, the population may not necessarily be so. This is because, due to the memory effect, each run of the DyScGA may have variables whose permissible scale ranges have been progressively tightened. Therefore, although the initial binary population may be the same as that of the simple GA and DyScGA with memory, it may represent different solutions when mapped onto a more compact solution space. The first independent run of both the DyScGA and the DyScGA with memory disabled were identical. They both produced a final solution of 89, with 21 population re-evaluations. (The simple GA produced a final solution of 145 during its first run). During the second run, all three strategies started out with an identical seed. However, while DyScGA remembered the tighter scale ranges learned during its previous run, DyScGA with memory disabled did not store these. Therefore, it started with the original scale ranges of the problem. Table 3.4 contains the solutions obtained by the DyScGA, simple GA and DyScGA with memory feature disabled, for the second independent run. Figure 3.5 contains a graphical depiction of the memory effect, based on the second independent run of Table 3.4. It is observed that DyScGA produced a final solution of 64, with five population re-evaluations. In comparison, DyScGA with memory disabled produced a higher final solution of 84 and required more population re-evaluations (11). This can be attributed to its lack of memory of the new scale ranges obtained during its first independent run. On the other hand, DyScGA retained the refined scale ranges learned during its first independent run. Thus the learning and memory feature of the DyScGA has been demonstrated and validated.

Gen	Simple GA	DyScGA		DyScGA with memory disabled	
	Best fitness	Best fitness	Pop. re-evals	Best fitness	Pop. re-evals
1	533	115	0	174	6
2	382	111	0	174	6
3	352	84	1	143	8
4	352	84	1	143	8
5	352	76	4	98	10
6	322	76	4	98	10
7	322	64	5	98	10
8	322	64	5	84	11
9	212	64	5	84	11
10	212	64	5	84	11
11	212				
12	204				
13	184				
14	184				
15	184				
16	164				
17	164				
18	164				
19	164				
20	164				
21	164				
22	164				
23	164				
24	164				
25	164				

Table 3.4 Comparison of DyScGA, simple GA and DyScGA with memory disabled

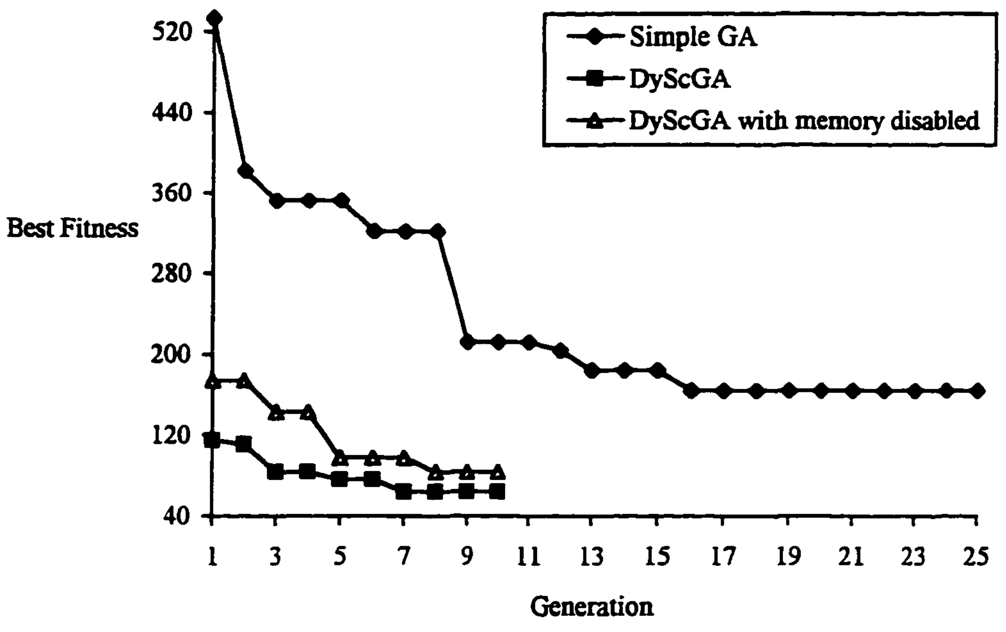


Figure 3.5 Graphical comparison of the simple GA, the DyScGA, and the DyScGA with memory disabled

3.10 Summary

The dynamic scale genetic algorithm, which is based on the theoretical foundation of the discrete optimization technique of implicit enumeration, was presented in this chapter. The DyScGA incorporates the concept of forming mutually exclusive subsets of a problem through an appropriate division of the scale ranges of its variables into the simple genetic algorithm. It then uses value dominance to systematically eliminate these subsets from consideration. The DyScGA can be viewed as multiple applications of the simple genetic algorithm to a solution space that is successively refined, by dynamically changing the mapping strategy. By refining the associated solution space over successive generations, the DyScGA aims to improve the performance of the simple GA.

The DyScGA eliminates subsets of the solution space if and only if there is mathematical evidence that the subset does not contain the optimum. Therefore, unlike other enhancements suggested in the GA literature it does not require inverse operators to re-claim previously discarded subsets. Furthermore, the other methods in the literature are triggered by population convergence measurements. This requires the user to arbitrarily set threshold levels for determining convergence, a process which impacts the performance of the method itself. The DyScGA, on the other hand, requires no additional inputs other than the problem itself. In addition, through the use of a changing mapping strategy, genetic diversity, an essential component for a successful genetic search is introduced into the population of the DyScGA. The DyScGA also has a built-in memory that retains the refined scale ranges and solution space over subsequent runs of the algorithms. Due to these features, the DyScGA stands apart from the existing GA enhancing strategies.

The dynamic scale genetic algorithm was implemented. Verification of the algorithm code was performed. The DyScGA was validated by demonstrating the improvement in performance over the simple genetic algorithm.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

The dynamic scale genetic algorithm was subjected to extensive testing by comparing its performance with that of the simple genetic algorithm on a testbed of problems. The testbed consisted of three discrete optimization problems with a linear objective function, each suitable for DyScGA. Two of these problems were listed in standard books on optimization (Gass 1985, Nemhauser and Wolsey 1988). The third problem considered in the testbed was the operations and support optimization problem being studied at the Vehicle Analysis Branch of LaRC. The size of the problems considered ranged from six to nineteen variables. The constraints of the problems included a wide variety: linearity and non-linearity, deterministic and stochastic, analytical formulation and lack of analytical formulation (predicted by simulation). Therefore, in terms of the size of the problems and nature of constraints, the testbed is considered to be adequately varied and representative of the general discrete optimization problem.

The dynamic scale genetic algorithm and the simple genetic algorithm are characterized by their stochastic nature and their dependence on control parameter settings. Hence any attempt at comparing the performance of these two algorithms needs to explicitly address these aspects. In this research, care was taken to assign identical random number generator seeds to both algorithms, during *each* experiment. Therefore, both DyScGA and the simple GA started out with identical populations for each individual run in each experiment. Furthermore, the same control parameter settings (crossover probability, mutation probability and population size) were used, thus making for a similar control environment for both algorithms, during each experiment. In addition, in order to perform a thorough investigation, testing was performed at various levels of the control parameters for at least one testbed problem. All of the above measures ensured an exact comparison between each experiment and each run of the DyScGA and the simple GA.

4.2 Problem I

Problem 3.2 which was presented in Chapter 3, formed Problem I of the testbed (Gass 1985). The original objective function of equation (3.2) was modified to accommodate penalties due to constraint violation. The penalty functions given by equation (3.3) were used. Testing on Problem I involved comparing the performance of the DyScGA and the simple GA at various levels of control parameters. Since the dynamic scale genetic algorithm does not alter the basic mechanism and fundamental properties of the simple GA, the theoretical basis of the latter still applies. Therefore, it is hypothesized that the control environment will affect the simple genetic algorithm and the dynamic scale genetic algorithm in the same manner. That is, the performance of both the simple GA and the DyScGA will simultaneously either improve or deteriorate as the levels of the control parameters change. For example, increasing the crossover rate would result in a performance improvement (or degradation) for both algorithms. This hypothesis was tested on Problem I by varying all the control parameters in a full factorial manner.

4.2.1 Dynamic scale and simple genetic algorithm at various control parameters

This section describes the experiments conducted to compare the performance of the two algorithms at different control parameter levels. The three control parameters of population size, crossover rate, and mutation rate were varied at two levels each in a full factorial manner. Commonly used control parameter levels were selected: population size 50, 100; crossover probability 0.9, 0.95; and mutation probability 0.1, 0.2 (De Jong 1975, Grefenstette 1986, Schaffer et al. 1989). The experimentation plan used appears in Table 4.1.

	Pop. size	Crossover prob.	Mutation prob.	DyScGA (10 gens, 100 replications)	GA (25 gens, 100 eplications)
Expt. 1	50	0.9	0.1	$\bar{x}_{DyScGA\,(50,0.9,0.1)}$ $s_{DyScGA\,(50,0.9,0.1)}$	$\bar{x}_{GA\,(50,0.9,0.1)}$ $s_{GA\,(50,0.9,0.1)}$
Expt. 2	50	0.9	0.2	$\bar{x}_{DyScGA\,(50,0.9,0.2)}$ $s_{DyScGA\,(50,0.9,0.2)}$	$\bar{x}_{GA\,(50,0.9,0.2)}$ $s_{GA\,(50,0.9,0.2)}$
Expt. 3	50	0.95	0.1	$\bar{x}_{DyScGA\,(50,0.95,0.1)}$ $s_{DyScGA\,(50,0.95,0.1)}$	$\bar{x}_{GA\,(50,0.95,0.1)}$ $s_{GA\,(50,0.95,0.1)}$
Expt. 4	50	0.95	0.2	$\bar{x}_{DyScGA\,(50,0.9,0.2)}$ $s_{DyScGA\,(50,0.9,0.2)}$	$\bar{x}_{GA\,(50,0.9,0.2)}$ $s_{GA\,(50,0.9,0.2)}$
Expt. 5	100	0.9	0.1	$\bar{x}_{DyScGA\,(100,0.9,0.1)}$ $s_{DyScGA\,(100,0.9,0.1)}$	$\bar{x}_{GA\,(100,0.9,0.1)}$ $s_{GA\,(100,0.9,0.1)}$
Expt. 6	100	0.9	0.2	$\bar{x}_{DyScGA\,(100,0.9,0.2)}$ $s_{DyScGA\,(100,0.9,0.2)}$	$\bar{x}_{GA\,(100,0.9,0.2)}$ $s_{GA\,(100,0.9,0.2)}$
Expt. 7	100	0.95	0.1	$\bar{x}_{DyScGA\,(100,0.95,0.1)}$ $s_{DyScGA\,(100,0.95,0.1)}$	$\bar{x}_{GA\,(100,0.95,0.1)}$ $s_{GA\,(100,0.95,0.1)}$
Expt. 8	100	0.95	0.2	$\bar{x}_{DyScGA\,(100,0.95,0.2)}$ $s_{DyScGA\,(100,0.95,0.2)}$	$\bar{x}_{GA\,(100,0.95,0.2)}$ $s_{GA\,(100,0.95,0.2)}$

Table 4.1 Problem I: Experimentation plan

Each experiment was replicated 100 times, starting with a new random number generator seed for both the simple and the dynamic scale genetic algorithms. This generated a statistically large sample size of 100. Experiments were conducted first on the DyScGA. In accordance with the plan outlined in Table 4.1, the DyScGA was run for eight experiments at 100 replications each, for ten generations. (Therefore a total of 800 experiments consisting of 10 generations each were run.) Each independent run was obtained by initializing the scale ranges of each variable to the limits specified in the original problem. The final solution obtained at the end of each replication of each experiment was used to compute the associated sample mean ($\bar{x}_{DyScGA}$), and standard deviation (s_{DyScGA}). Based on the outcomes of these experiments and replications, it was evident that an average of 14.8 population re-evaluations were performed by the DyScGA.

Next, experiments were performed on the simple GA. In order to compensate for the diversity-enhancing population re-evaluations performed by the DyScGA, the simple genetic algorithm was run for a higher number of generations. This turned out to be 25 generations (=10+14.8 generations) for this particular problem because of the 14.8 extra population evaluations required by the DyScGA. One hundred replications, starting with the same random number generator seeds as the DyScGA, for each of the eight experiments with the simple GA were conducted. Once again, the final solution obtained at the end of each replication of each experiment was used to compute the associated sample mean ($\bar{x}_{GA}$) and standard deviation (s_{GA}).

The sample mean and standard deviation values were utilized to test the following hypotheses *for each of the eight experiments*:

H_o: The performance of both the genetic algorithm and the dynamic scale genetic algorithm is equivalent (i.e., population mean of the solutions found by both techniques are not different $\mu_{GA} = \mu_{DyScGA}$).

H_1: The performance of the dynamic scale genetic algorithm is superior to that of the genetic algorithm (i.e., population mean of the solutions found by the genetic algorithm is higher than the population mean of the solutions found by the dynamic scale range genetic algorithm for a minimization problem $\mu_{GA} > \mu_{DyScGA}$).

The hypothesis tests were performed by using the z statistic for unknown variances and a large sample size. The z statistic was estimated by the following relation (4.1)

$$z = \frac{\bar{x}_{GA} - \bar{x}_{DyScGA}}{\sqrt{\frac{s_{GA}^2}{n} + \frac{s_{DyScGA}^2}{n}}} \tag{4.1}$$

where $\bar{x}_{GA}$ and $\bar{x}_{DyScGA}$ are the sample means of the final solutions, and s_{GA} and s_{DyScGA} are the sample standard deviations of the final solutions, obtained by the simple GA and the DyScGA through a large sample size ($n>30$). The resulting z statistics obtained for Problem I are tabulated in Table 4.2.

The high z statistic values in each row of Table 4.2 indicate that the null hypothesis H_0 can be rejected for each experiment at a very high confidence level (more than 99.9% confidence). Thus, regardless of the particular control parameter settings selected, the dynamic scale genetic algorithm outperforms the simple genetic algorithm. Additionally, it can be noted from the table that as we travel down the rows, the means and standard deviations obtained by both algorithms change in the same direction. Thus as the settings

Experiment No.. (Popln,Cross,Mutn)	DyScGA (10 gens) Mean $\bar{x}_{DyScGA}$, Std. Dev s_{DyScGA}	GA (25 gens) Mean $\bar{x}_{GA}$, Std. Dev s_{GA}	z statistic	p-value
Expt. 1 (50,0.9,0.1)	70.97, 10.86	105.92 27.43	12.54	< 0.001
Expt. 2 (50,0.9,0.2)	75.41, 13.72	128.11 33.25	14.98	< 0.001
Expt. 3 (50,0.95,0.1)	69.71, 9.66	108.96 29.77	13.64	< 0.001
Expt. 4 (50,0.95,0.2)	73.92, 13.46	134.53 35.67	16.93	< 0.001
Expt. 5 (100,0.9,0.1)	65.78, 6.30	89.24 15.50	13.49	< 0.001
Expt. 6 (100,0.9,0.2)	68.75, 7.68	120.28 28.43	19.57	< 0.001
Expt. 7 (100,0.95,0.1)	64.83, 5.48	90.18 18.00	14.19	< 0.001
Expt. 8 (100,0.95,0.2)	68.41, 8.69	118.08 27.48	17.87	< 0.001

Table 4.2 Problem I: z statistics for experiments of Table 4.1

of the control parameters change, the simple GA and the DyScGA either simultaneously improve or degrade in performance, implying that the control environment has a similar effect on their performances. It can be stated from these individual hypotheses tests at each experiment that the performance of the dynamic scale genetic algorithm is superior to the genetic algorithm regardless of the particular control parameters settings.

As an illustration, Figure 4.1 depicts a graphical comparison of the simple genetic algorithm and the dynamic scale genetic algorithm, with an identical initial population and control environment. This figure depicts an experiment conducted at the following control parameter levels: Population size 100, crossover probability 0.95, mutation probability 0.2. Since the same random number generator seed was used for both the DyScGA and the simple GA, both algorithms started with an identical initial population.

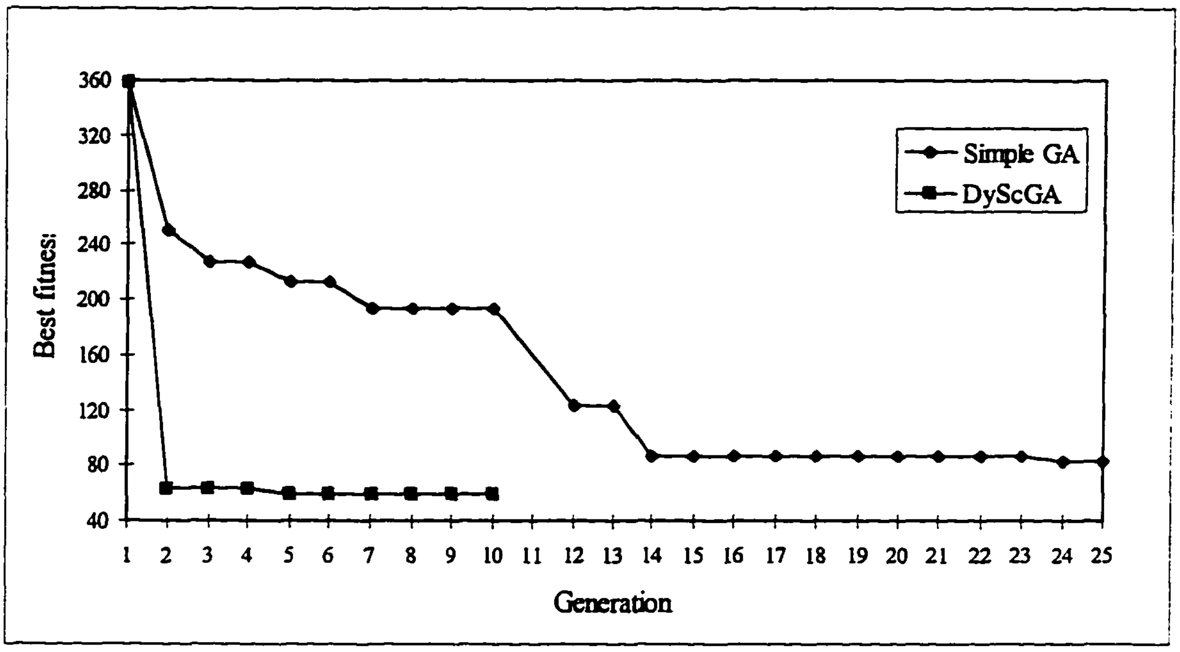


Figure 4.1 Problem I: Graphical comparison of the simple GA and the DyScGA

It can be observed from Figure 4.1 that, in the first generation, both had a best fitness of 359. However, in the second generation, DyScGA obtained a best fitness of 64 with five population re-evaluations, while the simple GA obtained a best fitness of 251. By the fifth generation, DyScGA achieved the global optimum of 60 at a cost of two more population re-evaluations. On the other hand, the simple GA terminated with a best fitness of 83 found in the 24th generation. Therefore, it is observed that the DyScGA found a *better* solution, and its computational requirements at five generations and seven population re-evaluations were *less* than the 24 generations of the simple GA.

In the 800 runs conducted, the dynamic scale genetic algorithm found the global optimum (60) 173 times, while the simple genetic algorithm found it only three times.

4.2.2 Testing for the Memory Effect of DyScGA

In the experiments of section 4.2.1, independent runs of the DyScGA were obtained by initializing each run with the original scale ranges of the problem, for statistical testing purposes. However, in reality, the DyScGA will be run so that the memory feature, which retains the scale ranges attained during the previous run comes into play. In this section, the outcomes of additional experiments that were performed to test this memory feature are described. During these experiments, the DyScGA with memory was compared to the simple GA and the DyScGA with its memory feature disabled. As before, the DyScGA with memory was run for one hundred trials. The control environment (population 100, crossover rate 0.95, mutation rate 0.2), and random number generator seeds were identical to those of the corresponding experiments of the simple GA and the DyScGA with memory disabled. Each run of the DyScGA (and DyScGA with memory disabled) consisted of 10 generations, lower than the 25 generations of the simple GA.

A histogram of the final solutions obtained by each algorithm is presented in Figure 4.2. It is observed that the performance of the DyScGA surpasses that of the simple GA and DyScGA with memory feature disabled.

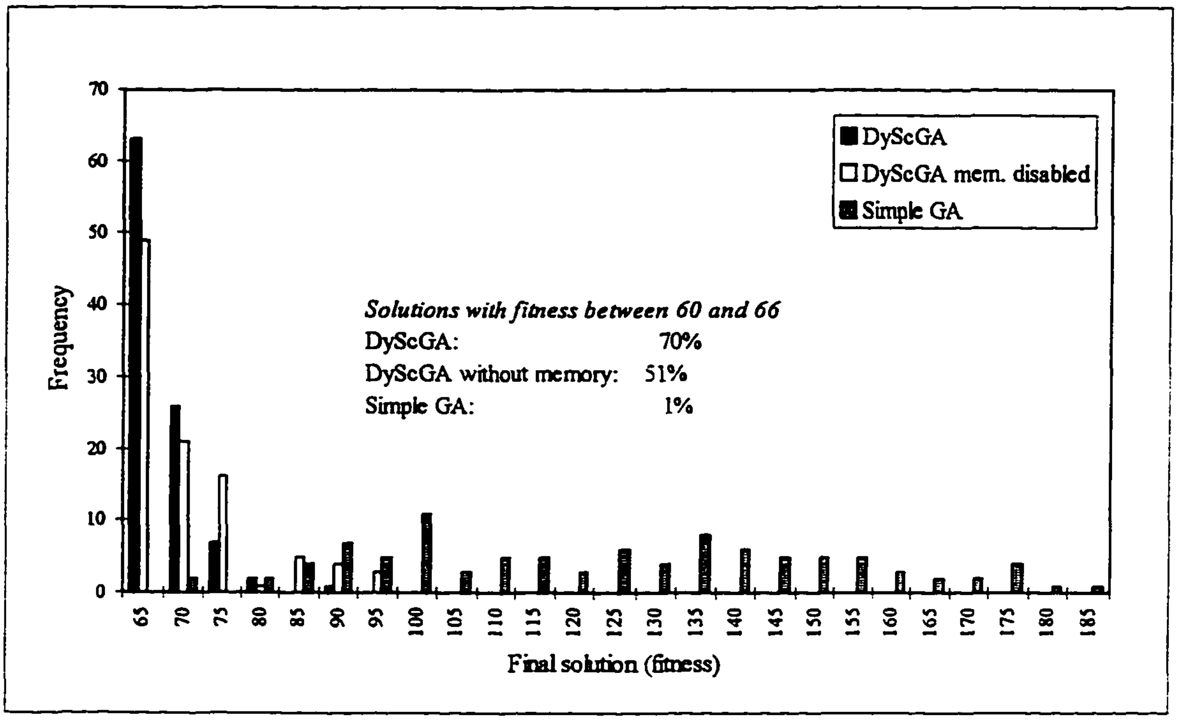


Figure 4.2 Problem I: Histogram of solutions

Seventy percent of the final solutions obtained by the runs of DyScGA were between the global optimum of 60 and 66. DyScGA found the global optimum a total of 28 times. The DyScGA with memory disabled resulted in a final solution between 60 and 66, 51% of the time. It found the global optimum a total of 23 times. Thus, it can be seen that when the memory of the DyScGA is disabled, its performance deteriorates. The simple GA resulted in just one solution between 60 and 66, which occurred at a fitness of 64. It did not find the global optimum of 60 in all of its 100 runs. It is apparent, therefore, that the memory feature enhances the performance of the DyScGA in a substantial manner.

4.3 Problem II

The second problem in the testbed is given by the following equation (4.2) (Nemhauser and Wolsey 1988):

$$\begin{array}{lllllllll}
\textit{Minimize} & x + & x_2 + & x_3 + & 2x_4 + & 0.5x_5 & & & \\
\textit{such that} & x_1 - & 2x_2 - & x_3 & & & & \leq & -2 \\
 & & 11x_2 & + & 3x_4 + & 2x_5 + & x_6 & \geq & 29 \\
 & - & 6x_2 - & 2x_3 & + & x_5 & & \leq & -9 \\
 & & & \multicolumn{4}{l}{0 \leq x_i \leq 99 \quad x = 1,\ldots,6} & & \\
 & \multicolumn{6}{l}{x_i \text{ are non-negative integers}} & &
\end{array} \qquad (4.2)$$

The penalty function was selected by running the simple genetic algorithm with several trial penalties. The penalty functions in case of constraint violation were:

$$\begin{array}{l}
10\,\{-2-(x_1-2x_2+x_3)\}^2 \\
10\,\{29-(11x_2+3x_4+2x_5+x_6)\}^2 \\
10\,\{-9-(-6x_2-2x_3+x_5)\}^2
\end{array} \qquad (4.3)$$

The problem was solved by the simple as well as dynamic scale genetic algorithm, at control parameter settings of population size 20, crossover probability 0.9 and mutation probability 0.1. The dynamic scale genetic algorithm was run for 10 generations, and used 100 replications starting with a new random number generator seed. Each independent run was obtained by initializing the scale ranges of variables to the limits set in the original problem. The final solution obtained at the end of each replication was used to compute the associated sample mean ($\bar{x}_{DyScGA}$) and standard deviation (s_{DyScGA}). Based on the extra evaluations required in each of the 100 replications, it was noted that an average of 18.3 extra population evaluations were performed. To compensate for the extra evaluations the simple genetic algorithm was run for 30 generations (more than 10+18.3). One hundred replications, starting with the same random number generator seeds as the DyScGA, were conducted. The final solution obtained at the end of each replication was used to compute the associated mean ($\bar{x}_{GA}$), and standard deviation (s_{GA}). The sample means and standard deviations were used to test the following hypothesis:

> H_o: The performance of both the simple genetic algorithm and the dynamic scale genetic algorithm is equivalent (i.e., the population mean of the final solutions found by both techniques are not different, $\mu_{GA} = \mu_{DyScGA}$).
>
> H_1: The performance of the dynamic scale genetic algorithm is superior to the performance of the simple genetic algorithm (i.e., the mean of the solutions found by the simple genetic algorithm is higher than the mean of the solutions found by the dynamic scale genetic algorithm, for a minimization problem, $\mu_{GA} > \mu_{DyScGA}$).

The z statistic was computed using the relation given in equation (4.1) as 5.427. Resulting statistics including sample mean and standard deviations and z statistic are given below.

DyScGA (10 gens)		GA (30 gens)			
Mean	Std. Dev	Mean	Std. Dev	z statistic	p-value
13.425	11.14	20.635	7.24	5.427	< 0.001

The high value of the z statistic (5.427), once again enables the rejection of the null hypothesis H_0 with 99.9% confidence. Thus, it can be stated that the performance of the dynamic scale genetic algorithm is superior to that of the genetic algorithm.

As an illustration, Figure 4.3 contains a graphical comparison of a single run of the DyScGA and the simple GA. Both the algorithms were initialized with the same random number generator seed and control environment. Therefore, the initial population for both was identical. It is observed that both algorithms started out with a best fitness of 148.5 in the first generation. However, DyScGA progressed to find a final solution of 20, in the 8th generation, which required 16 re-evaluations of the population. On the other hand, the simple GA found a higher solution of 32.5 in the 27th generation. Therefore, in this case, the DyScGA found a *better* solution, and its computational requirements at 10 generations and 16 population re-evaluations were *less* than the 27 generations of the simple GA.

In the experiments conducted above, the DyScGA found the best solution at a fitness of 7.5. Out of the one hundred runs, 18 runs of the DyScGA resulted in a solution of 7.5. For the simple GA, however, the best solution found was at a fitness of nine.

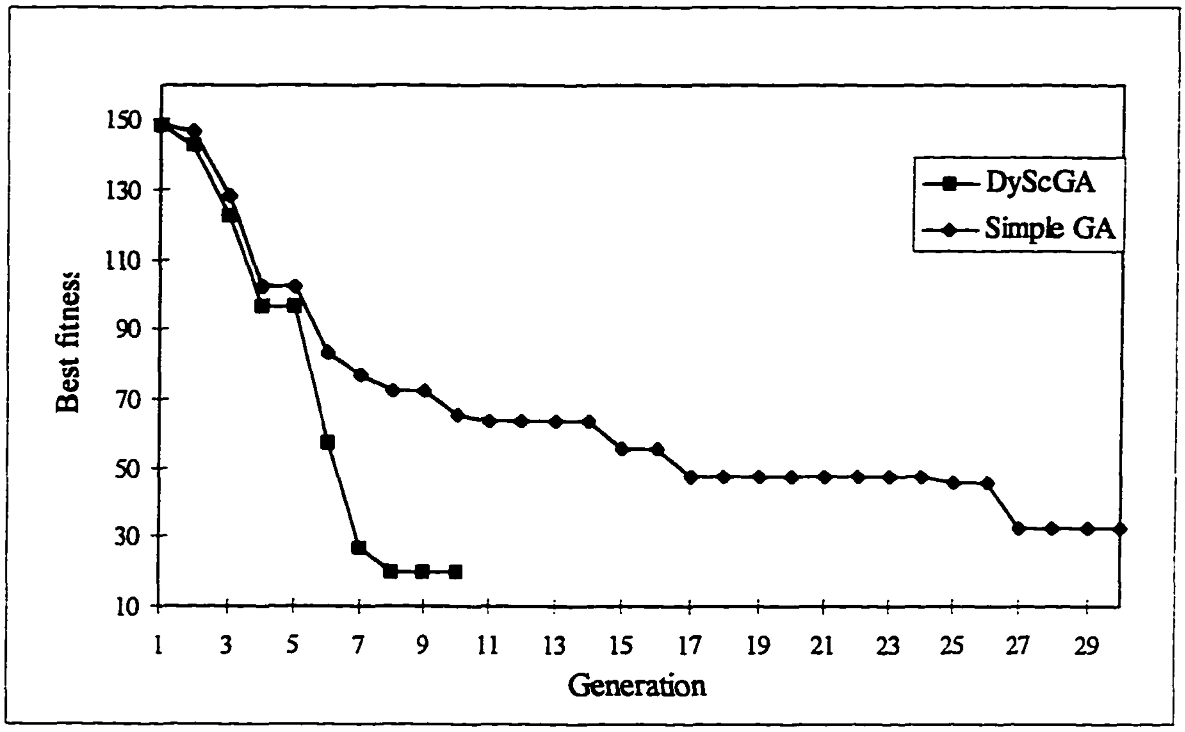


Figure 4.3 Problem II: Graphical comparison of the simple GA and the DyScGA

4.3.1 Testing for Memory Effect of DyScGA

The DyScGA in the preceding experiments consisted of independent runs in order to permit a statistical analysis. Independence was achieved by initializing each run of the DyScGA with the original scale ranges of the problem. In so doing the DyScGA was operated without the benefit of its memory feature. An additional aspect of testing was therefore to verify the merit of this memory feature. Additional experiments were conducted to test this feature, by comparing the performance of the DyScGA with memory to that of the simple GA and the DyScGA without the memory feature. Once again, 100 runs of each strategy were compared. The runs were conducted at conditions comparable to the experiments in section 4.3 (population size 20, crossover probability 0.9 and mutation probability 0.1). Random number generator seeds were identical to those used in the corresponding runs for the SGA and DyScGA with memory disabled of the preceding section. The DyScGA was run for 10 generations.

Figure 4.4 contains a histogram of the final solutions obtained by these experiments with DyScGA, as well as by the SGA and the DyScGA with memory feature disabled.

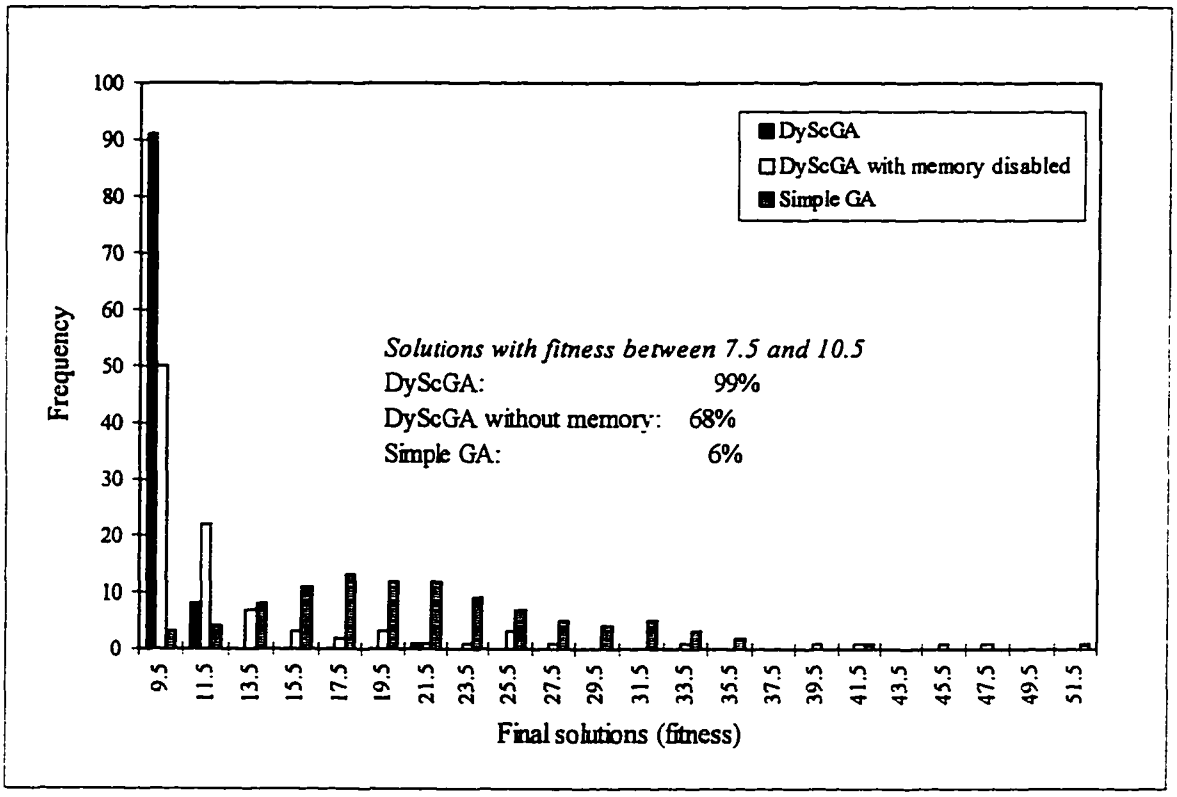


Figure 4.4 Problem II: Histogram of solutions

It can be observed that once again, the performance of the DyScGA surpasses that of the simple GA and the DyScGA with memory feature disabled. Of the 100 rus of DyScGA (with memory), 99 resulted in a solution between 7.5 and 10.5, with 25 of these solutions at 7.5. DyScGA with memory feature disabled, on the other hand, had 68% of its solutions between 7.5 and 10.5. Of these, 18 were at the optimum of 7.5. The simple GA had 6% of its solutions between 7.5 and 10.5, with none occurring at 7.5. Thus, it is apparent that the memory feature improves the quality of solutions found by the DyScGA.

4.4 Problem III: Operations and Support Simulation Optimization

This section describes the simulation optimization problem studied at NASA Langley Research Center. The problem involves the optimization of operation and support resources required for the maintenance of proposed spacecraft, through a discrete event simulation model. The simulation model was built using Simulation Language for Alternative Modeling (SLAM) (Pritsker 1984). It uses estimated values of component reliability and maintainability to simulate the preflight maintenance, the mission and the post flight maintenance (Ebeling and Donohue 1994).

Underlying processes such as component and system failure, repair and replacement times, and maintenance delays are simulated. Due to the random nature of these processes, the simulation model and its outputs are stochastic. For the purposes of simulation, maintenance, which includes scheduled and unscheduled activities, was divided into nine subsystems (power, structure, tanks, avionics, thermal, auxiliary, life support, mechanical and propulsion).

The user specified inputs to the simulation include the different crew and fleet sizes. A scheduled and unscheduled maintenance crew is assigned by the user at run-time, to each of the maintenance subsystems. (A fraction of the unscheduled crew is designated by the user to perform scheduled maintenance activities.) The number of vehicles employed or fleet size is also assigned by the user at run-time. Based on these user specified crew and fleet sizes, the simulation model predicts the successful missions flown and the mean vehicle turn-time between successive missions.

Thus, the model serves as a tool to observe how the decision variables (such as fleet and crew size) assigned by the user at run-time, affect the stochastic responses (such as mission rate and launch delays) for a particular space program. However, a problem faced by the LaRC engineers is that of determining the smallest fleet size and least manpower, that enables meeting the target mission rate in a timely manner. While the simulation model can predict the mission rate for a certain fleet and crew size, it cannot directly estimate the *least* fleet and *smallest* crew size to do so.

The problem was, therefore, to determine the least cost allocation of vehicles and manpower for a particular launch vehicle conceptual design that achieves the overall objectives of a space program. The minimization function was specified by LaRC engineers in terms of the relative cost attributed to the vehicles in the fleet and the maintenance manpower. The decision variables were non-zero, positive integers within a specified range. The constraints were specified in terms of the goals of a space program, as (i) The average mission launch delay is limited to a maximum of 48 hours, and (ii) The mean missions flown are at least 140 flights in five years. These constraints are dependent on random processes (such as component failures and repair times) and hence require stochastic evaluation through the simulation. The problem can be stated symbolically as optimizing over a discrete region $S \subset I^p$,

$$\min_{S \subset I^p} (X) = 100\,v + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 \in S$$

subject to

$E(delay) \leq 48$ *hours*		$E(suc\_mis) \geq 140$			(4.4)
$2 \leq v \leq 7$	$4 \leq c_1 \leq 9$	$6 \leq c_2 \leq 9$	$6 \leq c_3 \leq 12$	$4 \leq c_4 \leq 9$	
$15 \leq c_5 \leq 39$	$4 \leq c_6 \leq 10$	$5 \leq c_7 \leq 10$	$4 \leq c_8 \leq 10$	$8 \leq c_9 \leq 25$	
$3 \leq c_{1s} \leq 4$	$3 \leq c_{2s} \leq 6$	$3 \leq c_{3s} \leq 6$	$3 \leq c_{4s} \leq 4$	$7 \leq c_{5s} \leq 15$	
$3 \leq c_{6s} \leq 4$	$3 \leq c_{7s} \leq 5$	$3 \leq c_{8s} \leq 4$	$3 \leq c_{9s} \leq 8$		

where the non-negative integer variables are defined as:

p denotes the dimension of the discrete solution space, is 19
v denotes vehicles in the fleet
c_1 , c_{1s} denote unscheduled and scheduled crew assigned to power subsystem
c_2 , c_{2s} denote unscheduled and scheduled crew assigned to structure subsystem
c_3 , c_{3s} denote unscheduled and scheduled crew assigned to tanks subsystem
c_4 , c_{4s} denote unscheduled and scheduled crew assigned to avionics subsystem
c_5 , c_{5s} denote unscheduled and scheduled crew assigned to thermal subsystem
c_6 , c_{6s} denote unscheduled and scheduled crew assigned to auxiliary subsystem
c_7 , c_{7s} denote unscheduled and scheduled crew assigned to life support subsystem
c_8 , c_{8s} denote unscheduled and scheduled crew assigned to mechanical subsystem
c_9 , c_{9s} denote unscheduled and scheduled crew assigned to propulsion subsystem
E(delay), is a random variable representing the expected value of launch delay over the specified time horizon, as predicted by the simulation model
E(suc_mis), is a random variable representing the expected value of the number of missions completed successfully over the specified time horizon, as predicted by the simulation model

4.5 Simulation Optimization Framework

The average launch delay time and missions flown constraints are stochastic and observed through simulation. This means that each replication of the simulation has the potential to give rise to a varying average delay and successful missions flown. Stochastic measures or constraints require special consideration during optimization. In this research a unifying framework, based on well-established procedures in mathematical programming and statistics, was outlined and developed for the optimization of the simulated systems. Such a methodology has generally been lacking in the simulation optimization literature, which largely focuses on developing new approaches for optimization. The framework used in this research employs the chance constraint approach (Charnes and Cooper 1959) for treating stochastic constraints for the purpose of problem formulation. Standard statistical procedures recommended in the simulation literature are used to estimate the stochastic responses (Law and Kelton 1991, Pritsker 1984, Kleijnen 1987, Fishman 1978). These are based on the following assumptions:

1. The stochastic process is covariance stationary.
2. The sample variance is an unbiased estimator of the population variance.
3. The observations are independent and identically distributed.

The above assumptions do not always hold true in simulation studies. For instance, covariance stationarity may not rigorously hold for terminating simulations, unless the simulation time span is sufficiently long to warrant stationarity. (In terminating simulations, a simulation stops when a natural event signaling the end of the simulation run occurs. For instance, in the LaRC case, the simulation time span is specified as a five year period.) Since we do not necessarily run the terminating simulation until steady state, the underlying joint distributions of the random variables may change over time. Also, in order to generate independent and identically distributed observations, a true random number generator is required. However, in practice, pseudo-random number generators are used. Furthermore, Law and Kelton (1991) observe that simulation output data is usually correlated.

In practice, the above assumptions do not hold in the strictest sense and may be violated to a varying degree. However, due to a lack of alternative analyses methods for simulation data, it is recommended that standard statistical estimation be used regardless of slight violations (Law and Kelton 1991, Pritsker 1984, Kleijnen 1987, Fishman 1978). This practical strategy has been followed here.

The following sections describe an integrated and consistent simulation optimization framework.

4.5.1 Accuracy

One of the issues associated with a stochastic simulation is the accuracy with which a stochastic variable is to be estimated. The desired accuracy for each parameter can be specified by the decision maker in terms of statistical confidence intervals. Confidence intervals state the probability *$(1-\alpha)$* that the true mean is actually contained in an interval of width *(w)*, about the estimated mean.

When a simulation involves multiple stochastic variable, the overall confidence *$(1-\alpha)$* associated with an optimization study is based on satisfying the individual confidence intervals *$(1-\alpha_i)$* simultaneously. Thus the overall confidence satisfying the Bonferroni inequality given by equation (4.5) implies a lower overall accuracy:

$$P \geq 1 - \sum \alpha_i \tag{4.5}$$

For studies involving ten or less stochastic variables, if an overall confidence *$(1-\alpha)$* is desired, then the individual confidences *$(1-\alpha_i)$* can be selected by the following relation:

$$\sum \alpha_i = \alpha \tag{4.6}$$

However, for more than ten stochastic variables, the accuracy required of individual variables obtained by the Bonferroni inequality may be prohibitively high. For example, if an overall confidence of 90% is desired for a simulation involving ten variables, the individual confidences have to be at least 99%. (Similarly, if the individual confidence intervals of ten stochastic variables is 90%, the overall confidence is only greater than or equal to zero.) Therefore, for such cases, standard 90% or 95% individual confidence intervals are recommended (Law and Kelton 1991). The analysis results in such cases should be interpreted with caution, as one or more of the individual confidence intervals may not contain the corresponding true mean.

4.5.2. Replications

Once the accuracy for each stochastic variable is established, the number of replications or sample size required for the optimization study can be determined. The number of replications can be computed based on the specified confidence level *$(1-\alpha)$* of the true mean being within an interval *$\pm w$* of the estimated mean. A large sample size implies that the estimated mean is closer to the true mean, and hence increases the accuracy. A high level of accuracy is usually desired so that the results of the simulation study and hence subsequent decisions can be made with a satisfactory level of confidence.

However, due to the finite resource constraints (CPU time, time available for the simulation study), the number of replications that can be carried out are usually limited.

In order to estimate the replications, the following steps are undertaken (Law and Kelton 1991, Kleijnen 1987):

1. In an initial pilot experiment, the simulation model is run for 1000 or more replications to obtain a representative distribution for the stochastic responses.
2. The sample mean and standard deviations for the stochastic parameters are computed from the distributions obtained from the pilot experiment.
3. Based on the desired confidence or probability *(1-α)* that an interval (width ±*w* about the estimated mean) contains the true mean, the number of replications are determined using the following relation:

$$\left(\frac{z_{1-\alpha}s}{w}\right)^2 \tag{4.7}$$

where $z_{1-\alpha}$ represents the standard normal statistic covering an area *(1-α)*, and *s* is the standard deviation of the sample.

4.5.3. Chance Constraints

The variability inherent in stochastic constraints complicates the simulation optimization problem by forming fuzzy boundaries for the feasible region. This presents a danger of erroneously accepting a solution as feasible, while it may have a high probability of being infeasible, and vice versa. The chance constraint approach can be used to convert the stochastic constraints into deterministic constraints. Chance constraints (Charnes and

Cooper 1959) permit constraint violation up to a pre-specified probability limit. The decision maker expresses a risk tolerance, in terms of a permissible probability of constraint violation. Consider a stochastic constraint of the form $A(x) \leq b$, where $A(x)$ is a simulation response. Using the chance constraint approach, this can be reformulated in terms of risk tolerance as $P(A(x) > b) \leq \alpha$, where α denotes the extent to which constraint violation is permitted.

The chance constraints can be implemented through confidence interval estimates (Teleb and Azadivar 1994). We know that the confidence associated with an interval estimate denotes the probability that the interval contains the true parameter. For example, the upper limit associated with an interval of confidence $(1-\alpha)$ states that the probability that the true mean exceeds this limit is at most α. Thus, the upper and lower limits of the interval at the specified confidence $(1-\alpha)$ (or risk α) provide deterministic boundaries for the infeasible region. In this manner, confidence intervals provide a means of implementing chance constraints.

Using confidence intervals, the upper limit at confidence $(1-\alpha)$ can be used to denote the chance constraint $P(A(x) > b) \leq \alpha$, as $A(x)_{upp\_lim,\ \alpha} \leq b$. The confidence intervals are estimated by using the Student's t distribution in the standard manner:

$$Upp\_Lim = \bar{x} + \frac{(t_{(n-1),\alpha}\ s)}{\sqrt{n}}$$

$$Low\_Lim = \bar{x} - \frac{(t_{(n-1),\alpha}\ s)}{\sqrt{n}} \tag{4.8}$$

where n is the number of replications, $\bar{x}$ is the estimated mean, s is the standard deviation, and $t_{(n-1),\alpha}$ is obtained from the Student t distribution at $(n-1)$ degrees of freedom and α coverage.

The above confidence intervals based on Student's t are robust to minor deviations from normality. However, in cases of serious non-normality and very small sample size, the Johnson's modified $\bar{t}$ statistic for non-normal distributions is recommended (Johnson 1978, Kleijnen 1987). This adjusted statistic approximates the Student's t distribution by accounting for the skewness of the distribution, thus permitting its use for hypothesis testing and confidence intervals. Johnson's modified $\bar{t}$ statistic has been used successfully on distributions with varying degrees of non-normality, including the exponential distribution (Johnson 1978, Kleijnen 1986). The confidence intervals by the modified statistic are given by

$$Upp\_Lim = \bar{x} + \frac{(t_{(n-1),\alpha}\, s)}{\sqrt{n}} + \frac{\mu_3}{6s^2 n}$$

$$Low\_Lim = \bar{x} - \frac{(t_{(n-1),\alpha}\, s)}{\sqrt{n}} + \frac{\mu_3}{6s^2 n} \tag{4.9}$$

where μ_3 is the third central moment estimated in the standard manner:

$$\mu_3 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(x_i - \bar{x}\right)^3 \tag{4.10}$$

4.6 Problem III: Formulation

Based on the framework outlined in section 4.5, the NASA simulation optimization problem was formulated. The following steps were undertaken to achieve this.

Step 1.

The LaRC engineers were asked to state the desired accuracy in terms of confidence intervals. The desired accuracy was specified as follows:

	Desired width *±w*	Desired conf. *(1-α)*
Delay	± 48 hours	80% confidence
Successful Missions	± 2 missions	95% confidence

Table 4.3 Desired accuracy

This implies that the mean of the average delay is to be estimated within ± 2 days of the true mean, with an accuracy or confidence of 80%. Similarly, the mean successful missions flown are to be estimated within ± 2 missions, with a 95% confidence.

Step 2.

The individual confidence levels specified by the LaRC engineers were used to obtain the required sample size for the desired accuracy in estimating the individual means. The following steps were undertaken to compute the sample size (Law and Kelton 1991, Kleijnen 1987):

i) In a pilot experiment, the simulation model was run for 1000 replications to obtain a representative distribution for both the stochastic responses. The decision variables at this pilot experiment were selected by a process of trial-and-error, so as to yield a relatively wide distribution for the delay and successful missions variables. These levels are presented in Table 4.4.

Vehicles in fleet: 2

	Maintenance Subsystems								
Crew	*Powr*	*Struc*	*Tank*	*Avio*	*Thrm*	*Auxl*	*Life*	*Mech*	*Prop*
Unscheduled	5	6	7	4	22	6	4	7	8
Scheduled	3	5	4	3	11	3	4	4	6

Table 4.4 Decision variables at the pilot experiment

ii) The sample mean and standard deviation for both the stochastic parameters were computed from the distributions obtained from the pilot experiment as:

	Delay	Successful Missions
Mean $\bar{x}$	2.45 days	138.8 missions
Std. Deviation s	6.8 days	4.29 missions

Table 4.5 Statistics estimated from pilot experiment

iii) Based on the desired confidence or probability given in Table 4.3, and the statistics given in Table 4.5, the number of replications were determined as follows:

Delay	19 replications.
Successful missions	18 replications.

Based on these sample size estimations, a conservative sample size of 20 was selected.

Step 3.

For the present problem LaRC engineers were asked to express risk tolerances for the stochastic parameters. These were expressed as follows:

5% risk that mean of average delay exceeds 2 days.
5% risk that mean missions lag target of 140.

The above risk tolerances can be stated as:

$$\begin{array}{l} P[E(delay) > 48\ hours] \leq 0.05 \\ P[E(suc\_mis) < 140\ ] \leq 0.05 \end{array} \qquad (4.11)$$

Chance constraints were implemented through statistical interval estimates for a pre-specified confidence, by using the modified Student's *t* distribution given in Equation 4.8. The limits at the specified confidence (5% risk or 95% confidence) provide deterministic boundaries for the infeasible region, as follows:

$$\begin{array}{l} delay_{upp\_lim,0.5} \leq 48\ hours \\ suc\_mis_{low\_lim,.05} \geq 140 \end{array} \qquad (4.12)$$

Step 4.

Based on the above steps the LaRC simulation optimization problem was formulated as:

$$\min_{S \subset I^P} \quad (X) = 100\,v + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 \in S$$

subject to

$$\begin{array}{lllll} delay_{upp\_lim,0.5} & \leq 48\ hours & & suc\_mis_{low\_lim,.05} & \geq 140 \\ 2 \leq v \leq 7 & 4 \leq c_1 \leq 9 & 6 \leq c_2 \leq 9 & 6 \leq c_3 \leq 12 & 4 \leq c_4 \leq 9 \\ 15 \leq c_5 \leq 39 & 4 \leq c_6 \leq 10 & 5 \leq c_7 \leq 10 & 4 \leq c_8 \leq 10 & 8 \leq c_9 \leq 25 \\ 3 \leq c_{1s} \leq 4 & 3 \leq c_{2s} \leq 6 & 3 \leq c_{3s} \leq 6 & 3 \leq c_{4s} \leq 4 & 7 \leq c_{5s} \leq 15 \\ 3 \leq c_{6s} \leq 4 & 3 \leq c_{7s} \leq 5 & 3 \leq c_{8s} \leq 4 & 3 \leq c_{9s} \leq 8 & \end{array} \qquad (4.13)$$

all variables are non-negative integers

where,

mean average delay is estimated at 80% confidence of being within ± 2 days; and, mean successful missions are estimated at 95% confidence of being within ± 2 missions, giving an overall 75% accuracy for the optimization study.

4.7 Problem III: Experimental Results

The NASA LaRC simulation optimization problem was solved by both the simple genetic algorithm and the dynamic scale genetic algorithm, and their performance was compared statistically. The following penalties in case of constraint violation, determined experimentally on the simple genetic algorithm, were added to the objective function:

$$\begin{aligned}&1000*(48\ \text{hours} - \text{average delay}_{upp\_lim,\ 0.5})^2\\ &1000*(140 - \text{successful missions}_{low\_lim,\ 0.5})^2\end{aligned} \tag{4.14}$$

Control parameters were set as follows: population size 50, crossover probability 0.9, mutation probability 0.2. Each independent run of the dynamic scale genetic algorithm consisted of 10 generations. The DyScGA was run for 34 such independent runs, each run initialized with a different random number generator seed. The scale ranges for variables were set in accordance with the original problem for each independent run. The final solution (least cost allocation of vehicles and crew) obtained at the end of each of the 33 replications was used to compute the associated sample mean ($\bar{x}_{DyScGA}$) and standard deviation (s_{DyScGA}). On an average, the dynamic scale genetic algorithm took 5.18 extra population evaluations. To compensate for the extra population evaluations, the simple genetic algorithm was run for 16 generations (=10+5.18). The simple genetic algorithm was run for 33 independent runs, each run being initialized by different random number generator seeds. Once again, the final solutions obtained at the end of each of the 33 replications were used to compute the associated sample mean ($\bar{x}_{GA}$), and the standard deviation (s_{DyScGA}). The sample mean and standard deviation values were utilized to test the following hypothesis:

> H_o: The performance of the both the genetic algorithm and the dynamic scale genetic algorithm is equivalent (i.e., population mean of the solutions found by both techniques are not different $\mu_{GA} = \mu_{DyScGA}$,).
>
> H_I: The performance of the dynamic scale genetic algorithm is superior to that of the simple genetic algorithm (i.e., population mean of the solutions found by the simple GA is higher than the population mean of the solutions found by the dynamic scale genetic algorithm for a minimization problem $\mu_{GA} > \mu_{DyScGA}$,).

The hypothesis was tested by using the z statistic for unknown variances and large sample size, given by the relation (4.1). A *large* sample size of observations (33) for both the dynamic scale range genetic algorithm and the simple genetic algorithm was used. The following results were obtained:

DyScGA (10 gens)		GA (16 gens)			
Mean	Std. Dev	Mean	Std. Dev	z statistic	p-value
273.24	36.27	321.12	40.86	5.03	< 0.001

Based on the rather high z-statistic value of 5.03, the null hypothesis H_0 can be rejected with 99.9% confidence. Therefore, we reject the hypothesis that the dynamic scale genetic algorithm and the simple genetic algorithm are equivalent in performance.

As an illustration, Figures 4.5 (a), (b), (c) and (d) depict graphical comparisons of the DyScGA and the simple GA, initialized by different random number generator seeds.

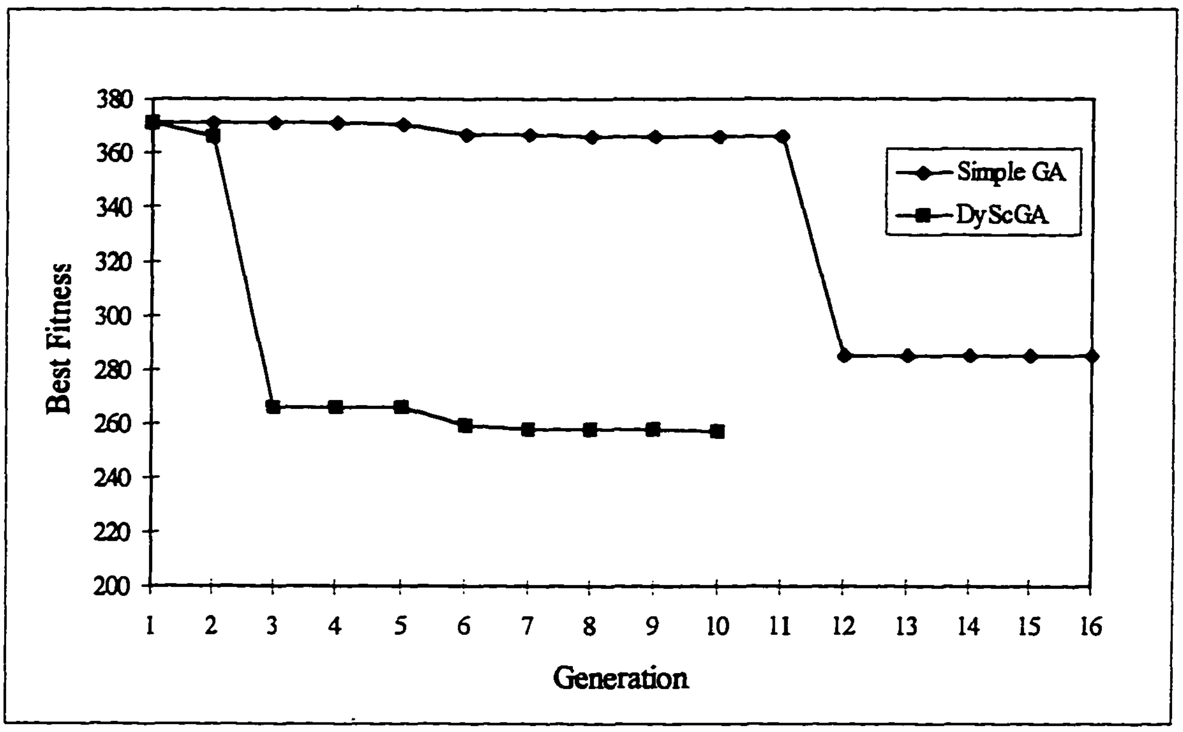


Figure 4.5a Problem III: Graphical comparison of the SGA and the DyScGA (Case i)

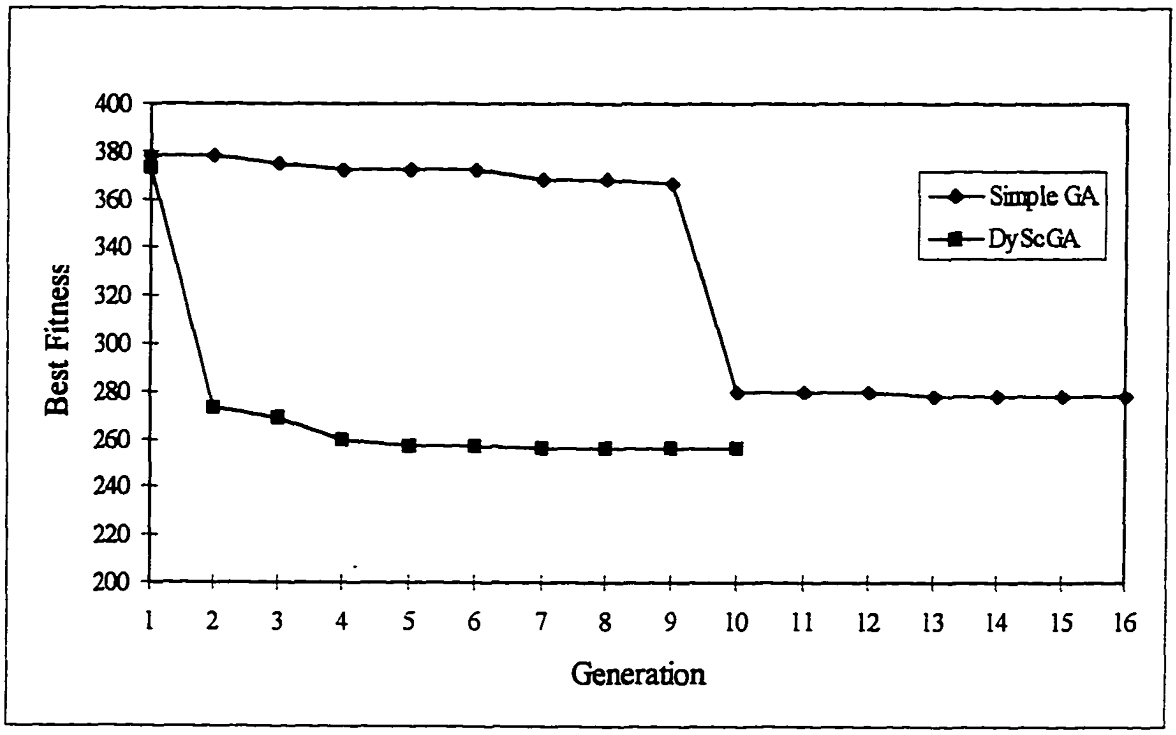


Figure 4.5b Problem III: Graphical comparison of the SGA and the DyScGA (Case ii)

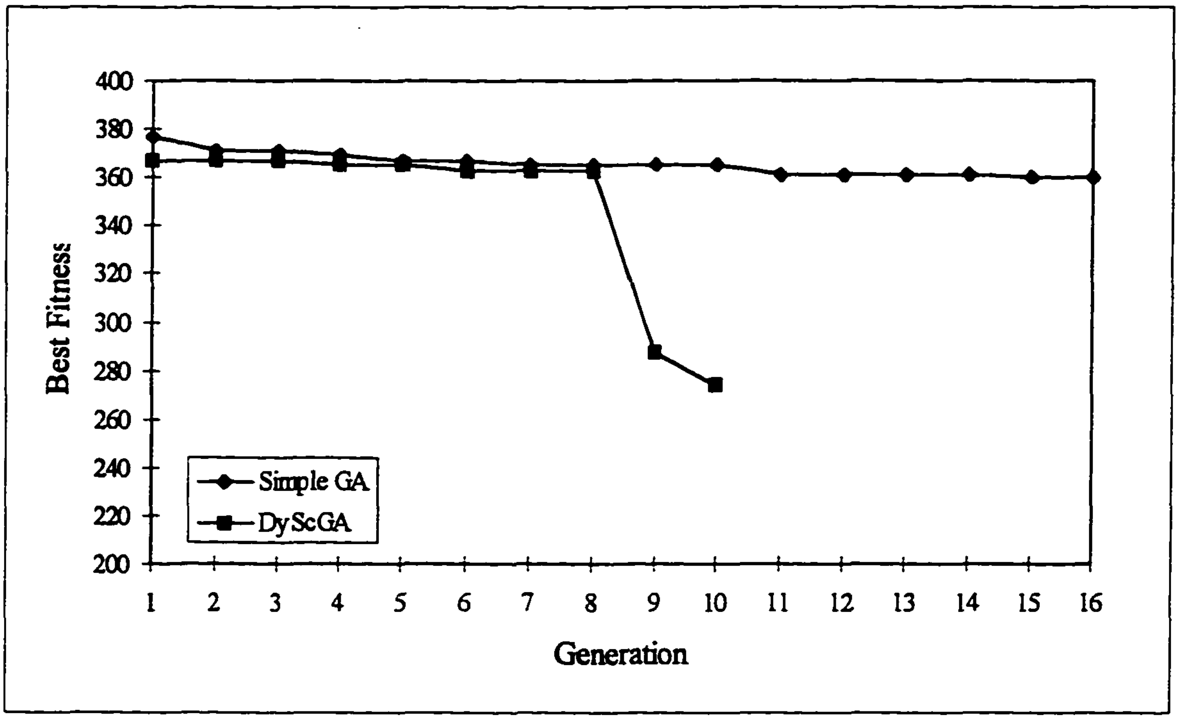


Figure 4.5c Problem III: Graphical comparison of the SGA and the DyScGA (Case iii)

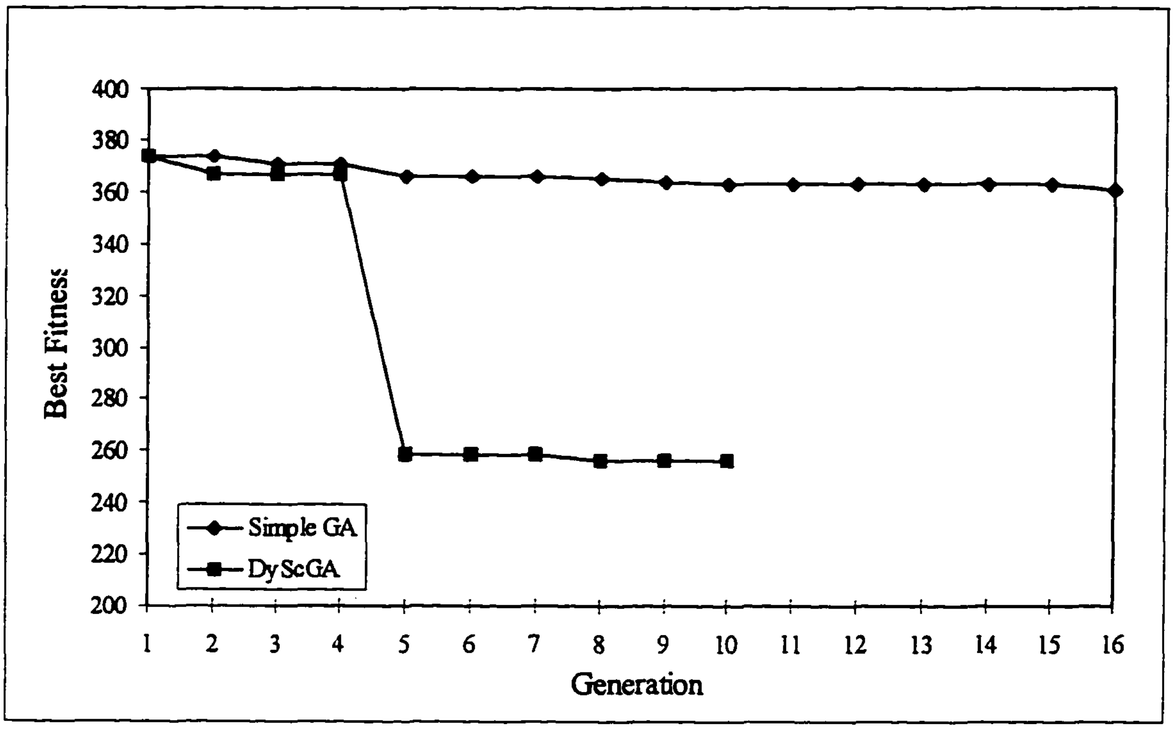


Figure 4.5d Problem III: Graphical comparison of the SGA and the DyScGA (Case iv)

Each figure contains runs obtained from an identical control environment and initialized by identical random number generator seeds. For instance, both the runs in Figure 4.5 (a) were initialized by a random number generator seed of 2.4. Therefore, the DyScGA and the simple GA both started with an identical initial population. However, the DyScGA very quickly progressed to a solution of 266 in the third generation, with 3 population re-evaluations. By the tenth generation, it produced a solution of 257 with 3 more re-evaluations. The simple GA, on the other hand, produced a final solution of 285 at the end of the twelfth generation. Figures 4.5 (b), (c) and (d), contain similar examples, where the DyScGA always produced better solutions than the simple GA. It is therefore demonstrated that the performance of the dynamic scale genetic algorithm on the NASA LaRC simulation optimization problem is superior.

4.7.1 Testing for Memory Effect

In the above section, the DyScGA was run by re-setting the scale ranges of variables to their original limits, so that the resulting independent runs would permit a statistical analysis. However, in reality, the memory feature of the DyScGA will retain the refinements made to the scale range. In this section, the benefits associated with the memory feature are verified. In order to verify the benefit associated with the memory feature, additional experiments were conducted. The DyScGA was run in order to retain learned information over subsequent runs. Thirty four such runs were conducted, with the same control parameters as the preceding section: population size 50, crossover probability 0.9, mutation probability 0.2. Each run was initialized with a different random number seed. Figure 4.6 contains a histogram of the outcomes of these experiments with the DyScGA, as well the simple GA and the DyScGA with memory disabled of the previous section. It is observed in Figure 4.6 that the majority of solutions (91.2%) produced by DyScGA resulted in a fitness between 256 and 265, with 85.3% of these at a fitness of 256. The DyScGA with memory disabled produced 73.5% solutions between 256 and 265, with 27.3% of these being at fitness 256. The simple GA, however, did not produce a single solution below a fitness of 273. Therefore, it is evident that the DyScGA produced better solutions than the simple GA, and the memory feature produced a dramatic improvement in its performance.

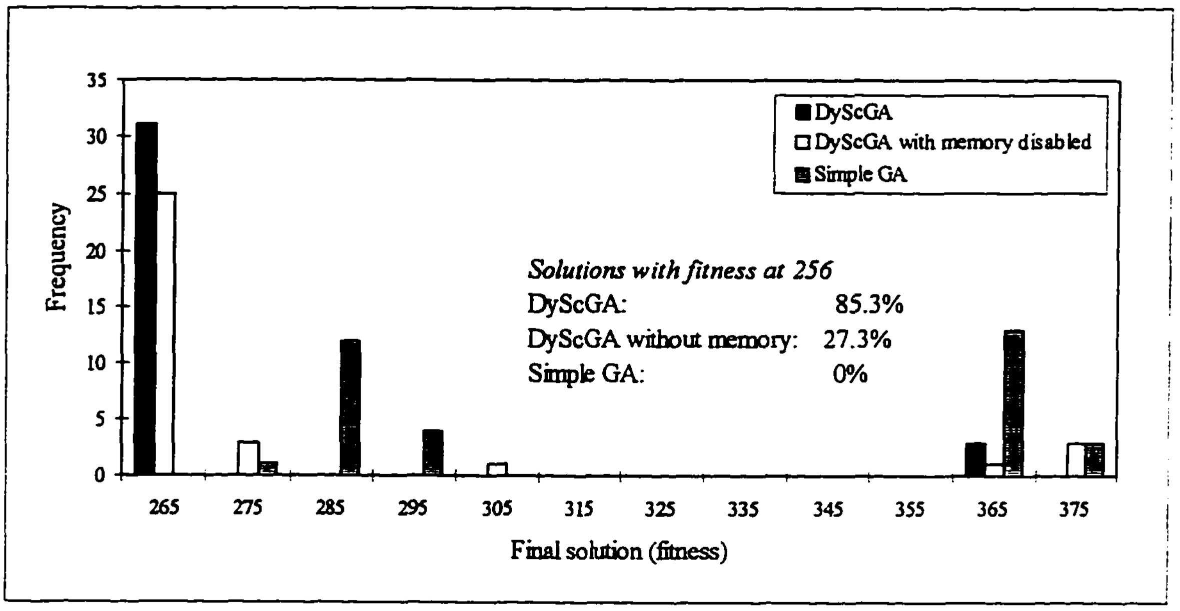


Figure 4.6 Problem III: Histogram of solutions

4.8 Problem III: Least Cost Allocation of Resources

The lowest cost allocation of operation and support resources for the NASA LaRC simulation optimization problem found by the simple GA was at 273. The least cost allocation found by the DyScGA was at cost 256. Several solutions with the fitness of 256 were produced by the DyScGA. These are given in Table 4.6. It is seen that the number of vehicles in the fleet (2) and the unscheduled crew allocation to the nine maintenance subsystems were constant. The scheduled crew assigned to the individual maintenance subsystems changed from solution to solution. The stochastic constraints (average delay not exceeding 48 hours and a total of at least 140 missions flown) were satisfied and well within their tolerance levels, as shown in the Table 4.6.

Although all the sets of crew and vehicle allocations in Table 4.6 had a cost of 256, the solution with the least average delay is preferred. The last row of Table 4.6 contains resource levels which yielded a mean of 2.3 hours average delay, with 95% confidence that the mean does not exceed 3.4 hours. Similarly, the target mission rate of 140 missions in a five year time span was achieved with 95% confidence. The above stochastic measures for the means of average delay and successful missions were estimated at the desired confidence specified by the LaRC engineers. The mean of average delay was estimated at an 80% confidence of being within ± 2 days of the true mean. The mean successful missions was estimated at a 95% confidence of being within ± 2 missions of the true mean. Thus the problem of optimizing the operations and support resources for future space vehicles using LaRC's discrete-event simulation model was successfully solved.

Flt. sze	Crew allocation for maintenance subsystems *U: Unscheduled crew S: Scheduled crew*																		Avg. delay (hours)		Missions	
	Powr		*Struc*		*Tank*		*Avio*		*Thrm*		*Auxl*		*Life*		*Mech*		*Prop*		*Mean*	*U.lim*	*Mean*	*L.lim*
	U	*S*	*U*	*S*	*U*	*S*	*U*	*S*	*U*	*S*	*U*	*S*	*U*	*S*	*U*	*S*	*U*	*S*				
2	4	3	6	3	6	3	4	3	15	11	4	3	5	4	4	3	8	6	7.4	13.9	140	140
2	4	3	6	5	6	5	4	3	15	10	4	3	5	4	4	3	8	3	7.4	10.1	140	140
2	4	3	6	5	6	3	4	3	15	8	4	3	5	4	4	3	8	7	5.1	7.4	140	140
2	4	3	6	4	6	3	4	3	15	9	4	3	5	4	4	3	8	5	2.3	3.4	140	140

Table 4.6 Solutions produced by DyScGA at fitness 256

4.9 Summary

The proposed DyScGA, based on the idea that the performance of the simple GA can be improved by refining a problem's solution space through successive generations, was subjected to extensive testing. A testbed consisting of problems that represent a range of discrete optimization problems was selected The DyScGA and the simple GA were both used to solve the testbed problems, and their performance was compared statistically. Care was taken to use identical control parameter settings and random number generator seeds, so that an exact comparison between runs of the two algorithms could be made. Based on these experiments, it was demonstrated that the performance of the DyScGA surpassed that of the simple GA on the problems considered. The benefit associated with the in-built memory feature of the dynamic scale genetic algorithm was also validated.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH

5.1 Introduction

The main motivation behind this research was to develop an efficient genetic algorithm based methodology to minimize the operation and support resources for reusable launch vehicles through simulation models. In this dissertation, the simple genetic algorithm was enhanced to provide for a more effective and efficient search in conjunction with discrete optimization. Specifically, the enhanced genetic search was designed for constrained discrete optimization problems with a linear objective function. This class of problems is the most commonly occurring integer optimization problem in industry and practical applications. Hence, the proposed dynamic scale genetic algorithm is widely applicable for discrete optimization. In addressing the main objective, issues related to the optimization of stochastic simulated systems, in general, were also identified and studied. A unifying framework for the optimization of simulated studies was presented based on existing statistical estimation techniques and mathematical programming approaches. The NASA LaRC operation and support resource optimization problem was addressed by using the dynamic scale genetic algorithm in conjunction with the above simulation optimization framework. In this chapter the main accomplishments of this research are highlighted, followed by a discussion of future research.

5.2 Dynamic Scale Genetic Algorithm

This research developed an enhanced genetic algorithm, the dynamic scale genetic algorithm, for constrained discrete optimization problems with linear objective functions. Based on the concepts of implicit enumeration, the DyScGA utilizes problem specific information to successively refine the search space and improve the effectiveness of genetic search. The DyScGA is not the first search-space-scaling genetic algorithm proposed in the literature. The dynamic parameter encoding, delta coding, adaptive search space scaling and adaptive representation genetic optimizer strategies all refine the search space in order to improve the genetic search. However, since these other methods use population convergence measurements to trigger search space pruning, there exists the possibility that the portion that contains the optimum is trimmed off accidentally. In addition, they do not have a built-in memory feature that retains the boundaries of the refined search space over subsequent applications of the GA. The DyScGA overcomes these basic deficiencies inherent in the other search-space-scaling strategies . Specifically, the following features of the DyScGA make it an attractive search strategy:

1. Unlike other proposed modifications, DyScGA refines the search space if and only if there is mathematical evidence that the discarded portion does not contain the optimum.
2. Therefore an 'inverse' prune operator to recover a discarded portion of the search space, such as required by the other proposed techniques, is not necessary.
3. Unlike the other methods, the user does not have to arbitrarily set additional control parameters during run-time. Therefore, its performance is consistent.

4. Has a built-in memory feature that retains the new refined boundaries of the search space across successive applications of the algorithm, unlike the other proposed enhancements.
5. Enhances diversity of the population by re-evaluating candidates with a new mapping strategy.

Experiments were conducted to statistically test the effectiveness of the dynamic scale genetic algorithm by comparing its performance to that of the simple genetic algorithm on a testbed of three problems. The results of the experiments clearly indicate that the DyScGA produced *better* solutions in *less* generations. The performance of the dynamic scale genetic algorithm was superior in all the experiments, with the z statistic ranging in value from 5.03 to 19.57, at a confidence level of over 99.9%. Table 5.1 contains a comparison of the quality of solutions obtained by both the algorithms.

	Best solution	Percentage of runs that produced best soln.	
		DyScGA	Simple GA
Problem I *(100,0.95,0.2)*	60	28%	0%
Problem II *(20,0.9,0.1)*	7.5	25%	0%
Problem III *(100,0.95,0.2)*	256	85.3%	0%

Figures in parenthesis indicate the settings of the control environment.

Table 5.1 Comparison of solutions obtained by the simple GA and the DyScGA

It is evident from the above table that the DyScGA consistently found a better solution than the simple GA.

5.3 Simulation Optimization Framework

With the growing incidence of simulation modeling in industry, it is essential to extend the role of traditional optimization to include the simulation domain. Some of the issues associated with a stochastic simulation optimization study that require special consideration are: (i) desired accuracy of the study, (ii) number of replications required for the study, and (iii) treating stochastic constraints. In this research, a statistically sound and consistent framework for optimization of simulation studies that addresses the above aspects was presented. It is based on standard statistical estimation and mathematical programming techniques.

Under this framework, the desired accuracy for the optimization study is specified by the decision maker in terms of statistical confidence intervals associated with each stochastic parameter or variable. The replications required to estimate the stochastic parameters with the pre-specified confidence intervals are then computed. The optimization study is subsequently carried out by replicating each simulation run with a new random number generator seed. Another aspect that complicates the simulation optimization problem is the stochastic nature of the constraints. Stochastic constraints present fuzzy boundaries between feasible and infeasible regions of the solution space. Thus there is a danger of erroneously accepting a solution as feasible when it may have a high probability of being infeasible, vice versa. The mathematical programming approach of chance constraints is used to convert the stochastic constraints to deterministic constraints. This approach requires the decision maker to specify a 'risk tolerance' or a permissible probability of constraint violation. Statistical confidence intervals provide a means of implementing the chance constraints.

Use of this framework ensures that the optimization study is conducted in a consistent and statistically sound manner. Specifically, the following objectives are achieved:

1. The stochastic parameters predicted by the simulation are estimated with the desired confidence specified by the decision maker.
2. The fuzzy boundaries provided by stochastic constraints are converted to deterministic boundaries. The violation of constraints is limited to the risk tolerance specified by the decision maker.
3. The inferences drawn from the study and subsequent decisions are based on statistical levels of confidence.

5.4 Operation and Support Problem

The NASA LaRC simulation optimization problem involving minimization of operation and support activities of reusable launch vehicles was addressed by using the dynamic scale genetic algorithm in conjunction with the framework outlined above. The best solution found by the dynamic scale genetic algorithm occurred at a cost (or utility) of 256. (The best solution found by the simple genetic algorithm occurred at a cost or utility of 273). There were several solutions with an objective function value of 256. The solution with the least average delay was selected from among these as having a superior relative utility. At this solution, the operation and support resources were allocated as follows:

Fleet size: 2 vehicles

	Maintenance Sub-System								
Crew	*Powr.*	*Struc.*	*Tank*	*Avio.*	*Thrm.*	*Auxl.*	*Life.*	*Mech.*	*Prop.*
Unsched	4	6	6	4	15	4	5	4	8
Sched.	3	4	3	3	9	3	4	3	5

The stochastic parameter of average launch delay was estimated within ± 2 days of the true mean with an accuracy or confidence of 80%. Similarly successful missions were estimated within ±2 missions with a 95% confidence. The constraints were well within their target levels. The mean of average launch delay was estimated at 2.3 hours, with a 95% confidence that it does not exceed 3.4 hours, well below the permissible delay of 48 hours. The expected number of successful missions was estimated with 95% achievement of the target mission rate of 140 missions in five years. The overall accuracy of the study was 75%.

The above solution represents a 23% improvement over the previous optimization approach followed at NASA LaRC. This was the one-variable-at-a-time approach, which involved varying the levels. of the input parameters of the simulation manually, one at a time, until a desired change in the output of the simulation was obtained. This approach resulted in a solution of utility 353, including a fleet size of three vehicles and a total of 53 maintenance crew. The approach was tedious to use and involved a lot of guess-work. Furthermore, the optimization study itself was carried out by ignoring its stochastic nature.

5.5 Future Research

The methodology developed in this research can be extended and applied to other applications and problems. Opportunities for future research include the following.

1. The DyScGA can be applied in conjunction with the simulation optimization framework to other operation and support simulation models that are in use at NASA LaRC.

2. The DyScGA can also be applied to various other situations involving discrete optimization problems with linear objective functions. The car rental agency and the commercial airline industry are two such examples. The car rental agency needs to maintain a fleet of cars to meet customer demand. Similarly, an airline is required to maintain a fleet of airplanes in order to meet a specific schedule of flights. The scope of these problems need not be limited to the simulation domain, but can include the non-simulation domain.

3. The research can be extended to perform cost optimization of operation and support resources for launch vehicles. Appropriate cost models would need to be developed for these resources. The cost models could be of the form of cost estimating

relations obtained through a statistical analysis of historical launch data. The dynamic scale genetic algorithm could then be applied to minimize the cost of operation and support resources.

4. The research can also be extended to multi-disciplinary design optimization. The cost models so developed could be integrated with other disciplinary models, in order to achieve a systems level multi-disciplinary design optimization of launch vehicles. This will enable an analysis of the life cycle cost of launch vehicles during the conceptual design phase.

REFERENCES

Brearly, A.L., G. Mitra and H. P. Williams, "An Analysis of Mathematical Programming Problems Prior to Applying the Simplex Method', *Mathematical Programming*, 8 (1975), 54-83.

Charnes A. and W. W. Cooper, "Chance Constrained Programming" *Management Science*, 6, 1 (1959), 73-79.

Ebeling, C. E., and C.B. Donohoe, "Integrating O&S Models During Conceptual Design - Part III", Annual NASA Report, Grant No. NAG1-1-1327, 1994.

Elketroussi, M., and D. P. Fan, "Optimization of simulation models with GADELO: a multi-population genetic algorithm", *International Journal of Bio-Medical Computing*, 35, 1 (1994), 61-77.

Davis, L., *Genetic Algorithms and Simulated Annealing*, Pitman, N.Y., 1987.

De Jong, K., "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", Ph.D. dissertation, University of Michigan, Ann Arbor, 1975.

Fabrycky, W.J. and B.S. Blanchard, *Life Cycle Cost and Economic Analysis*, Prentice Hall, New Jersey, 1991.

Farrell, W., "Literature Review and Bibliography of Simulation Optimization", in *Proceedings of the 1977 Winter Simulation Conference*, 1977, 117-124.

Fishman, G., *Principles of Discrete Event Simulation*, John Wiley, New York, 1978.

Friedman, M., and L.G. Savage, "Multiparametric Linear Programming", in , C. Gal, T. and J. Nedoma (Eds.), *Techniques of Statistical Analysis, Management Science*, 18, 7 (1972), 406-421.

Fu, M.C., "Optimization Via Simulation: A Review", *Annals of Operations Research*, 53, (1994), 199-247.

Gage, P. J., "New Approaches to Optimization in Aerospace Conceptual Design", NASA Report 196695, Contract Number NAG2-640, 1995.

Garey, M.R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, N.Y., 1983.

Gass, S. I., *Linear Programming - Methods and Applications*, 5th Edition, Mc-Graw-Hill, New York, 1985.

Glover, F., “Tabu Search -- Part I”, *ORSA Journal on Computing*, 1 (1989), 190-206.

Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, PA, 1989.

Grefenstette, J. J., "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man and Cybernetics*, 1 (1986), 122-128.

Grefenstette, J. J., C.L. Ramsey, and A. Schultz, "Learning sequential decision rules using simulation models and competition", *Machine Learning*, 5 (1990), 355-381.

Griffin, J.J, "Whole Life Cost Studies: A Defense Management Perspective", *Engineering Costs and Production Economics*, 14 (1988), 107-115.

Holland, J., *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

Holland, J.H., "Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems", in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, 2, Morgan Kaufmann, Los Altos, CA, 1986, 593-624

Hooke, R., and T.A. Jeeves, "A Direct Search Solution of Numerical and Statistical Problems", *Journal of Association for Computing Machinery*, 8 (1961), 212-229.

Jacobson, S.H., and L.W. Schruben, "Techniques for Simulation Response Optimization", *Operation Research Letters*, 8 (1989), 1-9.

Johnson, M., "Modified *t* Tests and Confidence Intervals for Asymmetrical Populations", *Journal of the American Statistical Association*, 73 (1978), 536-544.

Kleijnen, J., G. Kloppenburg and F. Meeuwsen, "Testing the Means of an Asymmetric Population: Johnson's Modified t Test Revisited", *Communications in Statistics, Simulation and Computation*, 15, 3 (1986), 715-732.

Kleijnen, J., *Statistical Tools for Simulation Practitioners*, Marcel Dekker, New York, 1987.

Law, A., and W. D. Kelton, *Simulation Modeling and Analysis*, Mc-Graw Hill, New York, 1991.

Mandava, V.R., J.M. Fitzpatrick and D.R. Pickens, III, "Adpative Search Space Scaling in Digital Image Registration", *IEEE Transactions on Medical Imaging,* 8, 3 (1989) 251-262.

Michalewicz, Z., "A Hierarchy of Evolution Programs: An Experimental Study", *Evolutionary Computing,* 1, 1 (1993), 51-76.

Michalewicz, Z. and J. Arabas, "Genetic Algorithms for 0/1 Knapsack Problem", *Lecture Notes in Computer Science,* 869 (1994), 135-143.

Morris, W.E., T.A. Talay, and D.G. Eide, "Operations Simulation for the Design of a Future Space Transportation System", in *Proceedings of the AIAA 21st Aerospace Sciences Meeting,* 1983.

Morris, W.D., N.H. White, W.T. Davis, and C.E. Ebeling, "Defining Support Requirements During Conceptual Design of Reusable Launch Vehicles", in *Proceedings of the AIAA 1995 Space Programs and Technologies Conference,* 1995.

Nelder, J.A., and R. Mead, "A Simplex Method for Function Minimization", *Computer Journal,* 7 (1965) 308-313.

Nemhauser, G. L. and L. A. Wolsey, *Integer and Combinatorial Optimization,* John Wiley and Sons, 1988.

Nutter, T., and Y. Ding, "Bridging the Gap: Combining High and Low Level Representations for Knowledge Retention with Genetic Algorithms", *International Journal of Expert Systems,* 4, 3 (1992), 249-280.

Papadimitriou, C.H., and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity,* Prentice-Hall, New Jersey, 1982.

Parker, R. G., and R. L. Rardin, *Discrete Optimization,* Academic Press, Inc., New York, 1988.

Pritsker, A.A.B., *Introduction to Simulation and SLAM II,* Systems Publishing, West Lafayette, IN, 1984.

Powell, D.J, S.S. Tong, M.M. Skolnick, "EnGENEous domain independent, machine learning for design optimization", in J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms,* 1989, 151-159.

Schaffer, J.D., R. Caruana, L. Eshelman and R. Das, "A Study of Control Parameters Affecting On-Line Performance of Genetic Algorithms for Function Optimization", in J.D. Schaffer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms,* 1989, 50-61.

Schraudolph, N.N., and R. K. Belew, "Dynamic Parameter Encoding for Genetic Algorithms", *Machine Learning,* 9, 1 (1992), 9-21.

Shaefer, C., "The ARGOT Strategy: Adaptive Representation Genetic Optimization Technique", in J. Grefenstette, (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms,* Lawrence Erlbaum, N.J., Cambridge, MA, 1987.

Smith, D. E., "An Empirical Investigation of Optimum-Seeking in the Computer Simulation Situation", *Operations Research,* 21 (1973), 475-497.

Spendley, W., G.R. Hext, and F.R. Himsworth, "Sequential application of simplex designs in optimization and evolutionary operations", *Technometrics,* 4 (1962), 441-461.

Teleb, R., and F. Azadivar, "A Methodology for solving multi-objective simulation-optimization problems", *European Journal of Operational Research,* 72 (1994), 135-145.

Tompkins, G. and R. Azadivar, “Genetic Algorithms in Optimizing Simulated Systems’, in C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman (Eds.), *Proceedings of the 1995 Winter Simulation Conference,* 1995.

Unal, R., E. B. Dean and A. A. Moore, "Space Transportation System Operations and Support Cost Modelling Approach", *Journal of Parametrics,* X, 4 (1990), 35-51.

Whitley, D, K. Mathias and P. Fitzhorn, "Delta Coding: An Iterative Search Strategy for Genetic Algorithms", in R.K. Belew and L.B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, 1991.

Whitley, D. and T. Starkweather, "GENITOR II: A Distributed Genetic Algorithm", *Journal of Experimental and Theoretical Artificial Intelligence,* 2 (1990), 189-214.

Yunker, J. M. and J. D. Tew, "Simulation optimization by genetic search", *Mathematics and Computers in Simulation,* 37, 1 (1994), 17-28.

Zhou, H.H., "CSM: A computational model of cumulative learning", *Machine Learning,* 5 (1990), 383-406.

VITA

Bela Dange Joshi was born on March 28, 1966 in Bombay, India. In 1987 she received a Bachelor's degree in Electrical Engineering from the Government College of Engineering, Pune. After working for a few years in India as an Electrical Engineer, Bela returned to education. In 1992 she graduated from the University of Tennessee at Knoxville with a Master of Science degree in Engineering Science with an emphasis in Artificial Intelligence. In the spring of 1993 she joined Old Dominion University to pursue a doctoral degree in Engineering Management. Her NASA funded doctoral research involved the optimization of operations resources of launch vehicles through stochastic discrete-event simulation modeling. She graduated with a Ph.D. in 1996. Her research interests are in the area of simulation, optimization and robust engineering.