1991

A genetic algorithm approach to optimization of asynchronous automatic assembly systems

Mark A. Wellman
Iowa State University

A genetic algorithm approach to optimization of asynchronous automatic assembly systems

by

Mark A. Wellman

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Departments: Industrial and Manufacturing Systems Engineering
Statistics
Major: Operations Research

Iowa State University
Ames, Iowa

1991

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

I would like thank Dr. Doug Gemmill for his continued support and helpful insight throughout this thesis project. His comments and encouragements were greatly appreciated.

I would also like to thank my parents for their undying support throughout the years. You were always there when I needed an encouraging word or when I just needed someone to put things back into proper perspective. You have given me much, for which I will be eternally grateful.

I would especially like to thank my loving wife, Sue. You were there whenever I needed your support. I thank you for enduring the many months which were filled with work on this thesis.

I. INTRODUCTION

Assembly processes are among the most important in a manufacturing facility. The costs associated with assembly operations can often account for more than 50% of the finished product [Boothroyd et al., 1982]. It has been estimated [Captor et al., 1983] that batch manufacturing accounts for approximately 75% of all manufacturing in the United States and that this process accounts for nearly 22% of the U.S. Gross National Product. Clearly, the assembly process plays an important role in manufacturing and the entire national economy.

A. Assembly Systems

From a managerial standpoint, assembly systems can have a large impact upon profitability and competitiveness. As industry is reducing production times in order to reduce lead-times of product deliveries, optimization of assembly processes is a primary concern. The most common objectives are to minimize the additional cost per part (attributed to assembly operations) or to maximize the efficiency of the system. With a real system the costs of floor space, labor, additional stations, and transport pallets can be estimated and a system configured on the basis of economic payback. Since we are studying a representative hypothetical system, the objective will be to maximize the efficiency, or throughput, of the system. Throughput is defined by marking a station which corresponds to the unload operation of the system and observing the rate of output.

At present we can surmise that an automatic assembly system (AAS) is merely a queuing system. An AAS, like any basic queuing system, consists of three basic elements: customers (parts), servers (workstations), and randomness. The randomness can be seen by way of variable service times, unreliable service stations, or variable arrival rates to the system.

An automatic assembly system can be manual or automatic, and can be synchronous or asynchronous. A manual system is one in which the service is completed by human operators. An automatic system, in contrast, service is completed by robots or automatic workheads. Synchronous systems are those in which the production rate is fixed by the transport mechanism. All parts move from station to station at the same time, thus all stations have the identical service time. Asynchronous systems allow the parts to move to the next station upon completion of the service. This allows the system some autonomy since when there is a work stoppage at one station, the remaining stations can continue processing. When a work stoppage occurs in a synchronous system, however, the entire system waits until the stoppage is corrected.

We can also classify assembly systems according to their topology. An open system (Figure 1) involves parts arriving at one end of the system, progressing through a series of workstations, and departing the system from the other end. A closed system (Figure 2) requires parts to be unloaded and loaded at the same position with the stations configured around a loop. The open system has the distinction of a variable number of parts in the system at any one time. The closed system, on the other hand, has a fixed number of parts in the system at all times.

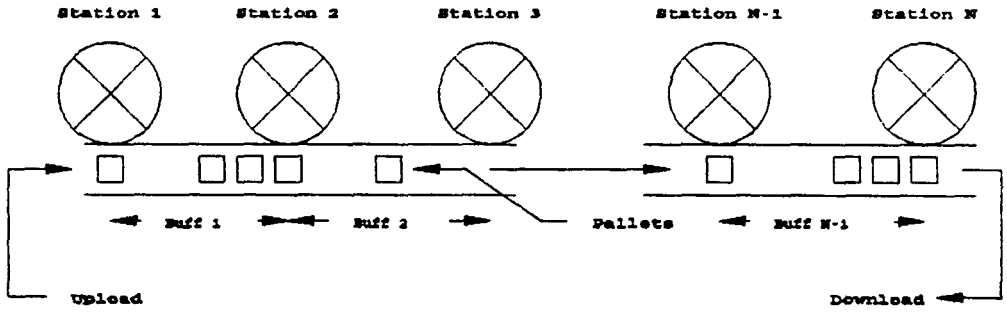

Figure 1. An open asynchronous automatic assembly system

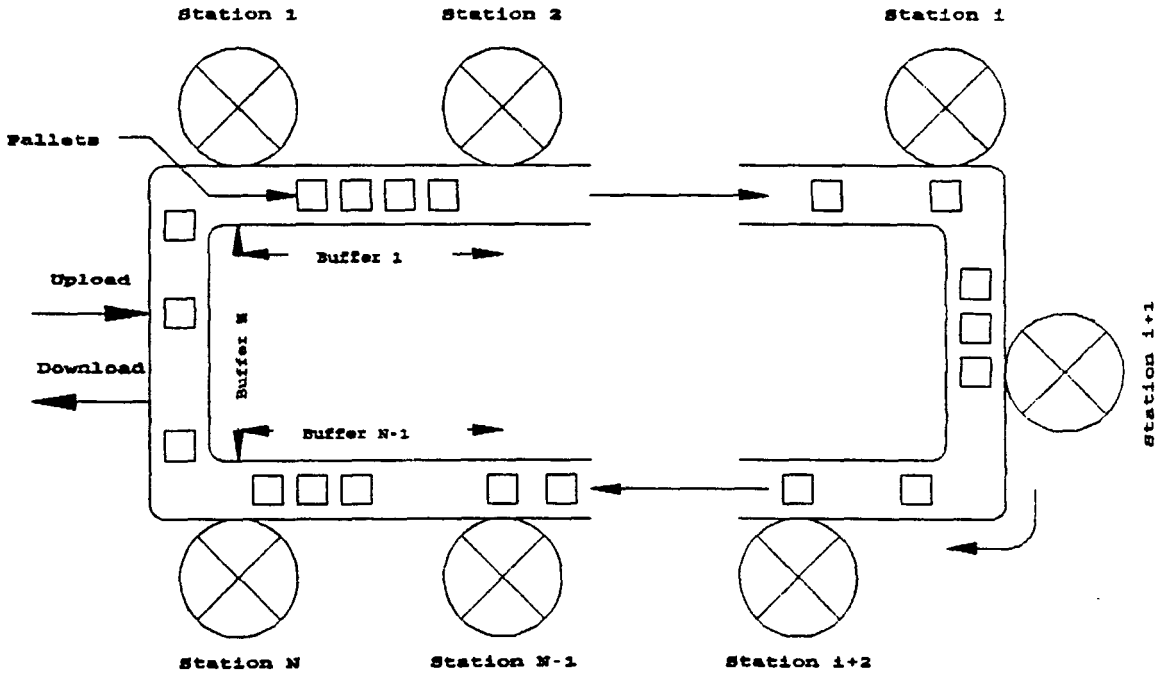

Figure 2. A closed asynchronous automatic assembly system

One of the primary decision variables in an asynchronous AAS is the optimal allocation of buffer space between stations. Allocating buffer space has the advantage of allowing stations to work more autonomously. The disadvantage is that buffer space introduces transport delays between stations. The application of large buffer spaces also will have large costs associated with the design. This can be attibuted to the incremental costs of adding additional conveyor, the cost of floorspace, and the cost of carrying extra work-in-process inventory. The cost of floorspace is not trivial, with a couple estimates being $1600 per square foot for General Motors [Liu, 1987] to $5000 per square foot for clean room facilities in computer chip manufacturing (data obtained for clean room estimates from discussions with an industrial engineer from a chip manufacturing facility).

Another major decision variable in the closed asynchronous AAS case, is the optimal number of fixture pallets to allocate to the system. From a system efficiency standpoint, allocating too many pallets introduces "blocking" of an upstream station when a station experiences a breakdown. Too few pallets will account for the "starvation" of downstream stations due to longer transportation delays. Also, from an economic standpoint, each fixture pallet can cost from $1000 to $5000 [Liu and Sanders, 1988].

B. Optimization Approaches

There have been several methodologies investigated to try and gain some insight on the problem of optimizing the allocation of pallets and buffer space. Their differences involve the manner in which the system is modeled, the number and variety of simplifying assumptions used in

modeling the system, and the algorithm used to optimize the AAS design. The primary algorithms investigated will include: stochastic quasigradient (SQG) methods, simulated annealing, and genetic algorithms.

The objective function (throughput, or total profit) is a stochastic function of the input variables. The optimization process of a stochastic function therefore leads to a Monte Carlo method of global optimization. The merits of each aforementioned algorithm as applied to the buffer allocation problem will be investigated. The genetic algorithm will be the primary method investigated. No literature has been found on a genetic algorithm's effectiveness on this problem.

John Holland [Holland, 1975] founded the field of genetic algorithms (GAs). His book, *Adaptation in Natural and Artificial Systems*, discussed the ideas of representing complicated structures by a simple representation of bit strings, and the power of simple transformations to improve these bit strings. These transformations, based on the mechanics of natural selection and "survival of the fittest", are reproduction, crossover, and mutation. The idea being that a model of the natural evolution process might be applicable to standard optimization problems.

Nature has been very good in optimizing the ecosystems since those individuals with "good" traits tend to populate where those with "bad" traits tend to die out. The genetic algorithm process, as applied to AAS design, might be interpreted as the operation on those designs with favorable objective results, leading to the generation of other favorable designs. Also, the operation on those designs with unfavorable objective

results will lead to those designs dying out.

C. Research Objectives

The primary effort of this research will be to investigate the applicability of a genetic algorithm to a closed asynchronous automatic assembly system. A hypothetical closed AAS model containing 10 stations configured in a single loop with unreliable stations is used as a testbed. Objective function estimates are furnished through a computer simulation model. From the analysis, we will describe how a genetic algorithm performed on the testbed and its applicability to similar systems.

II. REVIEW OF LITERATURE

This literature review assembles a collection of two areas of the automatic assembly process analysis. The first being the different approaches of modeling an AAS, including the different underlying assumptions of each modeling method. The second area includes a review of several stochastic optimization algorithms. This second area will summarize the algorithm method and investigate the advantages and disadvantages of each algorithm as applied to the AAS optimization problem.

A. AAS Modeling Methods

Queuing networks can be used to solve several practical problems. Two major classifications can be seen with respect to AASs. Open networks [Jackson, 1963] allow jobs to enter and leave the network. Closed networks [Gordon and Newell, 1967] have a constant set of jobs staying in the network. In a closed network, the practical interpretation is departing jobs are replaced with a statistically identical job so one can analyze the system as if no jobs enter or depart.

The AAS differs from the basic queuing system in that it includes a transport mechanism that moves parts from station to station. The transportation mechanism, reliability of the stations, topology of the system (open, closed, or multiple loop), and other complicating factors make the analytic modeling of a real assembly system difficult. Two major analytic methods include Markov chain analysis and queuing network models. The difficulties of each approach, as applied to an AAS, are summarized below.

Markov chain models require a large number of states to model a system of more than two or three stations. Each station in an AAS can be in one of two states: (1) a "down" state represents a station that cannot process parts due to a jam or machine breakdown, or the station is "forced down" when its buffer is empty (station is said to be "starved") or when the buffer space of the next downstream station is full (station is said to be "blocked"), (2) an "up" state represents a station that is processing parts normally. Also, each buffer having a storage capacity of N assemblies has N+1 states (corresponding to 0 to N assemblies in the buffer). For example, a 10 stage system with 10 buffers each having a capacity of 5 assemblies will have $2^{10} \times 6^{10}$ ($\approx$62 billion) states.

The queuing network models have a difficulty in incorporating the transportation delays or the blocking and starvation aspects of an AAS. Recently, however, some effort in trying to model the transportation delay and its effect on performance evaluation of transfer lines can be seen [Commault and Semery, 1990]. With regards to the blocking/starvation issue, most queuing models involve the assumption of infinite buffer space; therefore, the blocking effect is dismissed. In actuality, buffers are usually small and the blocking effect is not negligible.

To better understand the AAS model, it is necessary to define the appropriate variables which will be under investigation. These variables will also divide the literature into logical areas where the analysis will concentrate on the impact of several input variables on AAS performance. Also, the models will vary according to the simplifying assumptions of the model. The following represents the typical assumptions and model

parameters:

Station service time
- Deterministic
- Stochastic

Transfer mechanism
- Synchronous
- Asynchronous

Inter-Stage buffers
- None
- Finite
- Infinite

Transport delay time
- None
- Non-zero

Topology of the system
- Serial (open)
- Single loop (closed)
- Multiple loop (closed)

Inter-arrival time at first station
(this is for open systems, since the arrival rate for a closed system is equal to the output of the last station)
- Zero
- Deterministic
- Stochastic

Station jams or breakdowns
- None
- Stochastic

Stations clear or repair times
- Deterministic
- Stochastic

Scrapping of the assemblies after a jam/breakdown
- None
- Random fraction
- All

The remainder of this section will review the literature corresponding to AAS modeling approaches. The AAS modeling literature is divided by

those authors who investigate open systems versus those who investigate closed systems. The section will conclude with some discussion of literature corresponding to simulation models of assembly line systems. For additional material on analytical models, an excellent review of seven analytical models is presented by Buzacott and Hanifin [Buzacott and Hanifin, 1978].

1. Analytical models

Analytical models, or stochastic process models, is one approach to modeling assembly systems. The following review summarizes the literature available for modeling open systems and then follows with a review of closed systems models.

a. Literature review of open systems Open systems are those that have parts entering at one end of the system and departing at the other end. This type of system has received the most attention as it is relatively easier to model than the closed system configuration. This discussion of open systems is further divided into three areas, including:

- System models with reliable stations with random processing times
- System models with unreliable stations with deterministic processing times
- System models with both unreliable stations and random processing times

1) Models containing random processing times These models concentrate on determining what effect random processing times have on the system performance. Most models assume the processing times are

random variables with an exponential, Erlang, or normal distribution. The common objective is to determine what influence internal buffer storage, number of stations, and station sequence has on system performance. The investigations are typically restricted to the two- or three-station case.

Hillier and Boling [Hillier and Boling, 1966] discussed what effect the number of work stations, amount of buffer storage, and unbalancing of the station cycle times has on the production rate. They considered systems containing two-, three-, and four-station production systems. Their approach was not to develop a model yielding exact numerical results that were from a real system. Rather, they were interested in tractability where the objective was to gain insight in relative magnitudes of design changes on production performance. They used basic queuing theory equations to determine the effects of each factor. Their primary contributions were to support evidence that, in some cases, unbalancing a production line can in fact increase its efficiency. They found that production was maximized by assigning a somewhat lower mean operation time to the intermediate stations. They characterized this as the "bowl phenomenon."

Hatcher [Hatcher, 1969] investigated the impact of adding internal buffer storage capacity for two- and three-stage production lines. He also looked at the decrease in production rate caused by adding stages to the line. The service times for each station was assumed to be a statistically independent exponential random variable. Hatcher concluded that near optimum production rates could be attained with relatively small buffer allocations. He determined that after considering a wide variety of service

rates, that ten or less items per buffer would be sufficient. Hatcher also concluded that adding additional stations would reduce the output rate of the production line; however, the incremental effect would diminish as the number of stages increased.

Rao [Rao, 1975] described analytical solutions for determining the production rate of a two-stage serial production system with variable operation time at the stages. He reported that the methodology could be applicable to any type of service time distribution, where he worked out specific examples using the Erlang and normal distributions. The analysis lead to the conclusion that at high values of coefficient of variation, type of service time distribution had a considerable effect upon the system throughput.

2) Models containing unreliable stations This research effort concentrates on the effects of station failures and subsequent repairs on the efficiency of the system. The station processing times are considered to be deterministic and constant. The system randomness arises through random times between station jams/breakdowns and the random time required to clear/repair the station. Most studies assume that each station failure is independent of other station failures. The breakdowns and repairs are generally modeled assuming geometrically distributed random times. This can be attributed that most models discretize time. Thus, time moves in discrete increments where the system moves from one "state" to another "state." The geometric distribution is the discrete analog of the exponential; thus, an exponential distribution is often used in simulation models.

Sheskin [Sheskin, 1975] analyzed the allocation of storage capacity to buffers between consecutive machines in a serial, synchronized, automatic transfer production line. Exact numerical results were presented for lines up to 4 machines with a total buffer capacity of twelve units. A decomposition methodology was outlined which would give approximate results for lines containing more than four stations.

Yeralan and Muth [Yeralan and Muth, 1987] presented a general model for production lines containing two unreliable stations, a finite capacity inter-station buffer, constant cycle time, and synchronous transfer. They used Markov chain analysis and presented a general recursive procedure to solve for the steady-state probabilities. They also investigated cases when the steady-state probability vector leads to the ability to express the production rate as a closed-form function of the buffer capacity.

Jafari and Shanthikumar [Jafari and Shanthikumar, 1989] present a heuristic solution of the optimal distribution of intermediate storage buffers, given a total storage capacity, in a synchronized transfer line with an arbitrary number of stages. They formulate the system as a dynamic programming problem to compute the production rate of the transfer line. They showed that the dynamic programming results gave reasonable estimates of the production rate when compared to an exact production rate for the three-station case.

Recently, Commault and Semery [Commault and Semery, 1990] presented an approach to incorporate the transportation delay in buffers into an analytical performance evaluation of transfer lines. They showed

that the delays in buffers could be approximately represented as an "equivalent line" with the same machine characteristics but reduced buffer capacities. The reduction of a two-station intermediate buffer capacity by a number of ΔC spaces, where ΔC is defined as the transit time in the buffer divided by the maximum of the adjacent machines' processing time. This method was reported to give a pessimistic production rate evaluation. The error was determined to be less than one percent for the two-station case, and simulation results supported the application to systems containing more than two-stations.

3) Models containing unreliable stations and random process times These models allow the random variables to include the processing times at the stations, the jam/breakdown rates, and the time to clear/repair the station. These models add significant complexity to the analytics; therefore, there is a limited number of studies.

Buzacott [Buzacott, 1972] presents an exact Markov model of a two-station system with unreliable stations and random station processing times. He showed a simple method of combining the results from a previous work. This previous work contained a model using random processing times and breakdowns in a fixed-cycle time system. He showed this gave very good approximations to the results of the exact model. He also outlines how this method could be extended to systems with more than two stations. He pointed out that buffer storage capacity of larger than 4 or 5 units result in a low marginal improvement when dealing with only random processing times. He continued to point out that in the unreliable station and constant process time case, a buffer storage capacity should be at least

equal to the mean repair time in cycles. For example, if the mean repair time is equal to 10 cycles, a buffer storage capacity of 20 or 30 would be appropriate. He suggested the system designer consider the two random effects separately and decide whether to use the buffer storage capacity to reduce the effect of random processing times or reduce the effect of breakdowns. In the former case, a relatively small buffer capacity is required while the latter case a much larger capacity is required.

Choong and Gershwin [Choong and Gershwin, 1987] presented a decomposition method to evaluate the performance measures of a capacitated transfer line with unreliable machines and random process times. The decomposition was based on approximating the *k-1*-buffer system by *k-1* single-buffer systems. The numerical examples indicated that the approach was accurate as long as the probability that a machine is starved and blocked at the same time is small. They stated the accuracy of the algorithm didn't seem to be sensitive to the number of stations in the line. However, it did appear to be unstable and not always converge. Dallery, David, and Xie [Dallery et al., 1988] extended this analysis and proposed an algorithm with a lower computational complexity. Further, in all cases they tested, the algorithm always converged.

b. Literature review of closed systems The bulk of the literature considers open system configurations. However, closed systems have received some attention in recent years. The ability to analytically model a closed system usually involves making much more restrictive assumptions. Most studies assume unlimited buffer space, no transportation delays, and a

small number of stations. To present, computer simulation is the most widely used modeling approach to the closed system.

Suri and Diehl [Suri and Diehl, 1986] presented a model which enables efficient analysis of certain types of closed queuing networks with blocking due to limited buffer spaces. The networks they analyzed were those in which the limited buffers occur in tandem subnetworks. They assumed the buffer capacity of the loop-back buffer was infinite (i.e., in a M-station system, the buffer between the M_m and M_1 stations was infinite). The model was solved iteratively for each subnetwork and then for the entire network. Their study assumes exponential service times for all stations.

Kamath and Sanders [Kamath and Sanders, 1987] considered two analytical techniques, namely the Renewal Approximations (RA) method and the Product-Form Analysis (PFA) method. For the cases they examined, RA method was substantially superior to that of the PFA method. The performance models did not include the blocking due to finite buffers or the transportation delay times.

Bastani [Bastani, 1990] considered a closed-loop conveyor system with a single loading station and multiple unloading stations. He investigates a two-station system that allows breakdowns and random exponentially distributed interarrival times at the loading station. The service times of the unloading stations, the time between failures, and the repair times of unloading stations are all i.i.d. random variables having and exponential distribution. A matrix-geometric solution is obtained which provides an approximation of the steady-state probabilities. This configuration doesn't

quite fit our definition of a closed system, but its approach was felt to merit inclusion.

2. Simulation models

Simulation is a good tool whenever the system is very difficult to model, or when very accurate results are required. Simulation models allow many random variable probability distributions, a variety of system configurations, and a variety of statistical collection procedures to generate estimates of performance. Also, there is a host of discrete-event simulation languages (SIMAN, GPSS, SLAM, SIMSCRIPT, to name a few) that will decrease the amount of programming effort by the system designer. These simulation languages generally have statistics collection routines, random number generators, event clock mechanisms, and report generators. Also, simulation models can be implemented relatively easily in any procedural language (such as FORTRAN, BASIC, C, PASCAL, etc.). The main restriction to a simulation model approach is time and cost requirement of coding and running the model. For an excellent review of simulation languages and a discussion on procedural programming simulation techniques, the reader is directed to the text, *A Guide to Simulation* [Bratley et al., 1987].

Simulation models in the literature tend to describe systems with very specific configurations and applications. However, some authors have used simulations to develop some insight on the impact of design constraints on performance in a general system. Many studies use simulation as a check for the validity of the approximations set forth by an analytical model. The remainder of this section, however, will discuss the

findings of those studies which gave way to generalizations of AAS system design.

Freeman [Freeman, 1964] considered the operational and economic aspects of the number and sequence of production stages and the amount and allocation of storage capacity among the stages. He found that correct allocation of total buffer capacity is an important consideration. He presented several generalizations based on simulation results from a three-station production line. The generalizations are presented as follows:

- Avoid extreme allocations, that is no buffer capacity between some pairs of stages and all between other pairs.
- The worse a bad stage is, relative to the good stages, the more the buffer capacity that should be allocated to it.
- More buffer capacity should be allocated between two bad stages than between a bad and a good stage.
- The optimum relative allocation is substantially invariant to changes in the total buffer capacity
- The end of a line is more critical than the front. If a bad stage occurs toward the end of a line it should be allocated an even larger share of the total buffer capacity.

Okamura and Yamashina [Okamura and Yamashina, 1983] investigated the role of buffer stock in a multi-stage transfer line system. They used simulation results to show that an n-stage line should be designed such that the lowest stage production rate occurs in the nth stage, the second lowest in the first stage, the third lowest in the $(n-1)$st stage, the fourth

lowest in the second stage and so on, to maximize line production rate. They also reported the following generalizations:

- Uniform buffer storage capacity allocation does not guarantee the optimum allocation even for balanced identical lines.
- For a multi-stage line, the number of stages and buffer storage capacity between the stages are critical design factors strongly influencing the production rate of the system.
- The total buffer capacity should be allocated to the buffers in such a way that the difference between the two production rates of the stages on either side of a storage point is minimized.

3. Summary of literature

Based on the review, analytical models can be used effectively to model systems containing two or three stations. The difficulty of analytically modeling the occurrence of transportation delays, blocking effects, and closed system configurations make simulation modeling the method of choice if these phenomena are crucial to the system definition. Also, simulation is the chosen modeling method when the systems are very complex and contain significant interactions between stations.

B. Stochastic Optimization Algorithms

Stochastic optimization of an objective function estimated by computer simulation is also called Monte Carlo optimization. The optimization methods use the simulation to obtain an estimate of the objective function value, then applies some search algorithm to find the optimal solution. Classical

Monte Carlo methods include: Robbins-Monro methods, Kiefer-Wolfowitz methods, and response surface methods. These methods are mentioned as possible approaches, but were not investigated in order to concentrate on the modern optimization methods of stochastic quasigradient (SQG) methods, simulated annealing, and genetic algorithms (GAs).

1. Stochastic quasigradient methods

Stochastic quasigradient methods are stochastic algorithmic procedures for solving general constrained optimization problems with nondifferentiable, nonconvex functions. SQG methods allow us to solve optimization problems where the objective function and constraints are very complex and it is impossible to generate exact values of these functions (or the derivatives). In the case of AAS design, the objective functions are discrete (buffer space) or continuous (station service times) stochastic functions. The approach is to use statistical estimates for the objective and derivatives, then apply a standard constrained procedure for a step direction to drive the value of the input variables to the optimum solution. Some of the available literature on applications of SQG methods are summarized below.

Liu [Liu, 1987] presented a list of advantages and difficulties of using a SQG method. The advantages listed were as follows:

- Flexibility in the choice of gradient estimation methods during iterations.
- Selection between automatic or interactive modes. In automatic mode, the algorithm can modify the step size and stopping criteria. In manual mode, the user can change gradient

estimation methods, step size, and number of observations for function value estimation.

The difficulties in using a SQG method are:

- Sensitive to the choice of starting point.
- Problem with choice of a good starting step size and the choice of the method to modify it during iterations.
- Selection of various methods for quasigradient estimates.
- Determination of good stopping criteria
- Convergence to the optimal region if the starting point is far away from the optimal region.

Ermoliev [Ermoliev, 1983] gives a survey of the development of SQG methods. He gives a general overview of the method, then proceeds to illustrate the use of SQG methods on many different problem types. He formulated problems into four groups: general stochastic programming problems, recourse problems, stochastic minimax problems and nonlinear programming problems. He concludes with a computer implementation of an example stochastic facility location problem.

Liu and Sanders [Liu and Sanders, 1988] presented the application of the SQG method of Ermoliev and Gaivoronski [Ermoliev, 1983; Ermoliev and Gaivoronski, 1984] to the performance optimization of asynchronous flexible assembly systems (AFAS). They used a simulation to obtain objective function estimates of a closed-loop system with stations subject to random jams/breakdowns with geometrically distributed repair times. The station blocking effect due to finite buffers and the starvation effect due to

transportation delays were included in the simulation model. They used a hybrid algorithm which used a queuing network model to set the total number of pallets in the system and then used an SQG algorithm to allocate the buffer spacing to obtain optimal system throughput. Different forms of the SQG algorithm were examined to determine the specification of buffer sizes in a ten-station AFAS.

2. Simulated annealing

Simulated annealing is a computational technique derived from statistical mechanics for finding near-global minimum-cost solutions to large optimization problems. Here the objective function is assumed deterministic. The approach is analogous to first melting a substance, then by careful annealing, reduce the temperature slowly to obtain the desired crystalline structures. Higher energy states (those states with higher cost function results) can be reached, but the likelihood of acceptance decreases as the temperature is decreased. The general method is to randomly generate a state, say j, from the current state, i. The new state, j, is accepted if the cost is less than that at i. Otherwise, the new state, j, is accepted if a random number, r, generated uniformly over the interval [0,1] is less than a real number, y, defined:

$$y=\exp\left\{-\frac{c(j)-c(i)}{T_m}\right\}$$

where: $c(j)$ = cost of new configuration

$c(i)$ = cost of present configuration

T_m = temperature at time m (m = 0, 1, 2, ...)

The process is started using a large value for T_m and then reducing the value as the number of iterations increase. Therefore, a higher cost move is accepted with higher probability in early stages than in later stages when T_m is reduced. This allows the algorithm to escape local minima convergence. Again, the assumption is that the cost function is a deterministic function. To be applied to the AAS buffer allocation problem, certain modifications would have to be made in order to compensate for the stochastic nature of the problem.

Simulated annealing has seen a number of applications in large combinatorial optimization problems. Specifically, the algorithm has been used extensively in the area of VLSI design [Kirkpatrick et al., 1983; Romeo and Sangiovanni-Vincentelli, 1985; Sechen, 1988; Wong et al., 1988]. It also has been applied to a stochastic portfolio problem [Gemmill, 1988]. The portfolio problem deals with the problems involved with optimizing the inventory levels of variable sized stock sheets given a random bill of material. Some advantages can be seen to using the simulated annealing algorithm:

- Under certain assumptions of the rules used by the algorithm and on the time spent at each temperature, the algorithm generates a global optimum solution with probability one.
- Allows "hill climbing moves" which allow the algorithm to escape local optimization.

Some difficulties can be also seen:

- Determination of a good "cooling schedule" for a slow reduction

in the temperature value.

- Determination of a good stopping criteria.
- Lack of convergence proof for a stochastic cost function.
- Allocation of computer resources. The algorithm typically requires a very large number of iterations.

Several studies have been conducted to try and resolve some of these difficulties.

Metropolis et al. [Metropolis et al., 1953] proposed an algorithm for the efficient simulation of the evolution of a solid to thermal equilibrium. It wasn't until some thirty years later that Kirkpatrick, Gelatt, and Vecchi [Kirkpatrick et al., 1983] realized the similarity of this cooling process to the minimization of the cost function of a combinatorial optimization problem. They demonstrated the use of simulated annealing on a wire routing and component placement problem in VLSI design. They also demonstrated the application of simulated annealing to a 400 city traveling salesman problem.

Mitra, Romeo, and Sangivanni-Vincetelli [Mitra et al., 1986] presented a theoretical analysis of simulated annealing based on its precise model, a time-inhomogeneous Markov chain. An annealing schedule was given for which the Markov chain was strongly ergodic and the algorithm converged to a global optimum. The finite-time behavior of the algorithm was also analyzed and a bound obtained on the departure of the probability distribution of the state at finite time from the optimum. This bound gave an estimate of the rate of convergence and gave some insight into the conditions on the annealing schedule which gave optimum performance.

Hajek [Hajek, 1988] gave a simple necessary and sufficient condition on the cooling schedule for the algorithm state to converge in probability to the set of globally minimum cost states. He showed that in the special case that the cooling schedule had a parametric form $T(t) = c/\log(1+t)$ the condition for convergence was that c be greater than or equal to the depth of the deepest local optimum which was not the global minimum state.

3. Genetic algorithms

As was mentioned in the introduction, genetic algorithms demonstrate a method of representing complicated structures by a simple representation of bit strings, and the power of simple transformations to improve these bit strings. These transformations, based on the mechanics of natural selection and "survival of the fittest", are reproduction, crossover, and mutation. The algorithm is used to maximize a nonnegative deterministic objective funciton. The remainder of this section will review the literature available and discuss a general procedure for implementing a simple genetic algorithm (SGA).

a. General description of the algorithm The approach is to first discretize and encode the decision variables into a finite binary (or some other appropriate alphabet) string. The binary positions have the parallel of being the "genes" and the concatenation of the "genes" form an "individual". Next, a series of strings are randomly generated and the objective function results are computed. This collection of "individuals" has the parallel of being the "population." Individuals are selected in the reproduction step of the algorithm according to their "fitness", that is, those individuals having greater objective function results will have a

higher probability of being selected for the next generation of the population than those individuals with lower fitness values. Two individuals are selected during reproduction and then are "cross-bred." This is the crossover step of the algorithm. This amounts to randomly choosing a point along the finite string, then swapping all positions ahead of this point between the two individuals. For example, if two individuals, A and B (with string positions defined using a binary alphabet) were crossed at the tenth position; individuals A' and B' would arise after crossover (see Figure 3). The final step of the genetic algorithm is mutation. Mutation is the occasional random alteration of a string position from a 1 to a 0 and vice versa. The function of mutation is a secondary role; where reproduction and crossover are search mechanisms, mutation guards against losing potential useful genetic information at a bit position.

```
A = 1001 0100 01|11 0111
B = 1100 0101 11|00 1001

A'= 1100 0101 11 11 0111
B'= 1001 0100 01 00 1001
```

Figure 3. Example of simple crossover

Here we have formulated a representation of a simple genetic algorithm (SGA). The intent of presenting a general description is to acquaint the reader with some of the terminology and to get a general feel for the process of the genetic algorithm transformations. Perhaps Goldberg [Goldberg, 1986] explained the ability of a genetic algorithm to process information best:

> Consider a population of *n* strings over some appropriate alphabet coded so that each is a complete IDEA or prescription for performing a particular task. Substrings within each string (IDEA) contain various NOTIONS of what's important or relevant to the task. Viewed in this way, the population contains not just a sample of *n* IDEAS, rather it contains a multitude of NOTIONS. Genetic algorithms carefully exploit this wealth of information about important NOTIONS by 1) reproducing quality NOTIONS according to their performance and 2) crossing those NOTIONS with many other high-performance NOTIONS from other strings.

b. Applications of a genetic algorithm Genetic algorithms have seen a wide and diverse area of applications. Genetic algorithms have been used to solve the general traveling salesman problem [Goldberg and Lingle 1985; Grefenstette et al., 1985; Whitley et al., 1989], flow shop scheduling [Cleveland and Smith, 1989], job shop scheduling [Davis, 1985], machine learning [Goldberg, 1985a, 1989], and even to create production rules that pick winners of horse races [Maza, 1989]. This wide diversity of applications is an indication of the robustness of the GA procedure to perform well in a diverse problem domain.

c. Review of genetic algorithm literature The volume of literature on genetic algorithms has increased dramatically since John Holland first introduced the procedure in 1975. Specifically, with the organization of three international conferences (1985, 1987, and 1989) and another being

planned in the future (July 13-16, 1991, at the University of California at San Diego), the amount of literature available has grown particularly in the past five years. Also, with the publishing of the text by David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* [Goldberg, 1989], the field of genetic algorithms has seen new interest from a variety of fields.

In this section we will attempt to summarize the available literature dealing with aspects of the implementation questions of a GA. The theory of schemata and convergence properties of a GA will be discussed in the following chapter. For a thorough treatment of the theory, the reader is directed to the aforementioned text by Holland.

John Holland's text [Holland, 1975] discusses the theoretical foundations of a genetic algorithm. He explores the idea of abstracting the adaptive process of natural systems and designing artificial systems that retain the mechanisms of the natural systems. He introduces the idea of "implicit parallelism" where he reports that working with a population of N individuals, you are effectively processing N^3 information of the search space. This has been a "well known but poorly understood" [Goldberg, 1989, p. 40] claim, but recently has received some investigative studies to help understand this phenomenon [Grefenstette and Baker, 1989; Goldberg, 1985b].

In Goldberg's text [Goldberg, 1989], he explains the basic mechanisms of the genetic algorithm in a very general and clear manner. He discusses the major issues of a genetic algorithm with emphasis on computer

implementation, robustness, theoretical derivation and mathematical foundations, and applications of the algorithm. He has furnished many Pascal examples and included computer assignments at the end of each chapter. He also includes two chapters on the implementation and summary of the literature regarding the use of GAs in machine learning. The text also has the most complete bibliographic listing of GA literature to date.

Davis [Davis, 1987] edited a text that collected several papers, from a variety of authors, dealing with simulated annealing and genetic algorithms. In the first chapter he presents an overview of genetic algorithms and simulated annealing. The text continues with papers discussing the issues of premature convergence of a GA (contributed by Lashon Booker), the minimal, deceptive problem for a GA (contributed by David E. Goldberg), as well as many other issues concerning simulated annealing and GAs.

Goldberg and Richardson [Goldberg and Richardson, 1987] discussed a method of "sharing functions" to enhance a genetic algorithm's ability to optimize multi-modal objective functions. This method developed the formation of stable subpopulations of different strings to permit the parallel investigation of many peaks. The theory and implementation was investigated for two, one-dimensional test functions. For a test function containing five peaks of equal height, a GA without sharing was found to lose strings at all but one peak, but was found that with sharing a GA maintains approximately equally sized subpopulations at all five peaks. For a test function with five peaks of unequal height, a GA with sharing was found to allocate a proportionally decreasing number of strings to each decreasing peak.

Syswerda [Syswerda, 1989] investigated the applicability using a uniform operator to replace the normal one-point or two-point crossover operator. He developed the theoretical implications of the uniform operator with respect to the survival rate of the schemata expressed by the parents. He then compared the uniform operator's performance against a variety of function optimization problems.

Fogarty [Fogarty, 1989] discussed the effect of varying the mutation probability over time and its effect on GA performance. He used ten different simulations of multiple burner furnaces created randomly, where a GA was used to set the air inlet valve in order to minimize combustion stackloss in the common flue. Two initial populations of settings were used, one consisting of the most conservative starting point with all inlets fully open and the other randomly generated. Mutation rates were then varied according to four different time schedules. It was observed that varying the mutation rate significantly improved performance of the conservative initial population case, but not when the initial population was randomly generated.

Richardson et al. [Richardson et al., 1989] discussed some guidelines for genetic algorithms with penalty functions. The concept of the penalty function is to "penalize" those observations that are infeasible in a constrained optimization problem. Therefore, the purpose of the penalty function is to decrease (increase) the objective function result by a specified amount in order to achieve a global maximum (minimum) that is feasible. Current thought is to penalize infeasible observations very harshly. Richardson investigated this practice, and provided some

guidelines to the use of these functions.

4. Summary of literature

All approaches have their merits and difficulties, but all three could be considered as viable options to attempt the buffer allocation problem of AASs. Simulated annealing and genetic algorithms, however, are inherently designed for problems dealing with deterministic objective functions. The AAS optimization problem is a stochastic function of the decision variables; therefore, attention needs to be given to the fact that the objective function estimate is an expectation. The simulated annealing algorithm has been shown to converge in probability to the global optimum with probability one, but this again is for a deterministic function. One cannot assume the property in the stochastic case.

The SQG method is shown to be a viable option, but the algorithm tends to be a "greedy" algorithm in that it finds local optima quickly at the expense of locating global optima. The simulated annealing algorithm also presents itself as a possible optimization technique, but the long run-times required for convergence is a drawback. The genetic algorithm presents an interesting approach and has not been attempted on a AAS optimization problem; therefore, a genetic algorithm will be implemented and tested.

III. METHODOLOGY

This chapter will discuss the methods involved with constructing a simulation model of a representative AAS system, and the implementation details of a genetic algorithm. The discussion of the simulation model will concentrate on implementation issues concerning using an object-oriented general-purpose language (C++ was used) for simulation modeling. The model was validated by using a model described in Liu and Sanders [Liu and Sanders, 1988] and comparing their results with those obtained by the simulation model of this study.

The mathematical foundations of a genetic algorithm will also be addressed. The formulation of schema and the effective processing of these schema will be the concentration in the theoretical discussion. Also, the fundamental theorem of genetic algorithms will be derived and examined.

A. Simulation Model

In order to evaluate the impact of different buffer allocation configurations in an AAS, a method to obtain estimates of the objective function values is required. As discussed in Section II.A.1.b, analytical models for a closed loop system make very restrictive assumptions for the inputs and the number of stations allowed. Simulation allows one to design a model that can incorporate a higher degree of stochastic complexity, where the interactions of random variables need not be described explicitly. However, the simulation model can require a large investment of time to

design, debug, and execute the model.

Numerous dedicated simulation languages (GPSS, Simscript, Simula, etc.) are available to reduce the amount of programming effort required in designing a simulation model. The languages aim to make writing simulations more concise and making the simulation mechanics more transparent. A dedicated simulation language offers convenience, but often at the sacrifice of control.

Central to any simulation model is several essential components including: a clock mechanism, a source of random numbers, a listing of upcoming events to be processed (event schedule), data structures for statistics gathering, and data structures representing transactions, resources, and queues. A dedicated simulation language offers routines to automate some of these processes. However, a general-purpose programming language offers the flexibility and the opportunity to design all components of the simulation model. The programming language C++ was chosen for several reasons:

- A compiler was available for the PC style computer.
- Data structures can be created dynamically. That is, the memory required for a data structure is allocated at run-time versus compile-time. This enables the efficient use of memory by using only that which is required.
- Object oriented design of data structures.

The last reason is the primary reason for choosing C++ over the C programming language. An object-oriented language allows the grouping of data, and the procedural routines (functions) that work on this data, into a

single structure defined as an "object." For this model, this relates to defining objects such as queues, stations, and pallets, and then designing how these "objects" interact with one another.

A general-purpose programming language also permits the design of the simulation system resources. Thus, the system resources can be optimized for execution speed and reliability for the implementation of the specific model. The critical system resources for this model were the random number generator and the method of inserting and removing events from the event schedule.

The random number generator implemented was the generator proposed by Wichmann and Hill [Wichmann and Hill, 1982]. This used three simple multiplicative congruential generators to combine and make one uniform random number stream. The advantage of this generator is the long cycle length (reported to exceed 2.78×10^{13}).

A splay tree was used to store the upcoming simulated events. A splay tree is essentially a special form of a binary tree. A simple binary tree will become unbalanced by the repeated removal of the leftmost event on the tree. The splay tree eliminates this problem by balancing the tree with every insertion or removal of an event. The splay tree was found to be consistently stable and perform better than a variety of other implementations [Jones, 1986]. For a thorough comparison of implementations, the reader is directed to the aforementioned study by Douglas Jones.

Once a simulation model is coded, the program is checked in two stages: verification and validation. Verification involves checking the

simulation program to determine if it operates in the manner in which it was intended to operate. That is, this is the "debugging" procedure of any programming exercise. To validate this simulation model, as was mentioned in the opening of this section, Liu and Sanders' model was used as a check. Their study reported expected throughput for several system configurations. Three of these configurations were chosen, and ten simulation runs consisting of manufacturing 20,000 assemblies each were used to obtain the confidence intervals. In each case, the first 10% of each run is removed in an attempt to remove the initial transient. All three configurations had no significant difference between the simulation model used in this study with that used in Liu and Sanders' study. The results can be seen in Table 1.

B. Genetic Algorithm

Genetic algorithms can be shown to possess a random, yet structured, method for functional optimization. The discussion in Section II.B.3 was intentionally qualitative to simply introduce the mechanics of a genetic algorithm. Here, we will explore more rigorously the implications of the mechanisms of a genetic algorithm. The notion of schemata and similarity templates will be introduced and how the transformations of reproduction, crossover, and mutation effect these.

1. Mathematical foundations

Without any loss of generality, consider a string A containing ℓ elements defined on the binary alphabet V = {0, 1}. Where A may be represented:

Table 1. Verification of C++ simulation program code against results reported by Liu and Sanders (1988).

Buffer Size										C++ Model		Liu & Sanders	
B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	TP	95% CI	TP	95% CI
5	5	17	4	4	4	4	5	5	5	0.1275	±0.0012	0.1286	±0.0016
4	4	10	10	12	12	4	4	4	4	0.1285	±0.0024	0.1301	±0.0024
2	3	4	4	4	2	2	2	3	3	0.1273	±0.0021	0.1270	±0.0027

where:
TP = Throughput of last station (ie., average number of parts produced per time unit)
Geometric Mean Clear Time = 36 time units for all station
Station Cycle Time = 5 time units for all stations

Configuration 1
Total number in system = 40 pallets
Jam rates = (0, 3, 3, 0, 0, 0, 3, 0, 0, 0) per 100 assemblies for stations
95% confidence interval of the difference between the models = -0.0011 ± 0.0016

Configuration 2
Total number in system = 40 pallets
Jam rates = (0, 3, 0, 3, 0, 3, 0, 0, 0, 0) per 100 assemblies for stations
95% confidence interval of the difference between the models = -0.0016 ± 0.0025

Configuration 3
Total number in system = 20 pallets
Jam rates = (0, 3, 0, 0, 2, 0, 0, 2, 0, 0) per 100 assemblies for stations
95% confidence interval of the difference between the models = 0.0003 ± 0.0027

$$A = a_1 a_2 a_3 a_4 \ldots a_\ell$$

Here each a_i represents a binary feature (sometimes referred to as a gene or allele), and A represents the concatenation of the binary features (sometimes referred to as an individual). If we now consider a population of individual strings, A_j, j = 1, 2, 3, ..., n, contained in population A(t) at generation t, the notion of schemata can now be addressed.

Consider a schema H defined on the three-letter alphabet V+ = {0, 1, *}, where the * symbol is a wild card symbol which matches a 0 or 1 at a particular position. Therefore, if a string A_j has ℓ binary positions, there are 3^ℓ schemata or similarity templates defined. For the entire population,

there are at most $n \cdot 2^{\ell}$ schemata since each individual is a representation of 2^{ℓ} schemata. In general, a string with alphabet having cardinality C, there are $(C + 1)^{\ell}$ schemata defined with at most $n \cdot C^{\ell}$ schemata in the population. To establish a means of differentiating the properties of these schemata, schema order and defining length are used.

Schema order, denoted $o(H)$, is defined to be the number of positions that are fixed in a certain schema. For example, using a string length of seven ($\ell = 7$), a schema 1**01** has an order of 3, whereas the schema **0**** has an order of 1.

Schema defining length, denoted $\delta(H)$, is defined as the distance (in allele positions) from the first to the last fixed string position. For example, the schema 1**01** has a defining length of 4. This can be seen by subtracting the first fixed position's index from the last fixed position's, or 5 - 1 = 4. In the other example schema, **0****, has a defining length of 0.

In order to understand how the schema are processed, the expected number of schemata in a population after reproduction can be determined. To restate the definition, reproduction involves randomly selecting individual strings with replacement, weighted according to the relative "fitness" of an individual string. A string A_j has probability $pselect_j = f_j / \sum f_j$ of being selected where f_j is defined as the fitness of string j. The process of selecting an individual for reproduction has frequently been referred to as spinning a biased roulette wheel where each slot's dimension is sized according to string fitness. Now consider a schema H contained in

the population A(t) having m examples of this schema, denoted $m=m(H,t)$. The average fitness for a particular schema at time t will be denoted $f(H)$ and can be calculated using the following expression:

$$f(H) = \frac{\sum_{A_j \in H} f(A_j)}{m(H,t)}$$

The expected number of schema in a nonoverlapping population of size n is then given by the equation:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}$$

$$\textit{where: } \bar{f} = \frac{\sum f_j}{n}$$

That is, those schema with average fitness greater than the population average fitness expect to have an increasing number of representative strings, while those schema with average fitness below the population average will expect to receive a decreasing number. Thus, reproduction allocates increasing numbers of high performance schemata in parallel.

Reproduction allows the algorithm to distinguish between high and low performance schemata, but does nothing in the way of exploring new regions of the search space. The exploration process is performed primarily by crossover and secondarily by mutation. A simple crossover operation, to review, proceeds in two steps. First, two individual strings are chosen at random from the newly reproduced strings. Second, a crossover is performed with probability p_c, with a crossover occurring at a

uniform randomly selected position k along the string length less one [1, ℓ-1]. Thus, a particular schema is disrupted if a crossover occurs within the interval of the first or last fixed position of the schema. As an example, the string A_1 and a representative schema H_1 is defined as follows:

A_1 = 0 1 1 1 0 0 1 1

H_1 = 0 * * 1 0 * * *

If a crossover occurs at a point between the first and fifth positions, schema H_1 is disrupted (unless A_1's mate is identical, with the probability of this occurring neglected giving a conservative estimate for the probability of schema disruption). Therefore, the probability that a schema survives depends on the defining length of the schema and the length of the string. The survival probability p_s will then have a lower bound described by the expression:

$$p_s \geq \left[1-p_c\frac{\delta(H)}{\ell-1}\right], \quad \ell \geq 2$$

Incorporating this expression into the schema expectation equation:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{\ell-1}\right]$$

The final operation of a genetic algorithm is mutation. Mutation occurs at each position along the string with probability p_m. Hence, the position survives with probability (1 - p_m) and a particular schema will survive with probability $(1 - p_m)^{o(H)}$. For small values of p_m this can be approximated by $1 - o(H)\cdot p_m$ and the schema expectation can now be written

(ignoring cross-product terms):

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{\ell-1}-p_m o(H)\right], \quad \ell \geq 2$$

This result is the fundamental theorem of genetic algorithms. The schema theory describes how a genetic algorithm allocates an increasing number of trials to low order, short defining length, above average schema. However, the schema theory alone does not guarantee convergence for an arbitrary problem. This merely describes how a genetic algorithm processes many schema in a parallel fashion. Holland [Holland, 1975] estimated that by processing n strings, an order of n^3 schemata are usefully processed. This type of leveraged search he called *implicit parallelism.*

Several authors have addressed the issue of lack of guaranteed convergence. Bethke [Bethke, 1981] examined some sufficient conditions for simple GA convergence using Walsh function analysis. Goldberg [Goldberg, 1990] described a selection procedure for genetic algorithms called *Boltzmann tournament selection.* Here he borrows the concept of thermal equilibrium and the Boltzmann distribution from simulated annealing and adapts them to a genetic algorithm. This allows the implementation to exhibit the same asymptotic convergence as the simulated annealing algorithm.

2. Implementation

The implementation of a simple genetic algorithm was performed using the C++ programming language. The choice of using C++ over any other general-purpose programming language was not as critical an issue as it

was in the case of the simulation model. Typically, any programming language which contains the ability to group data into structures would suffice.

The program was designed for generality at some expense of program execution speed. The genetic algorithm optimization program, named GENOPT, manages all genetic operations and accumulates population statistics. The calculation of the objective function result was purposely not incorporated with the GENOPT program. This allows GENOPT to be applied to any executable program that generates an objective result, not just those programs compiled and linked with the GENOPT main program. GENOPT merely needs the name of the function calculation program (FCP) and any command line arguments, a FCP input file name, and a FCP objective function output file name. This was deemed desirable since the GENOPT program might be applied to simulation models coded in languages such as GPSS or Simscript in the future.

C. Application of a GA to an AAS Buffer Allocation Problem

The application of a simple genetic algorithm to the AAS buffer allocation problem will be the principal objective of this study. In order to gauge the effectiveness of a GA relative to other approaches, the analysis of Liu and Sanders [Liu and Sanders, 1988] will be closely followed. This will allow a direct comparison to the SQG method used in their study.

As was shown in Section III.A with the verification of the simulation model (Table 1), a 95% confidence interval of the difference between Liu and Sanders' model and the C++ model included zero for the three

configurations tested. This suggests that the two models are not significantly different. Therefore, we will adopt their system specifications and use them for comparison.

The system used by Liu and Sanders was a ten-station, asynchronous closed-loop automatic assembly system. All stations had constant service times, random failures with geometrically distributed repair times, and transportation times of 1 time unit per buffer storage unit. The number of pallets were fixed at 40 for the first two configurations and at 20 for the last. It was assumed that there was no scrapping of assemblies due to failures.

IV. RESULTS

The results of this study are assembled into two sections. The first section will report the results of using a genetic algorithm to optimize four different deterministic objective functions where the optimal solution is known. This will serve as a benchmark on the performance of the simple genetic algorithm under a variety of objective functions. The analysis will concentrate on the effects of population size, crossover probability, and mutation probability on GA performance.

A. GA Performance on Deterministic Functions

To verify the performance of the genetic algorithm coded in the GENOPT program, four different objective functions were used. These functions are presented in Table 2. The first function is maximized while the remaining functions are minimized. Functions f_2, f_3, and f_4 were used by DeJong [DeJong, 1975] in his dissertation, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems." DeJong's study consisted of evaluating GA performance under a variety of conditions using a five function testbed. The three functions used here were the first three of the five. Since the objective of this study is to apply a GA to an AAS buffer allocation problem, the four functions were deemed sufficient to verify the working of the GENOPT program.

In order to evaluate the performance of the GA, a method of quantifying system performance is required. DeJong used two functions to quantify GA performance and these functions will be used here. He defined

Table 2. Deterministic objective functions used in GA parameter performance evaluation

$f_1(x) = x^{10}$	$0 \le x \le 1$
$f_2(x_i) = \sum_1^3 x_i^2$	$-5.12 \le x_i \le 5.12$
$f_3(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2.048 \le x_i \le 2.048$
$f_4(x_i) = \sum_1^5 integer(x_i)$	$0 \le x_i \le 10.48$

a measure of the convergence and a measure of the ongoing performance. He called these measures, off-line and on-line performance respectively. On-line performance under strategy s of function i can be expressed:

$$O_i(s) = \frac{1}{T} \sum_1^T f_i(t)$$

where $f_i(t)$ is the objective function value for trial t. This is simply the running average of all individuals up to and including individual t. The strategy is defined as the current parameter settings of the genetic algorithm (i.e., population size, crossover probability, mutation probability, etc.). The off-line performance under strategy s of function i can be expressed:

$$O_i^*(s) = \frac{1}{T} \sum_1^T f_i^*(t)$$

where $f_i^*(t) = best\{f_i(1), f_i(2), \ldots, f_i(t)\}$. This is a running average of the best performance values for each generation up to a particular time.

1. Choice of population size

With these performance measures now defined, a thorough investigation of how the performance of a GA is impacted by choice of population size, crossover probability, and mutation probability can be achieved. The population size was the first parameter studied. The population size was set at 50, 100, 200, 300, and 600 individuals while maintaining a constant crossover probability, p_c = 0.6, and mutation probability of p_m = 0.001. The crossover and mutation probabilities were chosen according to previous findings in the literature that these settings are a reasonable compromise between good on-line and off-line performance. The results of this analysis are presented in Figure 4 through Figure 11.

As can be seen in the figures, off-line performance tends to improve as the population size increases. This can be explained by the greater number of individuals in the gene pool from which a best performer can be drawn. On-line performance, on the other hand, tends to improve as population size decreases. The individuals of the smaller population sizes experience more genetic operations, thus the population contains more "good" performers on average, but may be over-zealous and lose information at certain bit positions.

An example of this can be seen in the GA optimization of f_1 with a population size of 30 (see Figure 12). The optimum solution for this function would be a string consisting entirely of 1's. However, as can be seen, all population strings have a 0 at certain bit positions. The only way to regain 1's in these positions is to perform a mutation operation.

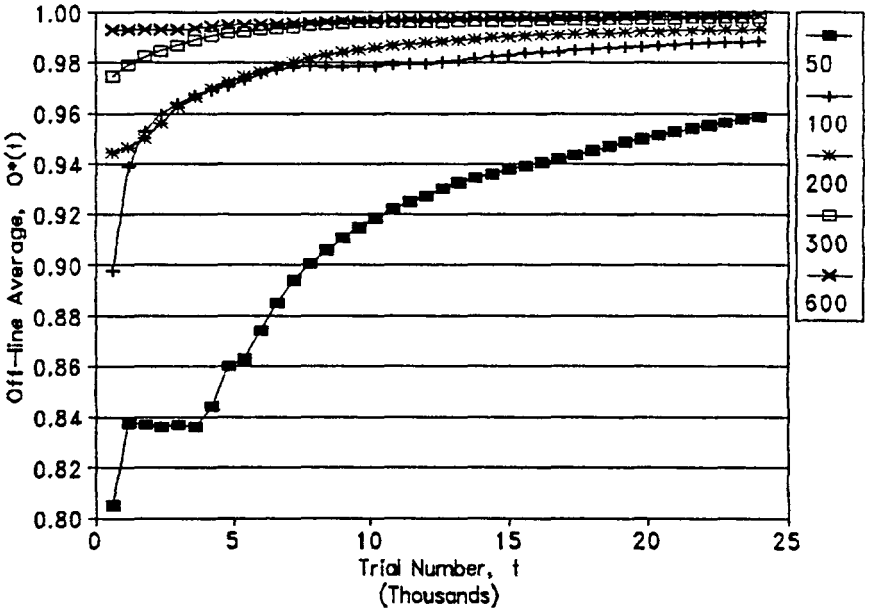

Figure 4. The effects of population size on off-line performance for function f_1

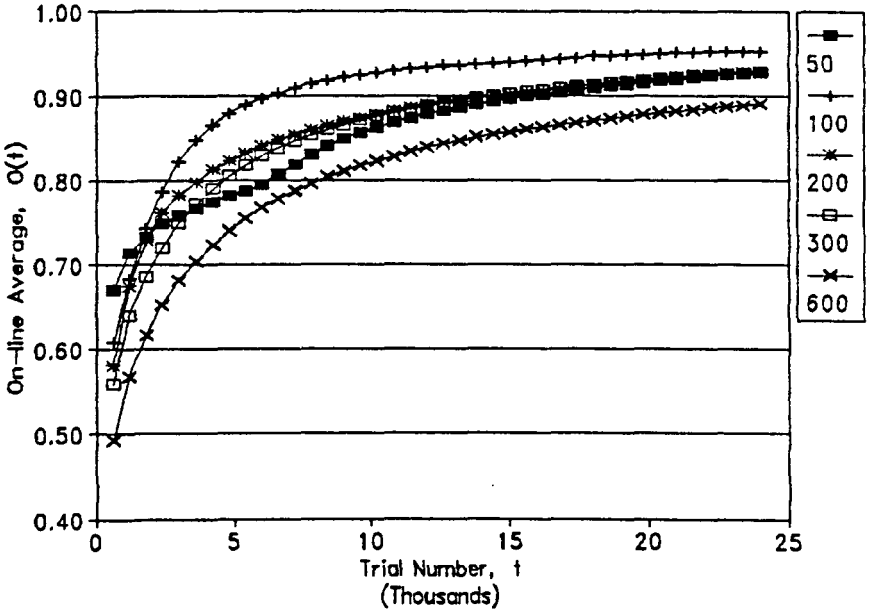

Figure 5. The effects of population size on on-line performance for function f_1

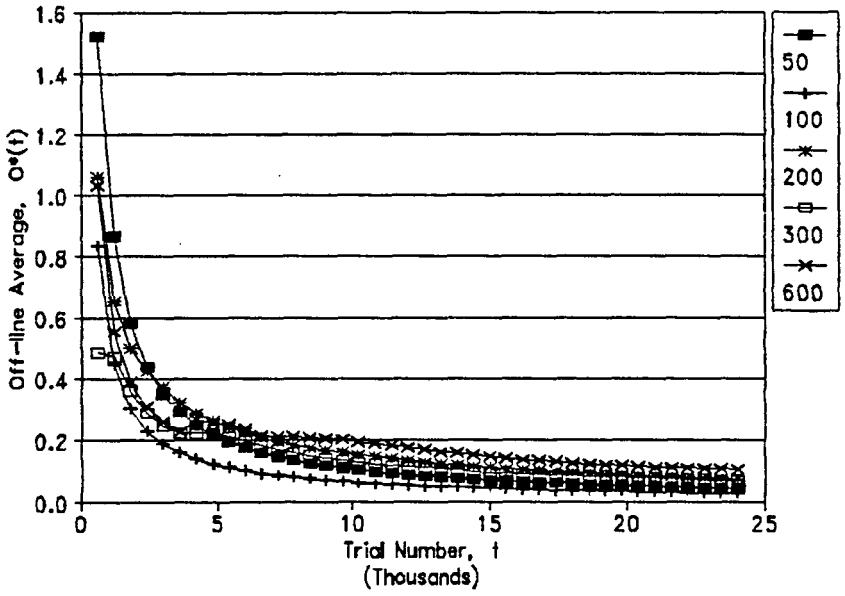

Figure 6. The effects of population size on off-line performance for function f_2

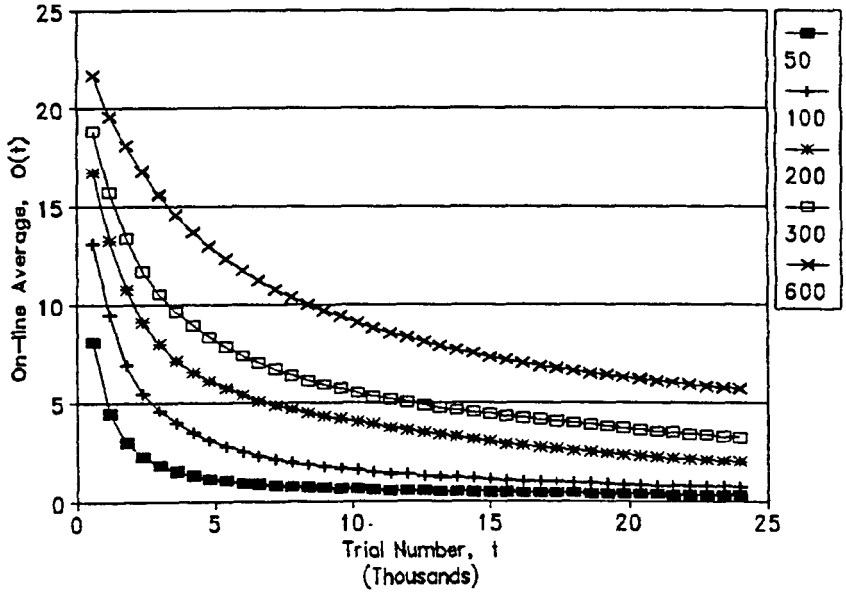

Figure 7. The effects of population size on on-line performance for function f_2

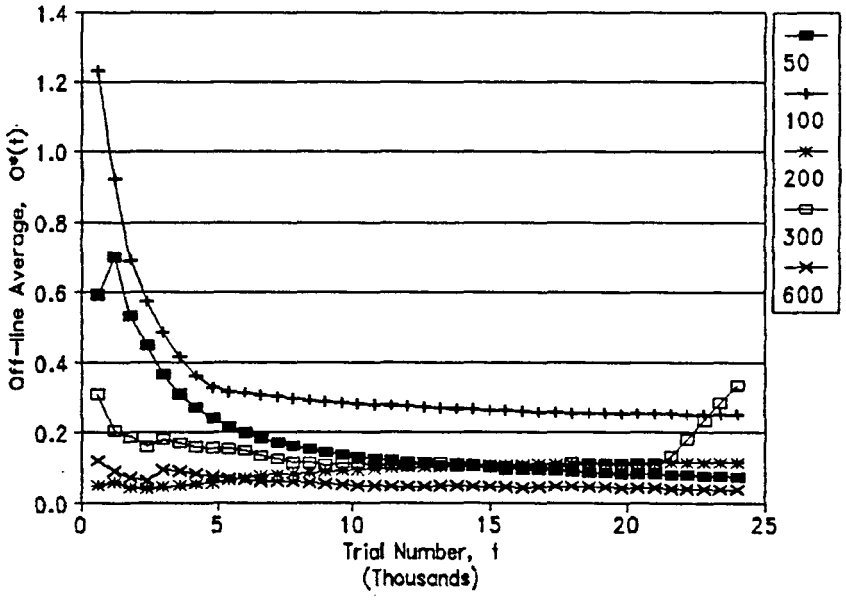

Figure 8. The effects of population size on off-line performance for function f_3

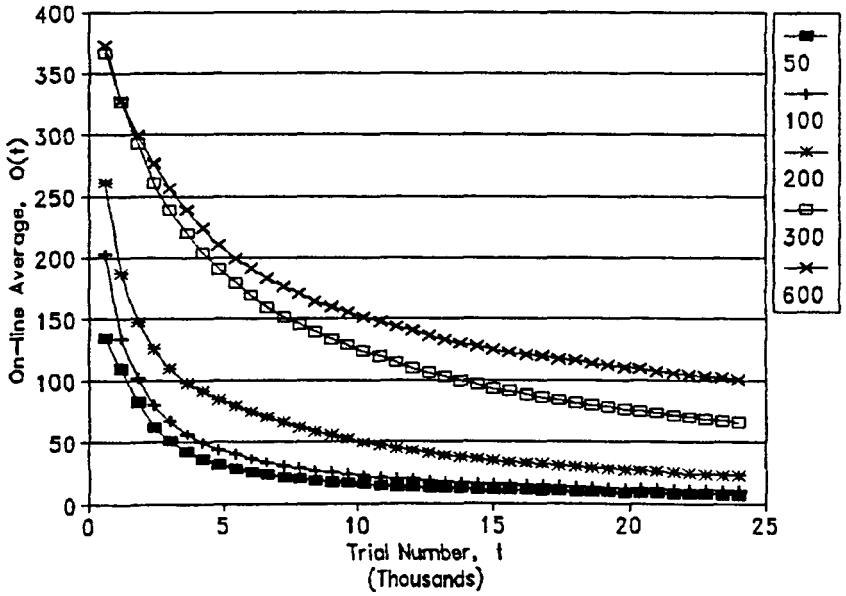

Figure 9. The effects of population size on on-line performance for function f_3

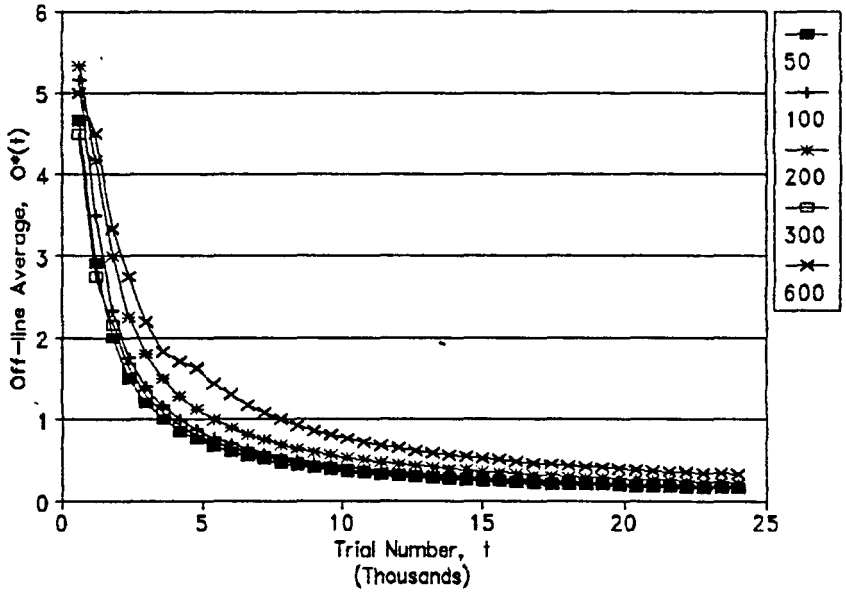

Figure 10. The effects of population size on off-line performance for function f_4

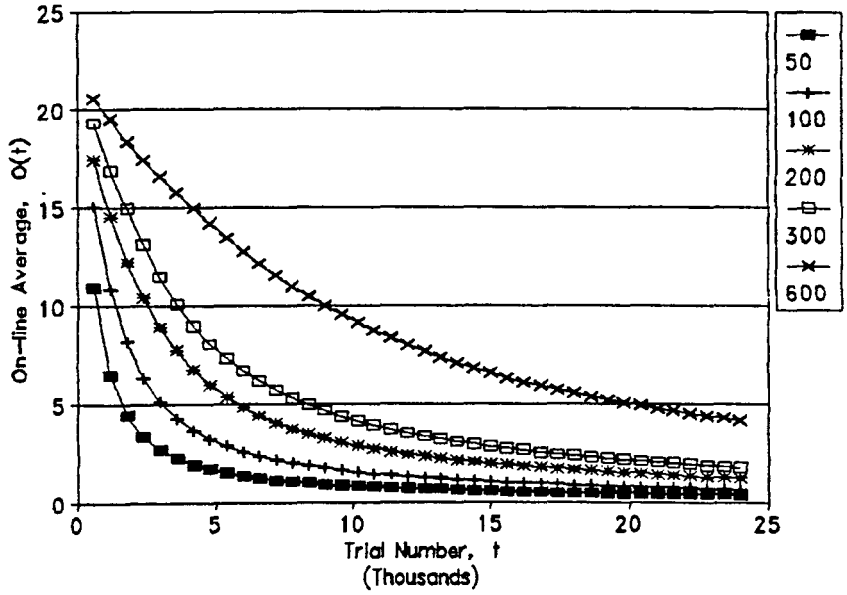

Figure 11. The effects of population size on on-line performance for function f_4

GENETIC ALGORITHM POPULATION REPORT
for FCP file: f1.exe

Population: 19			Population: 20		
Num	String	Objective Fn	Num parents	String	Objective Fn
1)	11111100011000001100110011101	8.67196440e-01	1) (14, 5)	11111100011000001100110011101	8.67196440e-01
2)	11111100011000001100110011101	8.67196440e-01	2) (14, 5)	11111100011000001100110011101	8.67196440e-01
3)	11111100011000001100110011101	8.67196440e-01	3) (2, 3)	11111100011000001100110011101	8.67196440e-01
4)	11111100011000000100110010001	8.67129233e-01	4) (2, 3)	11111100011000001100110011101	8.67196440e-01
5)	11111100011000001100110011101	8.67196440e-01	5) (30,21)	11111100011000001100110011101	8.67196440e-01
6)	11111000011000001100110011101	7.39152133e-01	6) (30,21)	11111100011000001100110011101	8.67196440e-01
7)	11111100011000001100110011101	8.67196440e-01	7) (3,29)	11111100011000001100110011101	8.67196440e-01
8)	11111100011000001100110011101	8.67196440e-01	8) (3,29)	11111100011000001100110011101	8.67196440e-01
9)	11111000011000001100110011101	7.39152133e-01	9) (4,16)	11111100011000001100110011101	8.67196440e-01
10)	11111100011000001100110011101	8.67196440e-01	10) (4,16)	11111100011000000100110010001	8.67129233e-01
11)	11111100011000101100110011101	8.67464923e-01	11) (2,11)	11111100011000101100110011101	8.67464923e-01
12)	11111000011000001100110011101	7.39152133e-01	12) (2,11)	11111100011000001100110011101	8.67196440e-01
13)	11111100011000001100110011101	8.67196440e-01	13) (27, 5)	11111100011000001100110011101	8.67196440e-01
14)	11111100011000001100110011101	8.67196440e-01	14) (27, 5)	11111100011000001100110011101	8.67196440e-01
15)	11111000011000001100110011101	7.39152133e-01	15) (4,20)	11111100011000001100110011101	8.67196440e-01
16)	11111100011000001100110011101	8.67196440e-01	16) (4,20)	11111100011000000100110010001	8.67129233e-01
17)	11111100011000001100110011101	8.67196440e-01	17) (3,19)	11111000011000001100110011101	7.39152133e-01
18)	11111100011000001100110011101	8.67196440e-01	18) (3,19)	11111100011000001100110011101	8.67196440e-01
19)	11111000011000001100110011101	7.39152133e-01	19) (18, 1)	11111100011000001100110011101	8.67196440e-01
20)	11111100011000001100110011101	8.67196440e-01	20) (18, 1)	11111100011000001100110011101	8.67196440e-01
21)	11111100011000001100110011101	8.67196440e-01	21) (16,18)	11111100011000001100110011101	8.67196440e-01
22)	11111100011000001100110011101	8.67196440e-01	22) (16,18)	11111100011000001100110011101	8.67196440e-01
23)	11111100011000001100110011101	8.67196440e-01	23) (23, 9)	11111000011000001100110011101	7.39152133e-01
24)	11111100011000001100110011101	8.67196440e-01	24) (23, 9)	11111100011000001100110011101	8.67196440e-01
25)	11111100011000001000110011101	8.67162885e-01	25) (18,14)	11111100011000001100110011101	8.67196440e-01
26)	11111100011000001100110011101	8.67196440e-01	26) (18,14)	11111100011000001100110011101	8.67196440e-01
27)	11111100011000001100110011101	8.67196440e-01	27) (29,16)	11111100011000001100110011101	8.67196440e-01
28)	11111100011000001000110011101	8.67162885e-01	28) (29,16)	11111100011000001100110011101	8.67196440e-01
29)	11111100011000001100110011101	8.67196440e-01	29) (8,30)	11111100011000001100110011101	8.67196440e-01
30)	11111100011000001100110011101	8.67196440e-01	30) (8,30)	11111100011000001100110011101	8.67196440e-01

Generation totals:
Minimum: 7.39152133e-01 Maximum: 8.67464923e-01 Average: 8.58664622e-01
Mutated: 0 Crossed: 11

Figure 12. Genetic algorithm report for generations 19 and 20 for function f_1

2. Choice of crossover probability

The choice of crossover probability was the next GA parameter studied. To review, the crossover probability is defined as the likelihood of mating two individual strings after reproduction. The implementation of this GA parameter amounts to generating a random number x, then checking whether the random number is less than the crossover probability p_c. If $x < p_c$, perform a crossover to obtain two new child strings, else leave parent strings unchanged in the future generation. The crossover probability was set at 0.2, 0.4, 0.6, 0.8, and 1.0 and the effects upon off-line and on-line performance was observed (see Figure 14 through Figure 20). From the analysis of the choice of population size, a population size of 100 was chosen while maintaining the mutation probability at the previously set level of .001.

In general, a crossover probability of 0.6 or 0.8 seemed to achieve acceptable off-line and on-line performance. The GA performance was observed to be less dependent on choice of crossover probability than for choices of population size and mutation probability. An exception was observed, however, for the function f_3. This can be easily explained by examination of the function itself. The function is minimized when both x_1 and x_2 are equal to 1, but will experience good results whenever $x_1 = x_2$. This leads to high performance schema that have high defining lengths $\delta(H)$. Thus, higher crossover rates will disrupt these schema with a higher probability.

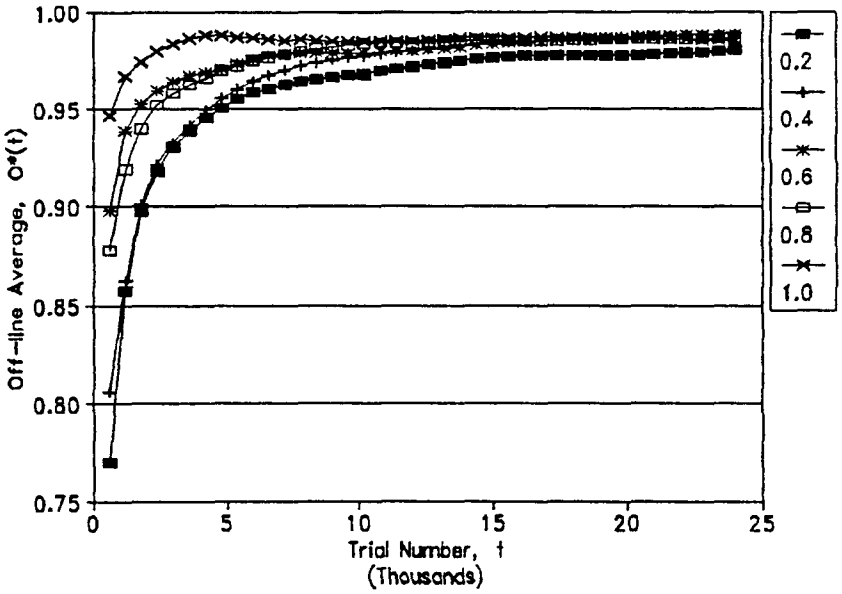

Figure 13. The effects of crossover probability on off-line performance for function f_1

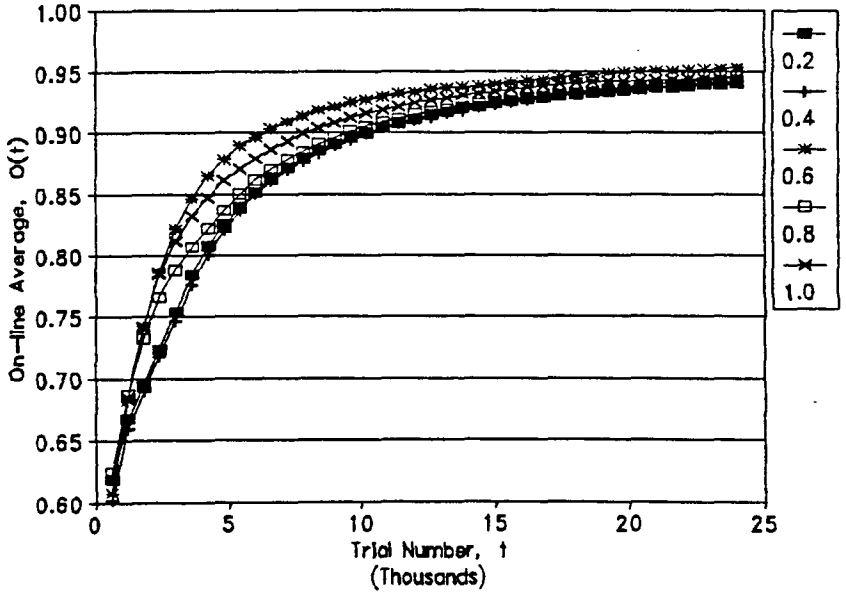

Figure 14. The effects of crossover probability on on-line performance for function f_1

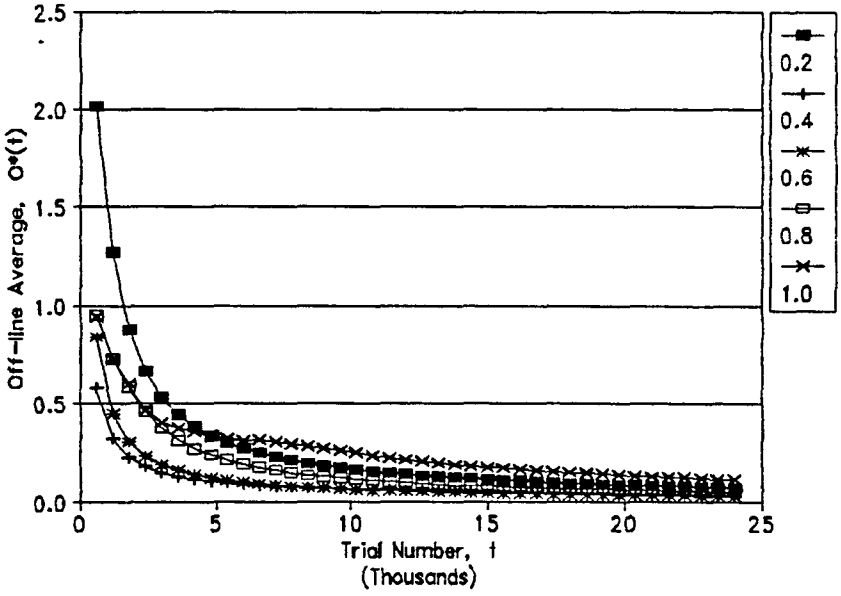

Figure 15. The effects of crossover probability on off-line performance for function f_2

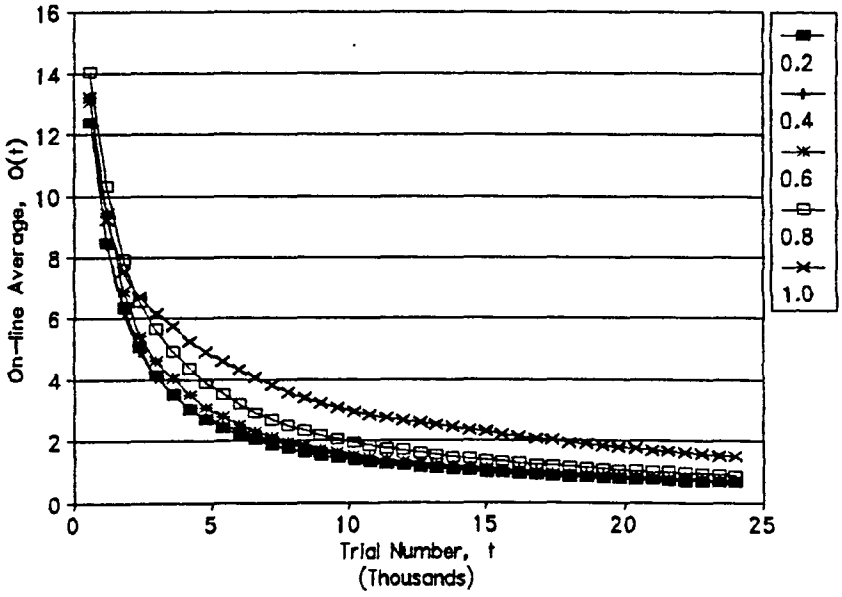

Figure 16. The effects of crossover probability on on-line performance for function f_2

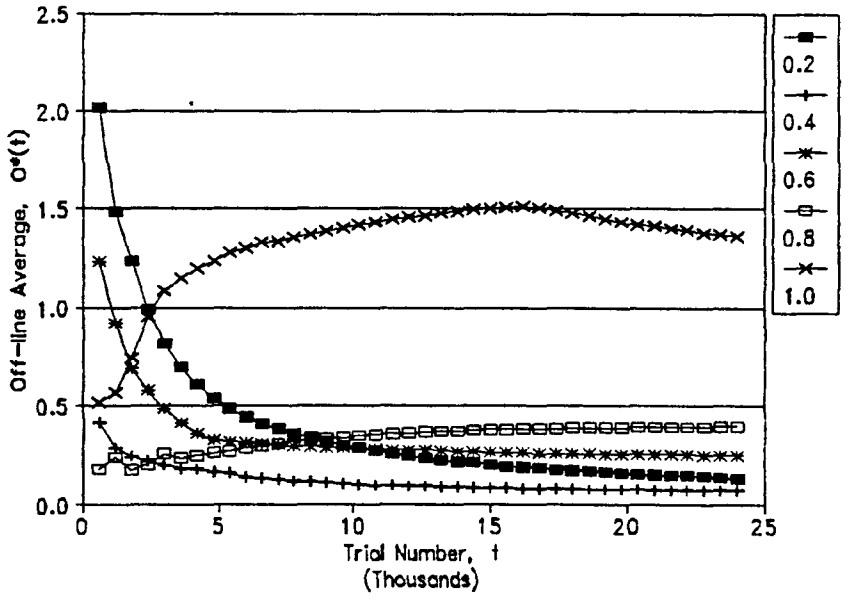

Figure 17. The effects of crossover probability on off-line performance for function f_3

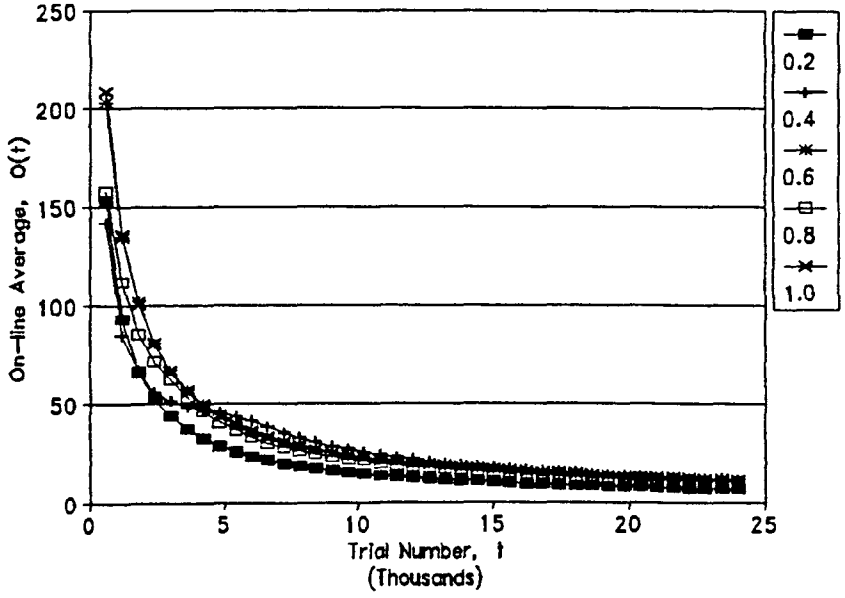

Figure 18. The effects of crossover probability on on-line performance for function f_3

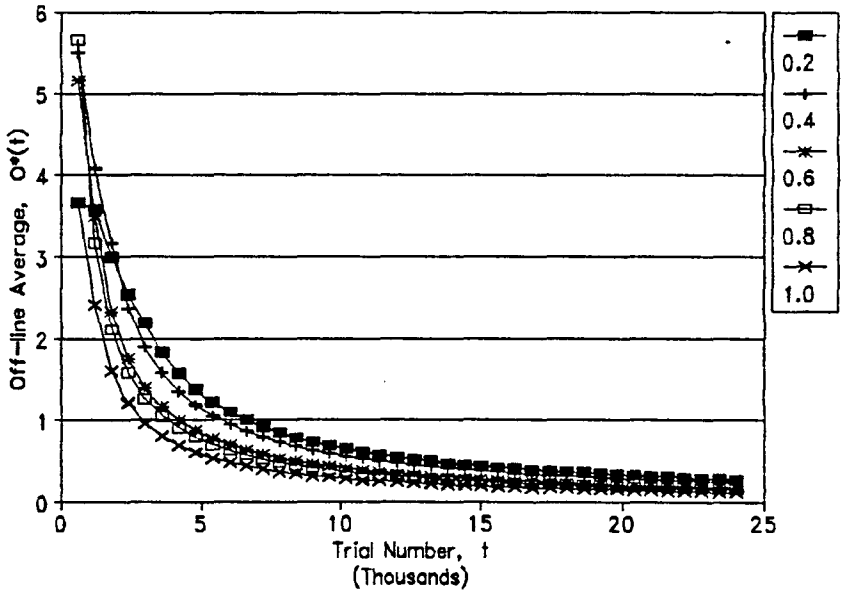

Figure 19. The effects of crossover probability on off-line performance for function f_4

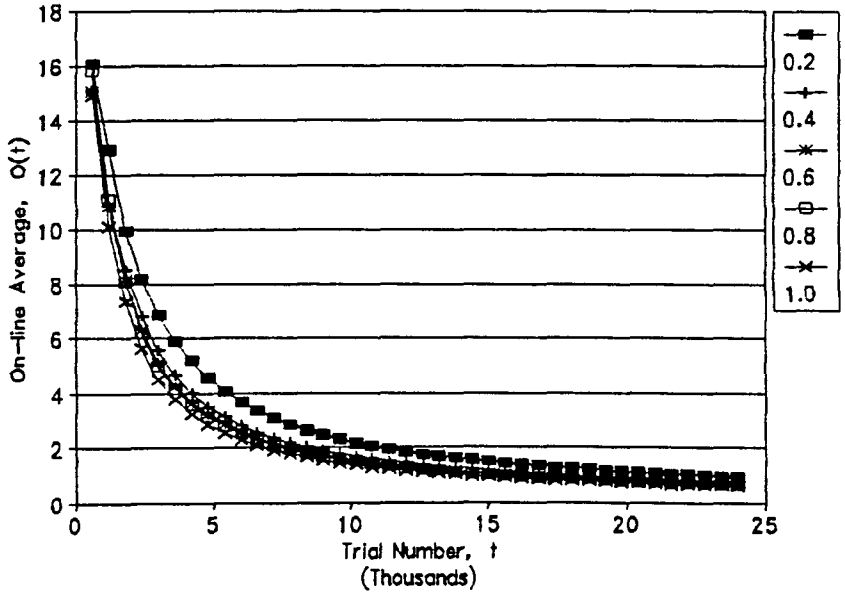

Figure 20. The effects of crossover probability on on-line performance for function f_4

3. Choice of mutation probability

The choice of mutation probability was the final GA parameter studied. Again, the mutation probability involves the switching a bit position from its current state to another state at random. For our case, this amounts to switching a bit from a 1 to a 0 or vice versa. The mutation probability was set at the levels .001, .005, .01, .02, .05, and .1 using a population size of 100 and a crossover probability of 0.6 (see Figure 21 through Figure 28).

Mutation probability was observed to have a large effect on both off-line and on-line performance. The effect was particularly evident with on-line performance. A mutation probability of 0.1 is changing 1 of every 10 bits exchanged during crossover on average. This greatly counteracts the productivity of the crossover and reproduction operations. Also, 0.1 is approaching a mutation probability of 0.5, which is a random walk of the search space at any population size.

A mutation probability of .001 or .005 seemed to enjoy the best performance. At this level, the mutation operation is sufficient to introduce new bit sequences without undermining the reproduction and crossover transformations.

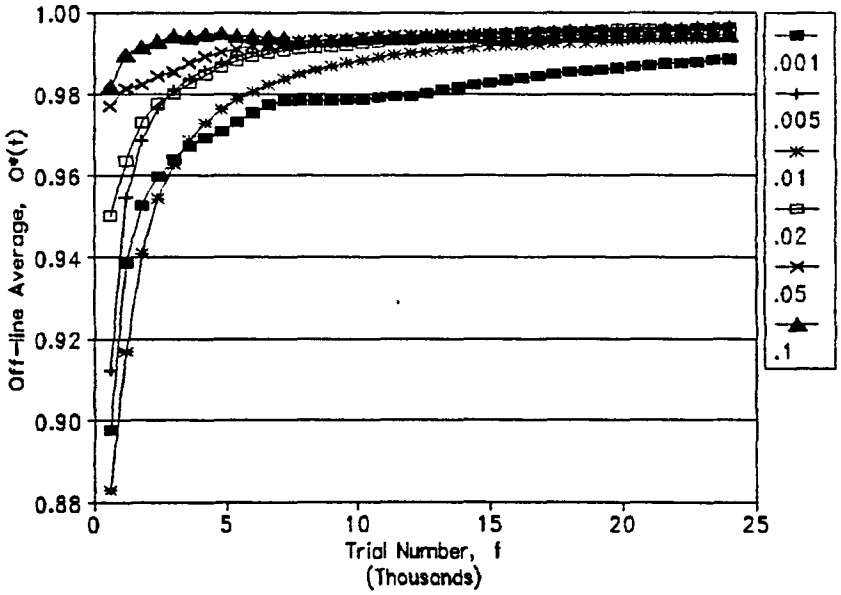

Figure 21. The effects of mutation probability on off-line performance for function f_1

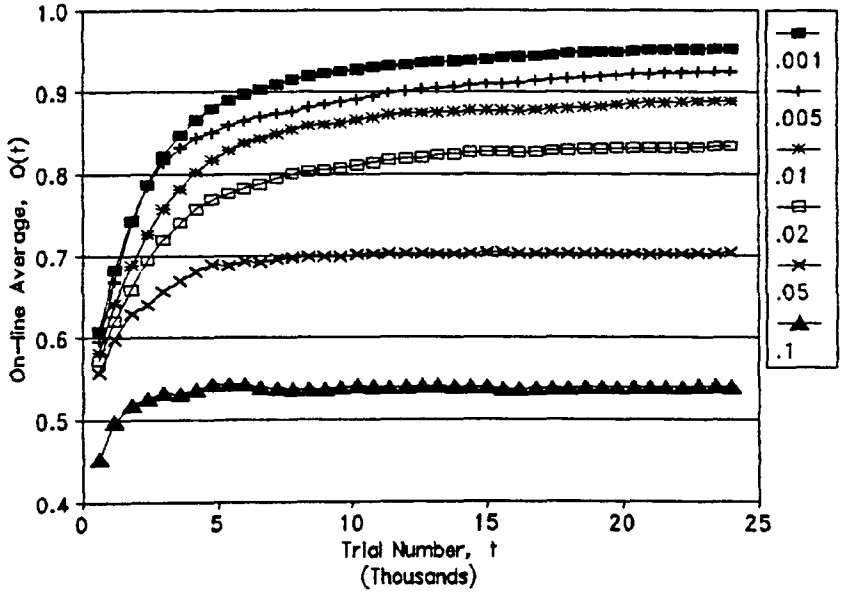

Figure 22. The effects of mutation probability on on-line performance for function f_1

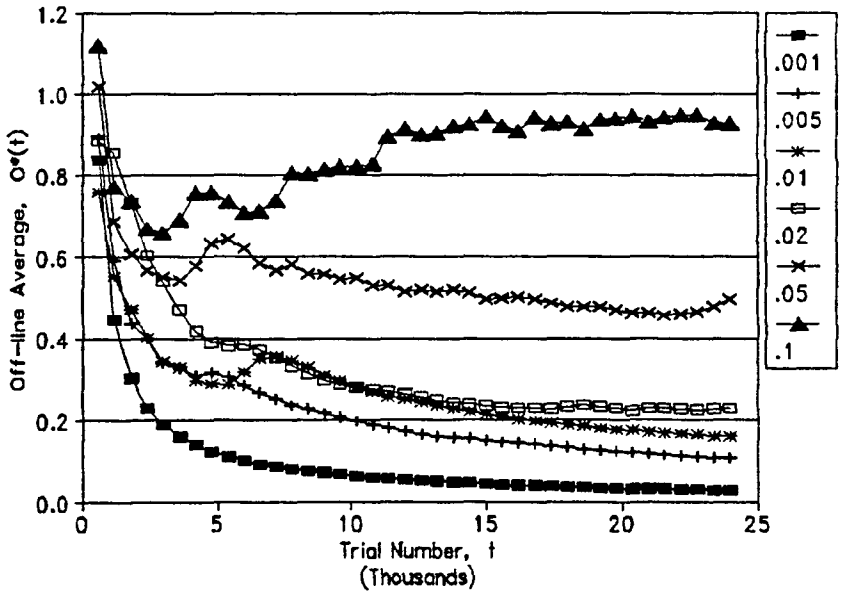

Figure 23. The effects of mutation probability on off-line performance for function f_2

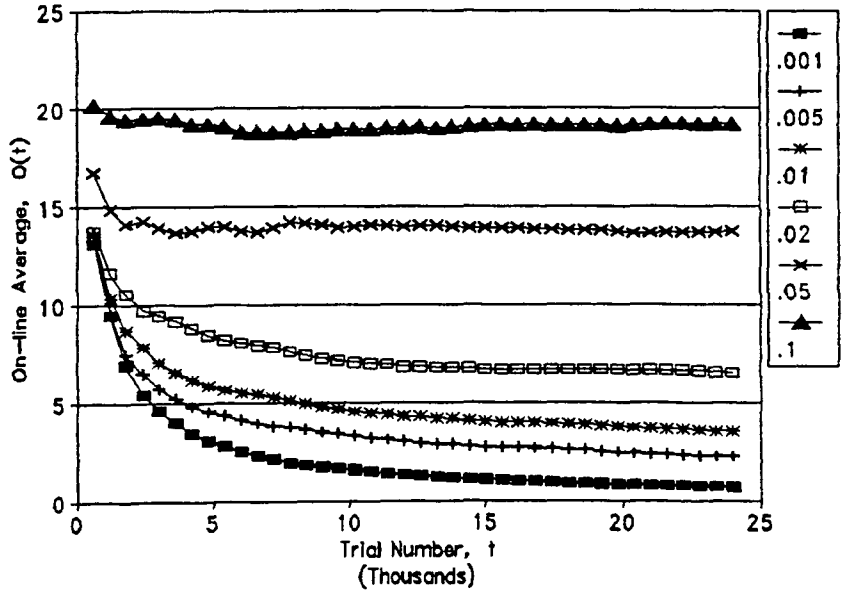

Figure 24. The effects of mutation probability on on-line performance for function f_2

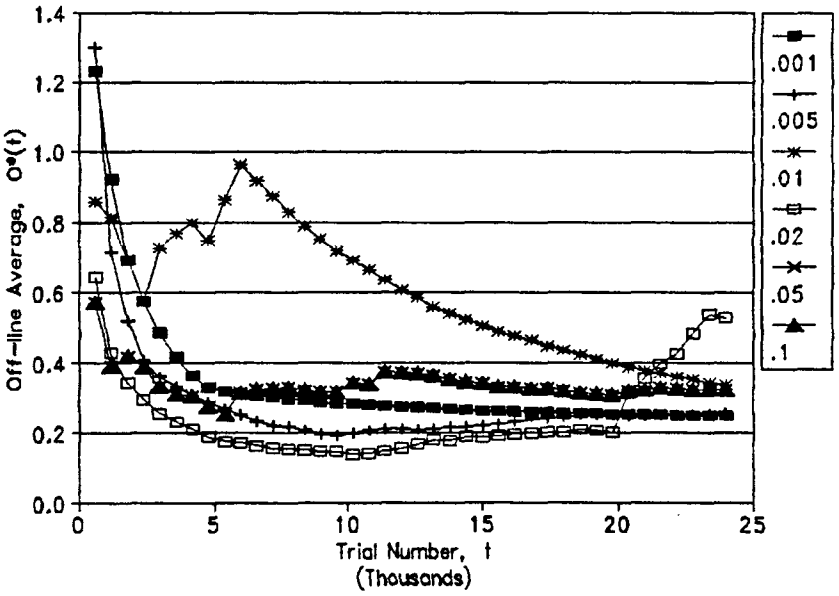

Figure 25. The effects of mutation probability on off-line performance for function f_3

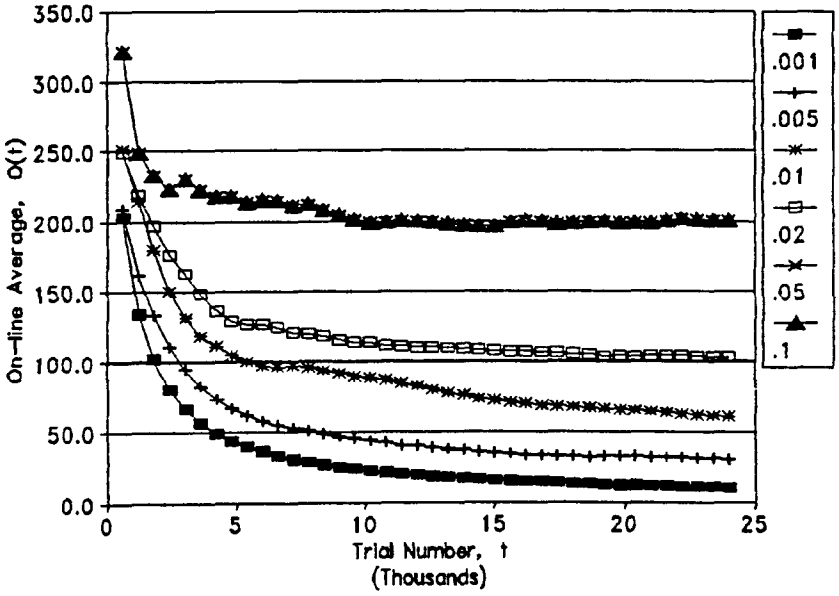

Figure 26. The effects of mutation probability on on-line performance for function f_3

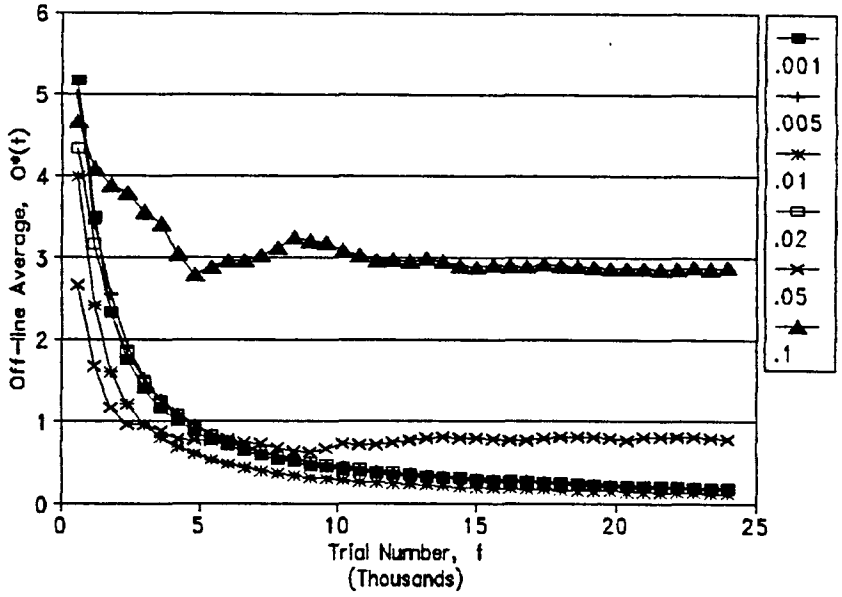

Figure 27. The effects of mutation probability on off-line performance for function f_4

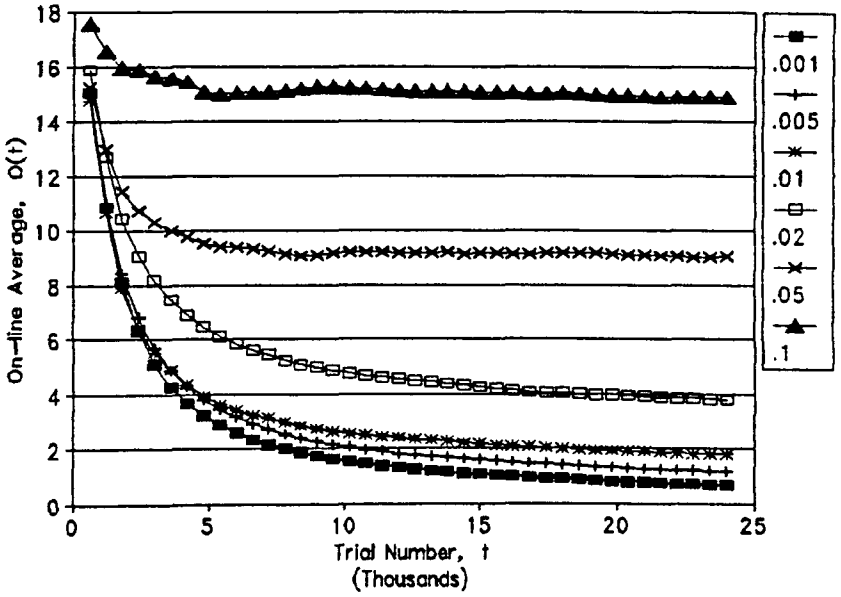

Figure 28. The effects of mutation probability on on-line performance for function f_4

4. Fitness scaling

At the beginning of a GA run, the population most likely contains few high performing individuals with many medium to low performing ones. If reproductive selection using the normal selection criteria ($pselect_j = f_j / \sum f_j$) is allowed, the few high performing individuals dominate the subsequent generation. This is an undesirable effect since many high performing allele positions (gene or bit positions) may be lost early in the run. This phenomenon is a leading cause for premature convergence of a GA.

At the end of GA run, a different problem arises. As the run matures, the population stabilizes and the population average fitness is close to the best fitness value. The reproductive selection now tends to produce generations comprised of a high proportion of these average performers, rather than concentrating on those high performing individuals.

In both of these cases, fitness scaling can help enhance the reproductive selection. A linear scaling was proposed by Goldberg [Goldberg, 1989] and is used in this study. The linear scaling function can be expressed as follows:

$$f' = af + b$$

where the scaled average should remain the same as the original average and all scaled observations do not violate the non-negativity restraint. Goldberg suggests using a scaling such that the following expression holds:

$$f'_{max} = C_{mult} \cdot f_{avg}$$

where C_{mult} is the expected number of copies desired for the best

individual in the subsequent generation. Goldberg states that a value of 1.2 to 2 has been successfully used in small populations (n = 50 to 100). If the generation has a few individuals which are far below the average fitness, the value of C_{mult} will have to be reduced. The fitness values can then be scaled such that the population average fitness remains unchanged and $f'_{min} = 0$.

The fitness scaling procedure presented in Goldberg's text was implemented and examined in order to determine if improved GA performance could be realized. The scaled GA runs were compared to the simple GA runs to observe any increased performance (see Figure 29 and Figure 30) for $C_{mult} = 2.0$. As can be seen, fitness scaling increased both off-line and on-line performance when a population size of 50 was used.

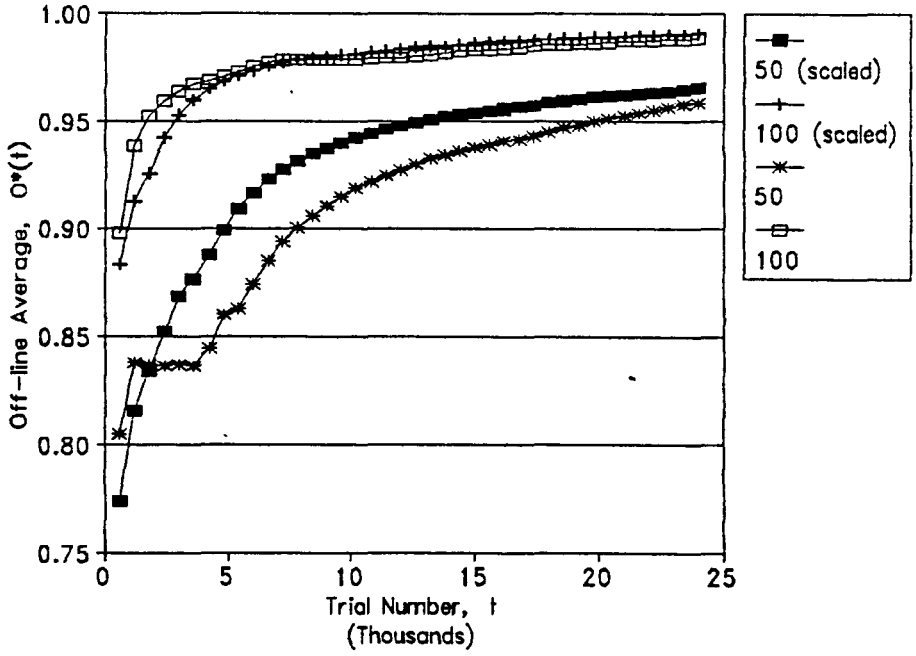

Figure 29. The effects of scaling fitness values on off-line performance for function f_1 (n = 50 and 100)

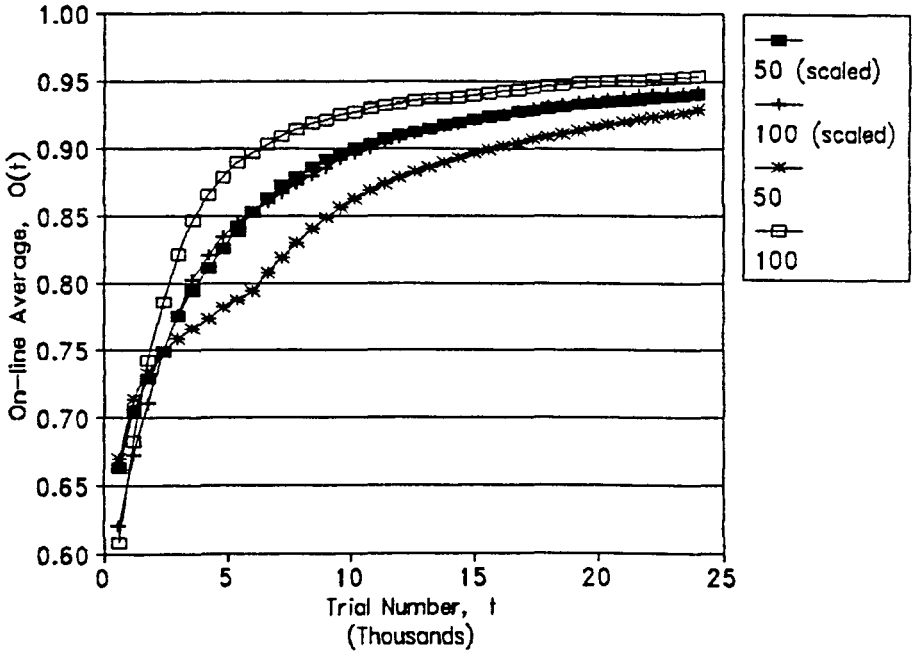

Figure 30. The effects of scaling fitness values on on-line performance for function f_1 (n = 50 and 100)

B. GA Performance on an AAS Simulation Model

As was described previously, this analysis will parallel Liu and Sanders' work where they evaluated the performance of a SQG method as related to the AAS buffer allocation problem. Liu and Sanders tested SQG performance for a variety of models. Of these models, three system configurations were chosen to examine GA performance.

The objective of the GA is to search and locate those buffer allocation configurations which maximize system throughput. Since we are interested in only those highest performing individuals, we will be concerned primarily with off-line performance of the GA. More importantly we will be interested in whether or not the GA can outperform the SQG method.

The first configuration was tested allowing all buffer storage capacities to vary from 1 to 32 units. Individual station buffer capacity was then able to be represented in 5 bit positions allowing the system to be defined by a string of length $\ell = 50$. Two GA runs were performed having population sizes $n = 50$ and $n = 100$. The off-line performance (see Figure 31) was improved by using the larger population size. However, when a 95% confidence interval was constructed, neither run could outperform the SQG method (see Table 3 and Table 4). However, the GA run having a population size of 100 could not be rejected as an inferior configuration (at a 5% level of significance).

The first configuration was then tested allowing storage capacities to vary from 1 to 16 units. This reduced the search space by 2^{10}

configurations, with the thought that the GA might search out high performing configurations quicker. As is shown in Table 5, the GA performed better than the previous run, but still was unable to outperform the SQG method at a statistically significant level.

The second and third AAS configurations were attempted while keeping the buffer capacities limited between 1 and 16 with a population size of 100. The results (see Table 6 and Table 7) show similar GA performance, however for the third configuration the GA did slightly better than the SQG method. This was not at a statistically significant level, however.

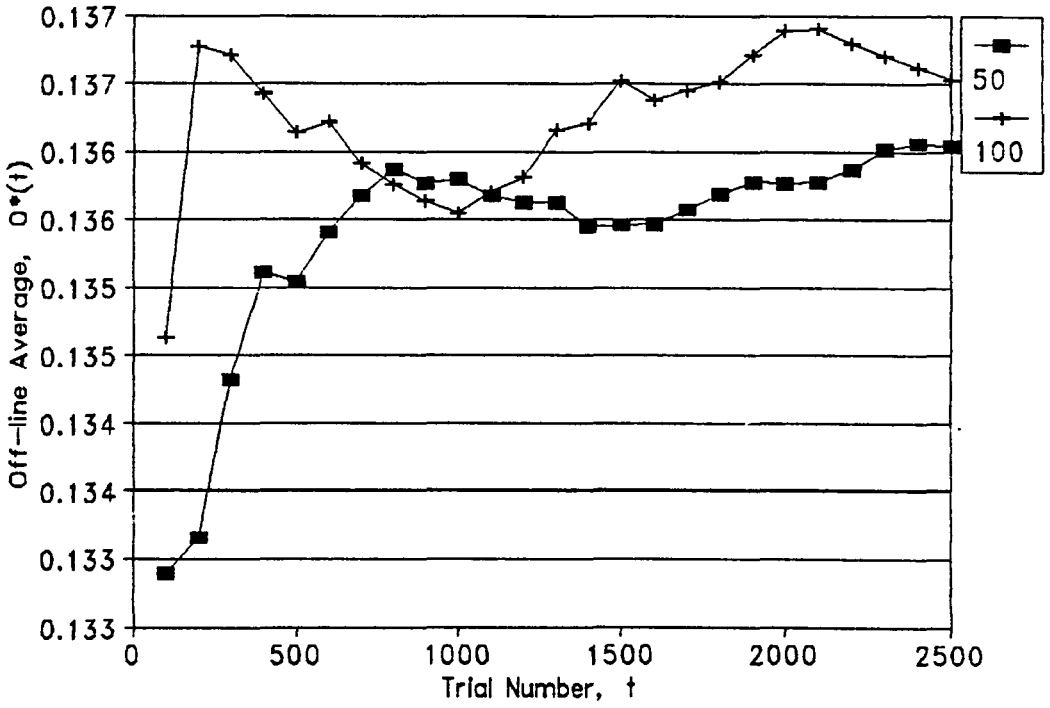

Figure 31. The effects of population size on off-line performance for AAS simulation model throughput for configuration 1

Table 3. Confidence interval estimates for GA and SQG best buffer configurations (buffer capacity allowed to vary from 1 to 32 units, GA population size n = 50)

	Buffer Capacity									
Optim. Method	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
SQG	5	5	17	4	4	4	4	5	5	5
GA	23	19	14	17	17	9	7	5	7	14

Estimation of throughput by C++ simulation model

Optim. Method Buffer Configuration	95% C.I.†
SQG	0.1275 ± 0.0012
GA	0.1249 ± 0.0020
Difference (GA - SQG)	-0.0026 ± 0.0011

† Confidence interval estimates calculated using 10 independent simulation runs of 20,000 assemblies each. For each throughput estimate, the first 10% is removed to in an attempt to eliminate the effects of initial transient. This technique will be used for all remaining tables unless otherwise noted.

Configuration 1 settings:

Total Number of Pallets in System = 40 pallets
Jam rates = (0, 3, 3, 0, 0, 0, 3, 0, 0, 0) per 100 assemblies for stations
Geometric Mean Clear Time = 36 time units
Cycle Time = 5 time units for all stations
Transport Time = 1 time unit per buffer unit

Table 4. Confidence interval estimates for GA and SQG best buffer configurations (buffer capacity allowed to vary from 1 to 32 units, GA population size n = 100)

	Buffer Capacity									
Optim. Method	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
SQG	5	5	17	4	4	4	4	5	5	5
GA	2	20	16	17	11	30	23	8	6	9

Estimation of throughput by C++ simulation model

Optim. Method Buffer Configuration	95% C.I.
SQG	0.1275 ± 0.0012
GA	0.1267 ± 0.0017
Difference (GA - SQG)	-0.0008 ± 0.0010

Configuration 1 settings:

Total Number of Pallets in System = 40 pallets
Jam rates = (0, 3, 3, 0, 0, 0, 3, 0, 0, 0) per 100 assemblies for stations
Geometric Mean Clear Time = 36 time units
Cycle Time = 5 time units for all stations
Transport Time = 1 time unit per buffer unit

Table 5. **Confidence interval estimates for GA and SQG best buffer configurations (buffer capacity allowed to vary from 1 to 16 units, GA population size = 100)**

	Buffer Capacity									
Optim. Method	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
SQG	5	5	17	4	4	4	4	5	5	5
GA	11	12	13	3	16	11	11	11	13	12

Estimation of throughput by C++ simulation model

Optim. Method Buffer Configuration	95% C.I.
SQG	0.1275 ± 0.0012
GA	0.1272 ± 0.0020
Difference (GA - SQG)	-0.0003 ± 0.0011

Configuration 1 settings:

Total Number of Pallets in System = 40 pallets
Jam rates = (0, 3, 3, 0, 0, 0, 3, 0, 0, 0) per 100 assemblies for stations
Geometric Mean Clear Time = 36 time units
Cycle Time = 5 time units for all stations
Transport Time = 1 time unit per buffer unit

Table 6. Confidence interval estimates for GA and SQG best buffer configurations for system configuration 2

	Buffer Capacity									
Optim. Method	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
SQG	4	4	10	10	12	12	4	4	4	4
GA	13	4	4	13	15	15	7	10	5	6

Estimation of throughput by C++ simulation model

Optim. Method Buffer Configuration	95% C.I.
SQG	0.1289 ± 0.0022
GA	0.1285 ± 0.0015
Difference (GA - SQG)	-0.0004 ± 0.0012

Configuration 2 settings:

Total Number of Pallets in System = 40 pallets
Jam rates = (0, 3, 0, 3, 0, 3, 0, 0, 0, 0) per 100 assemblies for stations
Geometric Mean Clear Time = 36 time units
Cycle Time = 5 time units for all stations
Transport Time = 1 time unit per buffer unit

Table 7. Confidence interval estimates for GA and SQG best buffer configurations for system configuration 3

	Buffer Capacity									
Optim. Method	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
SQG	2	3	4	4	4	2	2	2	3	3
GA	1	12	1	8	8	4	5	6	5	1

Estimation of throughput by C++ simulation model

Optim. Method Buffer Configuration	95% C.I.
SQG	0.1272 ± 0.0019
GA	0.1273 ± 0.0016
Difference (GA - SQG)	0.0001 ± 0.0014

Configuration 3 settings:

Total Number of Pallets in System = 20 pallets
Jam rates = (0, 3, 0, 0, 2, 0, 0, 2, 0, 0) per 100 assemblies for stations
Geometric Mean Clear Time = 36 time units
Cycle Time = 5 time units for all stations
Transport Time = 1 time unit per buffer unit

C. Summary

It has been shown that the GA shows acceptable performance for the AAS buffer allocation problem, but does not show great performance. This comes at the expense of considerable computation time. Liu and Sanders reported a time of 45 minutes was required to complete 10 iterations of the SQG algorithm. Typical execution times for the GA implementation were approximately 5 hours (on an '386 based PC running at 25 MHz). Therefore, unless a better implementation of a GA is discovered, the added computation effort is not merited.

V. CONCLUSION

The preceding chapters described and implemented a simple genetic algorithm and applied this algorithm to the buffer allocation problem of a closed-loop, asynchronous, automatic assembly system. The analysis also involved the investigation of GA parameter settings and how these parameters affect GA performance.

At this point, a successful implementation of a genetic algorithm on the buffer allocation problem has not been realized. There are several reasons why this might be the case. Since the objective function is stochastic, we investigate maximizing a point estimate of an expectation function, rather than a deterministic function. The natural variation of this estimate leads to an objective function that is inherently "noisy." Therefore, replicating observation points (through using several simulation runs, instead one longer simulation run) might be advantageous. This also might lead to the use of a penalty function, where the variance of the point estimate could be incorporated into the objective function; thus, "penalizing" those observations that have a high variance.

The computational requirements for a GA run were quite large. The simple fact that the GA might not have had enough time to properly mature could be another explanation for lack of performance.

Though the genetic algorithm did not outperform the SQG method, the results were somewhat encouraging. The GA does generate a large variety of system configurations, which the design analyst may or may not have considered. Since the algorithm uses blind inference, this can be beneficial

in locating system designs that might have been overlooked. The GA also has the advantage of being totally automatic, thus a system designer does not need to use the algorithm interactively.

Future research could be directed in several areas. More analysis is required to determine what constitutes good GA parameter settings. This is especially needed when a GA is applied to a stochastic objective.

Also, more analysis is required to determine how much effort should be given to generating an objective function estimate. With the execution time being critical, it is important not to run the model an unnecessarily long period. Perhaps a method could be devised that would increase the simulation run length as the number of generations increased. The use of penalty functions could also be investigated.

With the use of distributed processing computers (i.e., computers with parallel processor architectures), the long execution times might be reduced sufficiently to make the GA more appealing. Since the genetic algorithm searches many regions in parallel, the algorithm would be well suited for implementation on a parallel processor.

In summary, the genetic algorithm gave encouraging performance on optimizing the buffer sizes for this particular system configuration. Since neither the SQG method or the GA arose as a qualified winner, both seem to be adequate approaches to the buffer allocation problem.

REFERENCES

Bastani, A. S. *IIE Transactions* **1990**, 22(4), 351-360.

Bethke, A. D. Ph.D. dissertation, University of Michigan, 1981; *Dissertation Abstracts International*, 41(9), 3503B.

Boothroyd, G.; Poli, C.; Murch, L. E. *Automatic Assembly*; Marcel Dekker: New York, 1982.

Bratley, P.; Fox, B. L.; Schrage, L. E. *A Guide to Simulation: Second Edition*; Springer-Verlag: New York, 1987.

Buzacott, J. A. *AIIE Transactions* **1972**, 4(4), 308-312.

Buzacott, J. A.; Hanifin, L. E. *AIIE Transactions* **1978**, 10(2), 197-207.

Captor, N.; Biller, B. D.; Riggs, A. J.; Tomko, L. M.; Culver, M. C.; "Adaptable-programmable Assembly Research Technology Transfer to Industry"; final report to the National Science Foundation on Grant ISP 78-18773, 1983.

Choong, Y. F.; Gershwin, S. B. *IIE Transactions* **1987**, 19(2), 150-159.

Cleveland, G. A.; Smith, S. F. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 160-169.

Commault, C.; Semery, A. *IIE Transactions* **1990**, 22(2), 133-142.

Dallery Y.; David, R.; Xie, X. L. *IIE Transactions* **1988**, 20(3), 280-283.

Davis, L. in *Proceedings of an International Conference on Genetic Algorithms and Their Application*; Grefenstette, J. J., Ed.; Carnegie-Mellon University: Pittsburgh, 1985; pp. 136-140.

Davis, L. *Genetic Algorithms and Simulated Annealing*; Morgan Kaufmann: London, 1987.

DeJong, K. A. Ph.D. Dissertation, University of Michigan, 1975; *Dissertation Abstracts International*, 36(10), 5140B.

Ermoliev, Y. *Stochastics* **1983**, 9, 1-36.

Ermoliev, Y.; Gaivoronski, A. "Stochastic quasigradient methods and their implementation"; Working Paper WP-84-55, IIASA, Laxenburg, Austria, 1984.

Fogarty, T. C. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 104-109.

Freeman, M. C. *The Journal of Industrial Engineering* **1964**, 15(4), 194-200.

Gemmill, D. D. Ph.D. Dissertation, University of Wisconsin-Madison, 1988.

Goldberg, D. E.; Lingle Jr., R. in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*; Grefenstette, J. J., Ed.; Carnegie-Mellon University: Pittsburgh, 1985; pp. 154-159.

Goldberg, D. E. in *Proceedings of an International Conference on Genetic Algorithms and Their Application*; Grefenstette, J. J., Ed.; Carnegie-Mellon University: Pittsburgh, 1985a; pp. 8-15.

Goldberg, D. E. The Clearinghouse for Genetic Algorithms Report No. 85001, 1985b; University of Alabama: Tuscaloosa.

Goldberg, D. E. in *Proceedings of the Ninth Conference on Electronic Computation*; Will, K. M., Ed.; American Society of Civil Engineers: New York, 1986; pp. 471-482.

Goldberg, D. E.; Richardson, J. in *Proceedings of the Second International Conference on Genetic Algorithms*; Grefenstette, J. J., Ed.; Lawrence Erlbaum Associates: Hillsdale, New Jersey, 1987; pp. 41-49.

Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley: New York, 1989.

Goldberg, D. D. The Clearinghouse for Genetic Algorithms Report No. 90003, 1990; University of Alabama: Tuscaloosa.

Gordon, W. J.; Newell, G. F. *Operations Research* **1967**, 15(2), 254-265.

Grefenstette, J. J.; Gopal, R.; Rosmaita, B.; Van Gucht, D. in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*; Grefenstette, J. J., Ed.; Carnegie-Mellon University: Pittsburgh, 1985, pp. 160-168.

Grefenstette, J. J.; Baker, J. E. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 20-27.

Hajek, B. *Mathematics of Operations Research* **1988**, 13, 311-329.

Hatcher, J. M. *AIIE Transactions* **1969**, 1(2), 150-156.

Hillier, F. S.; Boling, R. W. *The Journal of Industrial Engineering* 1966, 17(12), 651-658.

Holland, J. H. *Adaptation in Natural and Artificial Systems*; Ann Arbor: The University of Michigan, 1975.

Jackson, J. R. *Management Science* 1963, 10(1), 131-142.

Jafari, M. A.; Shanthikumar, J. G. *IIE Transactions* 1989, 21(2), 130-135.

Jones, D. W. Communications of the ACM 1986, 29(4), 300-311.

Kamath, M.; Sanders, J. L. *Large Scale Systems* 1987, 12, 143-154.

Kirkpatrick, S.; Gelatt Jr., C. D.; Vecchi, M. P. *Science* 1983, 220, 671-680.

Liu, C. M. Ph.D. Dissertation, University of Wiconsin-Madison, 1987.

Liu, C. M.; Sanders, J. L *Annals of Operations Research* 1988, 15, 131-154.

Maza, M. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 208-212.

Metropolis, N.; Rosenbluth, A.; Rosenbluth, M. *Journal of Chemical Physics* 1953, 21, 1087-1091.

Mitra, D.; Romeo, F.; Sangiovanni-Vincentelli *Advanced Applied Probability* 1986, 18, 747-771.

Okamura, K.; Yamashina, H. *International Journal of Production Research* 1983, 21(2), 183-195.

Rao, N. P. *International Journal of Production Research* 1975, 13(2), 207-217.

Richardson, J. T.; Palmer, M. R.; Liepins, G.; Hilliard, M. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 191-197.

Romeo, F.; Sangiovanni-Vincentelli, A. in *1985 Chapel Hill Conference on VLSI*; Fuchs, H., Ed.; Computer Science: Rockville, Maryland, 1985; pp. 393-417.

Sechen, C. *VLSI Placement and Global Routing Using Simulated Annealing*; Kluwer Academic: Boston; 1988.

Sheskin, T. J *AIIE Transactions* 1975, 8(1), 146-152.

Suri, R.; Diehl, G. W. *Management Science* **1986**, 32(2), 206-224.

Syswerda, G. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Mogan Kaufmann: San Mateo, California, 1989; pp. 2-9.

Whitley, D.; Starkweather, T.; Fuqay D. in *Proceedings of the Third International Conference on Genetic Algorithms*; Schaffer, J. D., Ed.; Morgan Kaufmann: San Mateo, California, 1989; pp. 133-140.

Wichmann, B. A.; Hill, I. D. *Applied Statistics* **1982**, 31(2), 188-190.

Wong, D. F.; Leong, H. W.; Liu, C. L. *Simulated Annealing for VLSI Design*; Kluwer Academic: Boston, 1988.

Yeralan, S.; Muth, E. J. *IIE Transactions* **1987**, 19(2), 130-139.