*Computers Math. Applic.* Vol. 23, No. 12, pp. 83–94, 1992
Printed in Great Britain. All rights reserved

# A MODIFIED GENETIC ALGORITHM FOR OPTIMAL CONTROL PROBLEMS

ZBIGNIEW MICHALEWICZ
Department of Computer Science, University of North Carolina, Charlotte, NC 28223, U.S.A.
CEZARY Z. JANIKOW
Department of Computer Science, University of North Carolina, Chapel Hill, NC, 27599-3175, U.S.A.
JACEK B. KRAWCZYK*
Faculty of Commerce and Administration, Quantitative Studies Group
Victoria University of Wellington, PO Box 600, Wellington, New Zealand

*(Received September 1991)*

**Abstract** — This paper studies the application of a genetic algorithm to discrete-time optimal control problems. Numerical results obtained here are compared with ones yielded by GAMS, a system for construction and solution of large and complex mathematical programming models. While GAMS appears to work well only for linear quadratic optimal control problems or problems with short horizon, the genetic algorithm applies to more general problems equally well.

## 1. INTRODUCTION

In this paper we study the numerical optimization of discrete-time dynamic control systems[1]. In general, the task of designing and implementing algorithms for the solution of optimal control problems is a difficult one. The highly touted dynamic programming is a mathematical technique that can be used in variety of contexts, particularly in optimal control [2]. However, this algorithm breaks down on problems of moderate size and complexity, suffering from what is called "the curse of dimensionality" [3].

The optimal control problems are quite difficult to deal with numerically. Some numerical dynamic optimization programs available for general users are typically offspring of the static packages [4] and they do not use dynamic-optimization specific methods. Thus the available programs do not make an explicit use of the Hamiltonian, transversality conditions, etc. On the other hand, if they did use the dynamic-optimization specific methods, they would be even more difficult to be handled by a layman.

Genetic algorithms (GAs, Section 2) require little knowledge of the problem itself. Therefore, computations based on these algorithms are attractive to users without the numerical optimization background. Genetic algorithms have been quite successfully applied to static optimization problems like wire routing, scheduling, transportation problem, traveling salesman problem, etc., [5–10].

On the other hand, to the best of authors' knowledge, only recently GAs have been applied to optimal control problems [1]. We believe that previous GA implementations were too weak to deal with problems where high precision was required. In this paper we present our modification of a GA designed to enhance its performance in dynamic optimization problems, and we show its quality and applicability by a comparative study. As a reference, we use a standard computational package used for solving such problems: the Student Version of General Algebraic Modeling System [4]. We use the MINOS version of the optimizer [11]; in the rest of the paper this system is referred to briefly as GAMS.

*This research was supported by a grant from the North Carolina Supercomputing Center.
[1]This paper is an extended version of [1] which was presented at the 29th CDC (December, 1990).

The remainder of this paper is organized as follows. Section 2 gives an overview of genetic algorithms and Section 3 presents our modifications to the classical design. In Section 4 three optimal control problems with available analytical solutions are formulated, so that the reference points for comparisons are available. In Section 5 the results of application of the genetic algorithm to the control problems are presented. In Section 6 the genetic algorithm's performance is compared with that of GAMS. Section 7 provides some concluding remarks and directions for future work.

## 2. GENETIC ALGORITHMS

Genetic algorithms [6,12] are a class of probabilistic algorithms which begin with a population of randomly generated candidates and "evolve" towards a solution by applying "genetic" operators, modeled on genetic processes occurring in nature. As stated in [13]:

> "...the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The 'knowledge' that each species has gained is embodied in the makeup of the chromosomes of its members. The operations that alter this chromosomal makeup are applied when parents reproduce; among them are random mutation, inversion of chromosomal material, and crossover—exchange of chromosomal material between two parents' chromosomes."

For a given optimization problem (optimization of function $f(x)$), at each iteration $t$ of a genetic algorithm we maintains a population of solutions $P(t) = \{x_1^t, \dots, x_n^t\}$, where $x_i^t$ is a feasible solution, $t$ is an iteration number and $n$ is arbitrarily chosen length of the population. This population would undergo "natural" evolution. In each generation relatively "good" solutions reproduce; the relatively "bad" solutions die out, and are replaced by the offsprings of the former ones. To distinguish between the "good" and "bad" solutions $f(x_i^t)$ is used, which plays a role of the environment (see Figure 1).

```
procedure genetic algorithm begin    t = 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
begin
    t=t+1
    select P(t) from P(t-1)
    recombine P(t)
    evaluate P(t)
    end
end
```

Figure 1. A simple genetic algorithm.

During iteration $t$ the genetic algorithm maintains a population $P(t)$ of some solutions $x_1^t, \dots, x_n^t$ (the population size $n$ remains fixed for the duration of the computation). Each solution $x_i^t$ is evaluated by computing $f(x_i^t)$, which gives us some measure of "fitness" of the solution. Next, at iteration $t+1$ a new population is formed: we select solutions to reproduce on the basis of their relative fitness, and then the selected solutions are recombined using genetic operators (crossover and mutation) to form a new set of solutions.

The *crossover* combines the features of two parent structures to form two similar offspring. Crossover operates by swapping corresponding segments of a string of parents. For example, if parents are represented by five-dimensional vectors, say $x_1 = (a_1, b_1, c_1, d_1, e_1)$ and $x_2 = (a_2, b_2, c_2, d_2, e_2)$, then crossing the vectors between the second and the fifth components would produce the offspring $(a_1, b_1, c_2, d_2, e_1)$ and $(a_2, b_2, c_1, d_1, e_2)$.

The *mutation* operator arbitrarily alters one or more components of a selected structure—this increases the variability of the population. Each bit position of each vector in the new population undergoes a random change with the probability equal to the mutation rate, which is kept constant throughout the computation process.

A genetic algorithm to solve a problem must have five components:

1. A genetic representation of solutions to the problem;
2. A way to create an initial population of solutions;
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness";
4. Genetic operators that alter the composition of children during reproduction; and
5. Values for the parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

In our study we have used a modified genetic algorithm (defined in details in [14] and [15]) which for static optimization problems performed much better than the "simple" GA. The next section defines and discusses modifications and in Section 5 we examine how our modified GA works on the chosen control problem.

### 2.1. *A Modified Genetic Algorithm*

The modified genetic algorithm uses the floating point representation, some new (specialized) genetic operators and other enhancements; we discuss them briefly in turn.

### 2.2. *Representation*

In floating point representation each chromosome vector is coded as a vector of floating point numbers of the same length as the solution vector. Each element is initially selected as to be within the desired domain, and the operators are carefully designed to preserve this constraint (there is no such problem in the binary representation, but the design of the operators is rather simple; we do not see that as a disadvantage; on the other hand, it provides for other advantages mentioned below).

The precision of such an approach depends on the underlying machine, but is generally much better than that of the binary representation. Of course, we can always extend the precision of the binary representation by introducing more bits, but this considerably slows down the algorithm [16].

In addition, the floating point representation is capable of representing quite large domains (or cases of unknown domains). On the other hand, the binary representation must sacrifice the precision for an increase in domain size, given fixed binary length. Also, in the floating point representation it is much easier to design special tools for handling nontrivial constraints. This is discussed fully in [17].

### 2.3. *The Specialized Operators*

The operators we use are quite different from the classical ones, as they work in a different space (real valued). However, because of intuitional similarities, we will divide them into the standard classes: mutation and crossover. In addition, some operators are non-uniform, i.e., their action depends on the age of the population.

MUTATION GROUP:

- **uniform mutation**, defined similarly to that of the classical version: if $x_i^t = \langle v_1, \dots, v_m \rangle$ is a chromosome, then each element $v_k$ has exactly equal chance of undergoing the mutative process. The result of a single application of this operator is a vector $\langle v_1, \dots, v_k', \dots, v_m \rangle$, with $1 \le k \le n$, and $v_k'$ is a random value from the domain of the corresponding parameter

- **non-uniform mutation** is one of the operators responsible for the fine tuning capabilities of the system. It is defined as follow: if $x_i^t = \langle v_1, \dots, v_n \rangle$ is a chromosome, then each element $v_k$ has exactly equal chance of undergoing the mutative process. The result of a single application of this operator is a vector $\langle v_1, \dots, v_k', \dots, v_n \rangle$, with $k \in 1..n$,

$$v_k' = \begin{cases} v_k + \Delta(t, UB - v_k) & \text{if a random digit is 0} \\ v_k - \Delta(t, v_k - LB) & \text{if a random digit is 1} \end{cases}$$

  and $UB$ and $LB$ defined from: domain$_k = (LB, UB)$. The function $\Delta(t,y)$ returns a value in the range $[0,y]$ such that the probability of $\Delta(t,y)$ being close to 0 increases as $t$ increases. This property causes this operator to search the space uniformly initially (when $t$ is small), and very locally at later stages. We have used the following function:

$$\Delta(t,y) = y \cdot \left(1 - r^{(1-t/T)^b}\right),$$

  where $r$ is a random number from $[0 \dots 1]$, $T$ is the maximal generation number, and $b$ is a system parameter determining the degree of non-uniformity.

  Moreover, in addition to the standard way of applying mutation we have some new mechanisms: e.g., the non-uniform mutation is also applied to a whole solution vector rather than a single element of it, causing the whole vector to be slightly slipped in the space.

CROSSOVER GROUP:

- **simple crossover**, defined in the usual way, but with the only permissible split points between $v$'s, for a given chromosome.
- **arithmetical crossover** is defined as a linear combination of two vectors: if $s_v^t$ and $s_w^t$ are to be crossed, the resulting offspring are $s_v^{t+1} = a \cdot s_w^t + (1-a) \cdot s_v^t$ and $s_w^{t+1} = a \cdot s_v^t + (1-a) \cdot s_w^t$ This operator can use a parameter $a$ which is either a constant (uniform arithmetical crossover), or a variable which value depends on the age of population (non-uniform arithmetical crossover).

Here again we have some new mechanisms to apply these operators; e.g., the arithmetical crossover may be applied either to selected elements of two vectors or to the whole vectors.

There are some papers e.g., [18] which suggest that genetic algorithms are not fundamentally different than "evolutionary programming techniques," which relay on random mutation and hill-climbing only. In other words, they suggest that the usage of more complex genetic operators than mutation is not improving the algorithm's convergence. The results of our experiments with modified GA contradict that view. The crossover operators are very important in exploring promising areas in the search space and are responsible for early (but not premature) convergence; a decrease in the crossover rates deteriorate the performance of our system. However, it seems that the significance of particular operators changes with the age of the evolution process: it is worthwhile to explore it further.

## *2.4. Other Enhancements*

However, there are some problems that such applications encounter that sometimes delay, if not prohibit, the finding of the optimal solutions with a desired precision. Such problems originate from many sources as

1. the coding, which moves the operational search space away from the problem space,
2. insufficient number of iterations,
3. insufficient population size, etc.,

and manifest themselves as:

1. premature convergence of the entire population to a non-global optimum, and
2. inability to perform fine local tuning.

There are a number of enhancements implemented in the modified version of the algorithm which deal with the convergence problem and the fine local tuning. Our approach is to

1. detect such troublesome cases, and
2. make the GA to adjust appropriately.

For example, a detected multimodal function would cause a much smaller initial influence of the fitness on the selection process. For a detailed discussion on these topics, see [19].

## 3. TEST PROBLEMS

Three simple discrete-time optimal control models (frequently used in applications of optimal control) have been chosen as test problems for the modified genetic algorithm: linear-quadratic problem, the harvest problem, and the (discretized) push-cart problem. These are apparently easy convex optimal control problems. However, a renowned optimization package (though it is a 'Student Version' only) has had serious troubles in finding the optimal solution to one of them. GA, on the contrary, did not suffer any difficulty in solving any of them. We discuss these three problems in turn.

### *3.1. The Linear-Quadratic Problem*

The first test problem is a one-dimensional linear-quadratic model:

$$\text{minimize} \quad q \cdot x_N^2 + \sum_{k=0}^{N-1} (s \cdot x_k^2 + r \cdot u_k^2) \tag{1}$$

$$\text{subject to} \quad x_{k+1} = a \cdot x_k + b \cdot u_k, \qquad k = 0, 1, \dots, N-1, \tag{2}$$

where $x_0$ is given, $a, b, q, s, r$ are given constants, $x_k \in R$, is the state and $u_k \in R$ is the control of the system.

The value for the optimal performance of (1) subject to (2) is

$$J^* = K_0 \, x_0^2, \tag{3}$$

where $K_k$ is the solution of the Riccati equation

$$K_k = s + ra^2 \frac{K_{k+1}}{r + b^2 K_{k+1}} \qquad \text{and} \qquad K_N = q. \tag{4}$$

In the sequel, the problem (1) subject to (2) will be solved for the sets of the parameters displayed in Figure 2.

In the experiments the value of $N$ was set at 45 as this was the largest horizon for which a comparative numerical solution from GAMS was still achievable.

### *3.2. The Harvest Problem*

The harvest problem is defined as

$$\text{maximize} \quad \sum_{k=0}^{N-1} \sqrt{u_k} \tag{5}$$

$$\text{subject to the equation of growth} \quad x_{k+1} = a \cdot x_k - u_k \tag{6}$$

and one equality constraint (7)

| Case | $N$ | $x_0$ | $s$ | $r$ | $q$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|
| I | 45 | 100 | 1 | 1 | 1 | 1 | 1 |
| II | 45 | 100 | 10 | 1 | 1 | 1 | 1 |
| III | 45 | 100 | 1000 | 1 | 1 | 1 | 1 |
| IV | 45 | 100 | 1 | 10 | 1 | 1 | 1 |
| V | 45 | 100 | 1 | 1000 | 1 | 1 | 1 |
| VI | 45 | 100 | 1 | 1 | 0 | 1 | 1 |
| VII | 45 | 100 | 1 | 1 | 1000 | 1 | 1 |
| VIII | 45 | 100 | 1 | 1 | 1 | 0.01 | 1 |
| IX | 45 | 100 | 1 | 1 | 1 | 1 | 0.01 |
| X | 45 | 100 | 1 | 1 | 1 | 1 | 100 |

Figure 2. Ten test cases.

where initial state $x_0$ is given, $a$ is a constant, and $x_k \in R$ and $u_k \in R^+$ are the state and the (nonnegative) control, respectively.

The optimal value $J^*$ of (5) subject to (6) and (7) is:

$$J^* = \sqrt{\frac{x_0 \cdot (a^N - 1)^2}{a^{N-1} \cdot (a-1)}}. \tag{8}$$

Problem (5) subject to (6) and (7) will be solved for $a = 1.1$ and the following values of $N = 2,4,10,20,45$.

### *3.3. The Push-Cart Problem*

The push-cart problem consists of maximization of the total distance $x_1(N)$ traveled in a given time (a unit, say), minus the total effort. The system is second order:

$$x_1(k+1) = x_2(k), \tag{9}$$

$$x_2(k+1) = 2x_2(k) - x_1(k) + \frac{1}{N^2}\,u(k) \tag{10}$$

and the performance index to be maximized is:

$$x_1(N) - \frac{1}{2N}\sum_{k=0}^{N-1} u^2(k). \tag{11}$$

For this problem the optimal value of index (11) is:

$$J^* = \frac{1}{3} - \frac{3N-1}{6N^2} - \frac{1}{2N^3}\sum_{k=0}^{N-1} k^2. \tag{12}$$

The push-cart problem will be solved for different values $N = 5,10,15,20,25,30,35,40,45$. Note that different $N$ correspond to the number of discretization periods (of an equivalent continuous problem) rather than to the actual length of the optimization horizon which will be assumed as one.

## 4. EXPERIMENTS AND RESULTS

In this section we present the results of the modified genetic algorithm for the optimal control problems. For all test problems, the population size was fixed at 70, and the runs were made for 40,000 generations. For each test case we have made three random runs and reported the best results; it is important to note, however, that the standard deviations of such runs were almost

| | Generations | | | | | | | Factor |
|---|---|---|---|---|---|---|---|---|
| Case | 1 | 100 | 1,000 | 10,000 | 20,000 | 30,000 | 40,000 | |
| I | 17904.4 | 3.87385 | 1.73682 | 1.61859 | 1.61817 | 1.61804 | 1.61804 | $10^4$ |
| II | 13572.3 | 5.56187 | 1.35678 | 1.11451 | 1.09201 | 1.09162 | 1.09161 | $10^5$ |
| III | 17024.8 | 2.89355 | 1.06954 | 1.00952 | 1.00124 | 1.00102 | 1.00100 | $10^7$ |
| IV | 15082.1 | 8.74213 | 4.05532 | 3.71745 | 3.70811 | 3.70162 | 3.70160 | $10^4$ |
| V | 5968.42 | 12.2782 | 2.69862 | 2.85524 | 2.87645 | 2.87571 | 2.87569 | $10^5$ |
| VI | 17897.7 | 5.27447 | 2.09334 | 1.61863 | 1.61837 | 1.61805 | 1.61804 | $10^4$ |
| VII | 2690258 | 18.6685 | 7.23567 | 1.73564 | 1.65413 | 1.61842 | 1.61804 | $10^4$ |
| VIII | 123.942 | 72.1958 | 1.95783 | 1.00009 | 1.00005 | 1.00005 | 1.00005 | $10^4$ |
| IX | 7.28165 | 4.32740 | 4.39091 | 4.42524 | 4.31021 | 4.31004 | 4.31004 | $10^5$ |
| X | 9971341 | 148233 | 16081.0 | 1.48445 | 1.00040 | 1.00010 | 1.00010 | $10^4$ |

Figure 3. Genetic algorithm for the linear–quadratic problem (1)–(2).

| | Generations | | | | | | |
|---|---|---|---|---|---|---|---|
| N | 1 | 100 | 1,000 | 10,000 | 20,000 | 30,000 | 40,000 |
| 2 | 6.3310 | 6.3317 | 6.3317 | 6.3317 | 6.3317 | 6.3317 | 6.331738 |
| 4 | 12.6848 | 12.7127 | 12.7206 | 12.7210 | 12.7210 | 12.7210 | 12.721038 |
| 8 | 25.4601 | 25.6772 | 25.9024 | 25.9057 | 25.9057 | 25.9057 | 25.905710 |
| 10 | 32.1981 | 32.5010 | 32.8152 | 32.8209 | 32.8209 | 32.8209 | 32.820943 |
| 20 | 65.3884 | 68.6257 | 73.1167 | 73.2372 | 73.2376 | 73.2376 | 73.237668 |
| 45 | 167.1348 | 251.3241 | 277.3990 | 279.0657 | 279.2612 | 279.2676 | 279.271421 |

Figure 4. Genetic Algorithm for the harvest problem (5)–(7).

| | Generations | | | | | | |
|---|---|---|---|---|---|---|---|
| N | 1 | 100 | 1,000 | 10,000 | 20,000 | 30,000 | 40,000 |
| 5 | -3.008351 | 0.081197 | 0.119979 | 0.120000 | 0.120000 | 0.120000 | 0.120000 |
| 10 | -5.668287 | -0.011064 | 0.140195 | 0.142496 | 0.142500 | 0.142500 | 0.142500 |
| 15 | -6.885241 | -0.012345 | 0.142546 | 0.150338 | 0.150370 | 0.150370 | 0.150371 |
| 20 | -7.477872 | -0.126734 | 0.149953 | 0.154343 | 0.154375 | 0.154375 | 0.154377 |
| 25 | -8.668933 | -0.015673 | 0.143030 | 0.156775 | 0.156800 | 0.156800 | 0.156800 |
| 30 | -12.257346 | -0.194342 | 0.123045 | 0.158241 | 0.158421 | 0.158426 | 0.158426 |
| 35 | -11.789546 | -0.236753 | 0.110964 | 0.159307 | 0.159586 | 0.159592 | 0.159592 |
| 40 | -10.985642 | -0.235642 | 0.072378 | 0.160250 | 0.160466 | 0.160469 | 0.160469 |
| 45 | -12.789345 | -0.342671 | 0.072364 | 0.160913 | 0.161127 | 0.161152 | 0.161152 |

Figure 5. Genetic Algorithm for the push-cart problem (9)–(11).

negligibly small. The vectors $\langle u_0, \ldots, u_{N-1}\rangle$ were initialized randomly (but within a desired domains). Figures 3, 4, and 5 report the values found along with intermediate results at some generation intervals. For example, the values in column "10,000" indicate the partial result after 10,000 generations, while running 40,000. It is important to note that such values are worse than those obtained while running only 10,000 generations, due to the nature of some genetic operators. In the next section we compare these results with the exact solutions and solutions obtained from the computational package GAMS.

Note, that the Problem (5)–(7) has the final state constrained. It differs from the Problem(1)–(2) in the sense that not every randomly initialized vector $\langle u_0, \ldots, u_{N-1}\rangle$ of positive real numbers generates an admissible sequence $x_k$ (see condition (6)) such that $x_0 = x_N$, for given $a$ and $x_0$. In our version of genetic algorithm, we have generated a random sequence of $u_0, \ldots, u_{N-2}$, repeated the initialization process: this happened in less than 10% of cases. The same difficulty occurred during the reproduction process. An offspring (after some genetic operations) need not satisfy the constraint: $x_0 = x_N$. In such a case we replaced the last component of the offspring vector $u$ using the formula: $u_{N-1} = a \cdot x_{N-1} - x_N$. Again, if $u_{N-1}$ turns out to be negative, we do not introduce such offspring into new population (again, the number of such cases did not exceed 10%). For an automatic way of handling such constrains see [17].

## 5. GENETIC ALGORITHMS VERSUS OTHER METHODS

In this section we compare the above results with the exact solutions as well as those obtained from the computational package GAMS (with MINOS optimizer).

### 5.1. *The Linear-Quadratic Problem*

Exact solutions of the problems for the values of the parameters specified in Figure 2 have been obtained using Formulae (3) and (4).

To highlight the performance and competitiveness of the genetic algorithm, the same test problems were solved using GAMS. The comparison may be regarded as not totally fair for the genetic algorithm since GAMS is based on search methods particularly appropriate for linear-quadratic problems. Thus the Problem (1)–(2) must be an easy case for this package. On the other hand, if for these test problems the genetic algorithm proved to be competitive, or close to, there would be an indication that it should behave satisfactorily in general. Figure 6 summarizes the results, where columns $D$ refer to the percentage of the relative error.

| | Exact solution | Genetic Algorithm | | GAMS | |
|---|---|---|---|---|---|
| Case | value | value | $D$ | value | $D$ |
| I | 16180.3399 | 16180.3928 | 0.000% | 16180.3399 | 0.000% |
| II | 109160.7978 | 109161.0138 | 0.000% | 109160.7978 | 0.000% |
| III | 10009990.0200 | 10010041.3789 | 0.000% | 10009990.0200 | 0.000% |
| IV | 37015.6212 | 37016.0426 | 0.000% | 37015.6212 | 0.000% |
| V | 287569.3725 | 287569.4357 | 0.000% | 287569.3725 | 0.000% |
| VI | 16180.3399 | 16180.4065 | 0.000% | 16180.3399 | 0.000% |
| VII | 16180.3399 | 16180.3784 | 0.000% | 16180.3399 | 0.000% |
| VIII | 10000.5000 | 10000.5000 | 0.000% | 10000.5000 | 0.000% |
| IX | 431004.0987 | 431004.4182 | 0.000% | 431004.0987 | 0.000% |
| X | 10000.9999 | 10001.0038 | 0.000% | 10000.9999 | 0.000% |

Figure 6. Comparison of solutions for the linear-quadratic problem.

As shown above the performance of GAMS for the linear-quadratic problem is perfect. However, this was not at all the case for the second test problem.

### 5.2. *The Harvest Problem*

To begin with, none of the GAMS solutions was identical with the analytical one. The difference between the solutions were increasing with the optimization horizon as shown below (Figure 7), and for $N > 4$ the system failed to find any value.

It appears that GAMS is sensitive to non-convectness of the optimizing problem and to the number of variables. Even adding an additional constraint to the problem ($u_{k+1} > 0.1 \cdot u_k$) to restrict the feasibility set so that the GAMS algorithm does not "lose itself"[2]) has not helped much (see column "GAMS+"). As this column shows, for optimization horizons sufficiently long there is no chance to obtain a satisfactory solution from GAMS.

[2]This is "unfair" from the point of view of the genetic algorithm which [illegible]

| *N* | Exact solution | GAMS | | GAMS+ | | Genetic Alg | |
|---|---|---|---|---|---|---|---|
| | | value | *D* | value | *D* | value | *D* |
| 2 | 6.331738 | 4.3693 | 30.99% | 6.3316 | 0.00% | 6.3317 | 0.000% |
| 4 | 12.721038 | 5.9050 | 53.58% | 12.7210 | 0.00% | 12.7210 | 0.000% |
| 8 | 25.905710 | * | | 18.8604 | 27.20% | 25.9057 | 0.000% |
| 10 | 32.820943 | * | | 22.9416 | 30.10% | 32.8209 | 0.000% |
| 20 | 73.237681 | * | | * | | 73.2376 | 0.000% |
| 45 | 279.275275 | * | | * | | 279.2714 | 0.001% |

Figure 7. Comparison of solutions for the harvest problem. The symbol '*' means that the GAMS failed to report a reasonable value.

| | Exact solution | GAMS | | GA | |
|---|---|---|---|---|---|
| *N* | value | value | *D* | value | *D* |
| 5 | 0.120000 | 0.120000 | 0.000% | 0.120000 | 0.000 % |
| 10 | 0.142500 | 0.142500 | 0.000% | 0.142500 | 0.000 % |
| 15 | 0.150370 | 0.150370 | 0.000% | 0.150370 | 0.000 % |
| 20 | 0.154375 | 0.154375 | 0.000% | 0.154375 | 0.000 % |
| 25 | 0.156800 | 0.156800 | 0.000% | 0.156800 | 0.000 % |
| 30 | 0.158426 | 0.158426 | 0.000% | 0.158426 | 0.000 % |
| 35 | 0.159592 | 0.159592 | 0.000% | 0.159592 | 0.000 % |
| 40 | 0.160469 | 0.160469 | 0.000% | 0.160469 | 0.000 % |
| 45 | 0.161152 | 0.161152 | 0.000% | 0.161152 | 0.000 % |

Figure 8. Comparison of solutions for the push-cart problem.

| N | No. of iterations needed | Time needed (CPU sec) | Time for 40,000 iterations (CPU sec) | Time for GAMS (CPU sec) |
|---|---|---|---|---|
| 5 | 6234 | 65.4 | 328.9 | 31.5 |
| 10 | 10231 | 109.7 | 400.9 | 33.1 |
| 15 | 19256 | 230.8 | 459.8 | 36.6 |
| 20 | 19993 | 257.8 | 590.8 | 41.1 |
| 25 | 18804 | 301.3 | 640.4 | 47.7 |
| 30 | 22976 | 389.5 | 701.9 | 58.2 |
| 35 | 23768 | 413.6 | 779.5 | 68.0 |
| 40 | 25634 | 467.8 | 850.7 | 81.3 |
| 45 | 28756 | 615.9 | 936.3 | 95.9 |

Figure 9. Time performance of genetic algorithm and GAMS for the push-cart problem (9)–(11): number of iterations needed to obtain the result with precision of six decimal places, time needed for that number of iterations, time needed for all 40,000 iterations.

## *5.3. The Push-Cart Problem*

For the push-cart problem both GAMS and genetic algorithm produce very good results (Figure 8). However, it is interesting to note the relationship between the time different search algorithms need to complete the task.

For most optimization programs, the time necessary for an algorithm to converge to the optimum depends on the number of decision variables. This relationship for dynamic programming is exponential ("curse of dimensionality"). For the search methods (like GAMS) it is usually

Figure 9 reports number of iterations genetic algorithm needed to obtain exact solution (with six decimal place rounding), the time needed for that, and the total time for all 40,000 iterations (for unknown exact solution we can not determine the precision of the current solution). Also, the time for GAMS is given. Note, that GAMS was run on PC Zenith z-386/20, while genetic algorithm on DEC-3100 station.

It is clear, that the genetic algorithm is much slower than GAMS: there is a difference in absolute values of CPU time as well as computers used. However, let us compare not the times needed for both system to complete their calculations, but rather their growth rates of the time as a function of the size of the problem. The Figure 10 show the growth rate of the time needed to obtain the result of the genetic algorithm and GAMS.

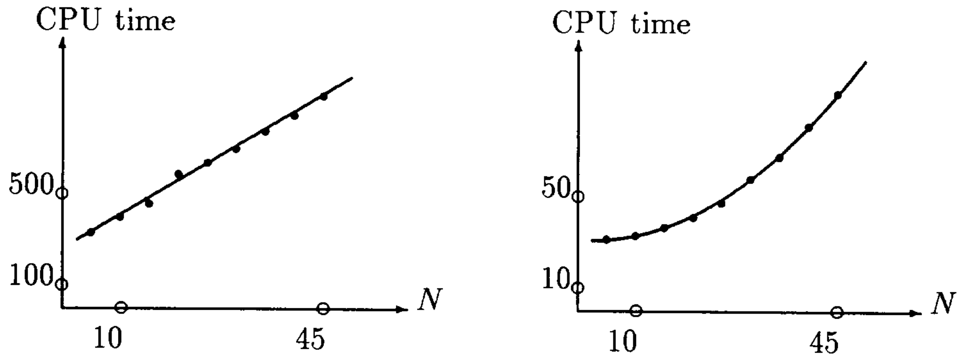

Figure 10. Time as a function of problem size ($N$).

These graphs are self-explanatory: although genetic algorithm is generally slower, its linear growth rate is much better than that of GAMS (which is at least quadratic). Similar results hold for the linear-quadratic problem and the harvest problem.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper we have initiated the study of the application of the (modified) genetic algorithm to discrete-time optimal control problems. The experiments were successful on three selected optimal control problems. In particular, the results were encouraging because the closeness of the numerical solutions to the analytical ones was satisfying, additionally, the coding and computation efforts were reasonable (about 1500 lines of code in C, and, for the 40,000 generations, few minutes of CPU time on CRAY Y-MP and up to 15 minutes on a DEC-3100 station).

The numerical results were compared with those obtained from a search-based computational package (GAMS). While the genetic algorithm gave us results comparable with the analytic solutions for all test problems, GAMS failed for one of them. Additionally, the genetic algorithm displayed some qualities not always present in the other systems:

- The optimization function for genetic algorithm need not be continuous. In the same time some optimization packages will not accept such function at all.
- Some optimization packages are all-or-nothing propositions: the user has to wait until the program completes. Sometimes it is not possible to get partial (or approximate) results at some early stages. Genetic algorithms give the users additional flexibility, since the user can monitor the "state of the search" during the run time and make appropriate decisions. In particular, the user can specify the computation time (s)he is willing to pay for (longer time provides better precision of the answer).
- The computational complexity of genetic algorithms grows at the linear rate; most of other search methods are very sensitive on the length of the optimization horizon. Moreover,

"In a world where serial algorithms are usually made parallel through countless tricks and contortions, it is no small irony that genetic algorithms (highly parallel algorithms) are made serial through equally unnatural tricks and turns."

It seems we can easily improve the performance of our system using parallel implementations; often it is difficult for other optimization methods.

This paper is the first step toward building a control-problem optimization system based on genetic algorithms. Here, precise solutions for three frequently used simple control models were obtained. In general, an optimization system to be interesting for practitioners has to be more control-problem specific than the system here introduced. In particular the system has to allow for:

- inequality/equality local constraints of the state variable,
- inequality/equality local constraints of the control variable,
- mixed inequality/equality local constraints,
- inequality/equality global constraints of the state variable,
- inequality/equality global constraints of the control variable,
- free final time.

In this paper a model with one local equality constraint on the final state has been successfully handled. Future developments of the system will concentrate on the constraints' classes specified above.

## REFERENCES

1. Z. Michalewicz, J. Krawczyk, M. Kazemi and C. Janikow, Genetic algorithms and optimal control problems, In *Proceedings of the 29th IEEE Conference on Decision and Control*, Honolulu, Hawaii, pp. 1664–1666, (December 5–7, 1990).
2. D.P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall, Englewood Cliffs, N.J., (1987).
3. R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, N.J., (1957).
4. A. Brooke, D. Kendrick and A. Meeraus, *GAMS: A User's Guide*, The Scientific Press, (1988).
5. K.A. De Jong, Genetic algorithms: A 10 year perspective, In [7], pp. 169–177 (1985).
6. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, (1989).
7. J.J. Grefenstette, Ed., *Proceedings of the First International Conference on Genetic Algorithms*, Pittsburg, Lawrence Erlbaum Associates, Publishers, (July 24–26, 1985).
8. J.J. Grefenstette, Ed., *Proceedings of the Second International Conference on Genetic Algorithms*, MIT, Cambridge, Lawrence Erlbaum Associates, Publishers, (July 28–31, 1987).
9. J. Schaffer, Ed., *Proceedings of the Third International Conference on Genetic Algorithms*, George Mason University, Morgan Kaufmann, Publishers, (June 4–7, 1989).
10. G.A. Vignaux and Z. Michalewicz, A genetic algorithm for the linear transportation Problem, *IEEE Transactions on Systems, Man, and Cybernetics*, **21** (2), 445–452 (1991).
11. B.A. Murtagh and M.A. Saunders, MINOS 5.1 user's guide, Report SOL 83-20R, Stanford University, (December, 1983, revised January, 1987).
12. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, (1975).
13. L. Davis, Ed., *Genetic Algorithms and Simulated Annealing*, Pitman, London, (1987).
14. C. Janikow and Z. Michalewicz, Specialized genetic algorithms for numerical optimization problems, *Proceedings of the Second International Conference on Tools for AI*, Washington, pp. 798–804, (November 6–9, 1990).
15. Z. Michalewicz and C. Janikow, Genetic algorithms for numerical optimization, *Statistics and Computing*, **1** (2) (1991).
16. C. Janikow and Z. Michalewicz, Experimental comparison of binary and floating point representations in genetic algorithms, *Proceedings of the 4th International Conference on Genetic Algorithms*, San Diego, pp. 31–36 (July 13–16, 1991).

17. Z. Michalewicz and C. Janikow, GENOCOP: A genetic algorithm for numerical optimization problems with linear constraints, *Communications of the ACM*, (1992)(to appear).

18. D.B. Fogel and J.W. Atmar, Comparing genetic operators with Gaussian mutation in simulated evolutionary process using linear systems, *Biol. Cybern.*, **63**, 111–114, (1990).

19. Z. Michalewicz, *Data Structures + Genetic Operators = Evolution Programs*, Springer-Verlag, Symbolic Computation Series, (1992).

20. Z. Michalewicz, G.A. Vignaux and L. Groves, Genetic algorithms for approximation and optimization problems, *Proceedings of the 11th New Zealand Computer Conference*, pp. 211–223, Wellington, (August 16–18, 1989).

21. Z. Michalewicz, G.A. Vignaux and M. Hobbs, A genetic algorithm for the nonlinear transportation problem, *ORSA Journal on Computing*, **3** (4) (1991).

22. H.P. Schwefel, *Numerical Optimization for Computer Models*, J. Wiley & Sons, (1981).