A Specialized Genetic Algorithm for Numerical Optimization Problems

Cezary Z. Janikow* Zbigniew Michalewicz†

Abstract

This paper describes a specialized genetic algorithm for numerical optimization problems and discusses the application of such algorithms to discrete-time optimal control problems. Numerical results obtained here are compared with those obtained from a classical genetic algorithm and a system for construction and solution of large and complex mathematical programming models, GAMS. As this specialized algorithm is only a part of a proposed unified generic package, further extensions are also outlined.

1 Introduction

This paper describes a specialized genetic algorithm suitable for all parameter optimization problems where high precision is required.

Traditionally, genetic algorithms are applicable to a broad range of problems. In particular, the problem of tuning parameters of some system (without any constraints) is considered to be one for which genetic algorithm performs very well. However, there are problems where the domains are unlimited, the number of parameters is quite large, and a high precision is required. These requirements imply that the length of the (binary) solution vector, necessary to be constructed for the algorithm, is quite significant (for 100 variables with domains in the range $\langle -500, 500 \rangle$, where the precision of six decimal digits is required, the length of the binary solution vector is 3000). For such problems the performance of genetic algorithms is quite poor. In addition, it is widely recognized that genetic algorithms are not well suited to perform fine local exploitation [Grefenstette, 1987], which prohibits obtaining high precision solutions; it is widely recognized that for such cases the only solution is to apply some other specilized methods to such approximate solutions obtained from the genetic algorithm.

We introduce a specialized genetic algorithm based on a floating point representation, with several "genetic" operators suitable for performing both space exploration and highly

*Department of Computer Science, University of North Carolina, Chapel Hill, NC, 27599, USA
†Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA

local exploitation. As our intuition suggests, and the experiments support, this algorithm is capable of finding the correct solution to very high precision, otherwise not possible in the sigle algorithm scenario.

As our study case, we selected two optimization problems of discrete time dynamic control systems. As it is well known, the task of designing and implementing algorithms for solving the 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 (*cf.* [Bertsekas, 1987]). However, this approach breaks down on problems of moderate size and complexity, suffering from what is called the "curse of dimensionality" [Bellman, 1957].

In particular, optimal control problems are quite difficult to deal with numerically. The numerical dynamic optimization programs available for general users are typically offspring of the static packages [Brooke et al., 1988], and they do not use dynamic–optimization specific methods; they 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 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 [Goldberg, 1989], [DeJong, 1985], [Vignaux & Michalewicz, 1989a,b,c] applied to static optimization problems like wire routing, scheduling, transportation problem, travelling salesman problem, *etc.*, but, to the best of authors' knowledge, they have not been applied to optimal control problems: due to mentioned size of the representation vector, and to high precision requirements. In this paper we present a first step toward a genetic algorithm based standard package: here we try to design a system able to overcome both the size and precision problems (which are strongly related). In other related publications (*e.g.* [Michalewicz et al., 1990), we present an extention able to deal with any class of linear constraints. Such a system, when implemented, may prove to be a very powerful, yet user friendly, tool in solving the broad class of numerical optimization problems with linear constraints, with a precision higher than that of currently available packages.

For a comparative evaluation of the performance of the genetic algorithm, we use a version (called *Student Version*) of a commertial computational package for construction and solution of large and complex mathematical programming models, called GAMS ([Brooke et al., 1987]). In the rest of the paper we will refer to this system as GAMS only.

The remainder of this paper is organized as follows. Section 2 gives an overview of genetic algorithms. In Section 3 two simple optimal control problems are formulated, and solved analytically so that the reference points for comparisons are available. In Section 4 the specialized genetic algorithm is described, by discribing its differences from the classical one. In Section 5 the results of application of this algorithm to the control problems are presented. In Section 6 the analytical solutions of the test problems are presented, and the performance of this system is compared with that of the classical genetic algorithm and of GAMS. Section 7 provides some concluding remarks, along with

further planned extentions.

2 Genetic Algorithms

Genetic algorithms ([Davis, 1987], [De Jong, 1985], [Goldberg, 1985], [Holland, 1975]) 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.

For a given optimization problem, at each generation t of a genetic algorithm, it maintains a population of solutions $P(t) = \{x_1^t, \ldots, x_n^t\}$, where x_i^t is a feasible solution, t is the generation number, and n is arbitrarily chosen size of the population. This population undergoes simulated evolution; at each generation relatively good solutions reproduce, the relatively bad solutions die out, and are replaced by the offspring of the former ones. The initial population may be a set of random feasible solutions, which may also express some additional knowledge about the nature of the problem. To distinguish between different solutions, the function being optimized, $f(x_i^t)$, is used — it plays the role of the environment (see Figure 1). The evolutionary process is based on two primary operators: *mutation* and *crossover*.

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.

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 chromosomes are represented by five-dimensional vectors, say $x_1 = \langle a_1, b_1, c_1, d_1, e_1 \rangle$ and $x_2 = \langle a_2, b_2, c_2, d_2, e_2 \rangle$, then crossing the vectors after the second element would produce the offspring $\langle a_1, b_1, c_2, d_2, e_2 \rangle$ and $\langle a_2, b_2, c_1, d_1, e_1 \rangle$. Such an action allows for random, yet structured exchange of information contained in different solutions.

The *mutation* operator arbitrarily alters one or more components of a selected structure, which increases the variability of the population. Each bit position of each vector in the new population undergoes a random change with equal probability.

An additional *inverse* operator is also usually used. Such an operator inverses the ordering of a random part of a chromosome, and its applicability is based on itroducing "fairness" to different kinds of solutions in the presence of *mutation* and *crossover*.

The theoretical basis of a genetic algorithm state that, in a given population, chromosomes (solutions) better suited to the environment (evaluation) will have exponentially greater chance of survival, and, therefore, better chance of producing offspring [Holland, 1976]. Moreover, this genetic search method is far from being a pure *hill–climbing*, for at any time it provides for both exploitation of the best solutions, and exploration of the search space.

A genetic algorithm for a particular problem must have the following 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 implementation we have used a specialized genetic algorithm designed to work on numerical problems. All modification to the above classical version are described in Section 4 of this paper.

3 Two Optimal Control Problems

Two simple discrete-time optimal control models have been chosen as test problems for the specialized genetic algorithm.

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

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

subject to

$$x_{k+1} = a \cdot x_k + b \cdot u_k,\ k = 0, 1, \ldots, N-1, \tag{2}$$

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 second test problem is

$$\max \sum_{k=0}^{N-1} \sqrt{u_k} \tag{3}$$

subject to

$$x_{k+1} = a \cdot x_k - u_k \tag{4}$$

and

$$x_0 = x_N \tag{5}$$

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.

Let us recall that the value for the optimal performance of (1) subject to (2) is

$$J^* = K_0 x_0^2 \tag{6}$$

where K_k is the solution of the Riccati equation

$$\begin{aligned} &K_k = s + r a^2 K_{k+1}/(r + b^2 K_{k+1}), \text{ and} \\ &K_N = q. \end{aligned} \tag{7}$$

The optimal value J^* of (3) subject to (4) and (5) is (after elementary calculations):

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

The optimal control and state trajectory can obviously be determined analytically as well.

The value $N = 45$ is chosen as the largest horizon for which a comparative numerical solution from GAMS was still achievable.

Problem (3) subject (4) and (5) will be solved for the following values of N: $N = 2$, $N = 4$, $N = 10$, $N = 20$, and $N = 45$.

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

Table 1. Ten test cases for the problem (1) subject to (2).

4 The Specialized Genetic Algorithm

The specialized version of a genetic algorithm we have used is designed especially to work on numerical parameter optimization problems with real valued domains. The main objective behind this implementation was to move the genetic algorithm closer to the problem space. Such a move forced, but also allowed, the operators to be more problem specific — by utilizing some real space specific characteristics. For example, such representation has the property that two points, close to each other in the representation space, must also be close in the problem space, and vice versa. On the contrary, this is not true in the binary approach, where the distance in a representation is normally defined by the number of different bit positions. One of the advantages of this algorithm is that it is well suited to perform local fine tuning (due to dynamic operators), a property normally not attributed to such algorithms. Another of its advantages is that it is less dependent on the problem, or the function itself (due to function mapping, as described shortly).

4.1 The Representation

We use floating point, rather than binary, representation. Such a representation does not enjoy the theoretical foundations given for the binary case ([Holland, 1975]), but it was empirically shown to give respectable results for real valued parameter optimization ([De Jong, 1990]). The reason for the applicability of such floating point representation is that it matches the problem objectives (real values) closer than the binary one, which requires an extra mapping:

Range of representation $\rightarrow$ *Domain*

Because of this mismatch, the binary representation is less applicable to implementing our main objective — moving the genetic algorithm closer to the problem space, all in order to implement problem specific operators. As mentioned earlier, our representation allows to operate more directly in the problem space, allowing, for example, the implementation of the dynamic operators, introduced shortly.

The exact representation is as follow: for a problem of n variables, each chromosome, representing a permissible solution, is represented as a vector of n floating point numbers $x_i^t = \langle v_1, \ldots, v_n \rangle$. The precision of such a representation is fixed for a given machine, and based on the precision of the floating point (or double, if needed) type.

4.2 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 three classes: mutation, crossover, and inverse. In addition, we divide the operators into static and dynamic, where static ones do not change over the life of the population, while dynamic ones are functions of t.

Mutation group:

- **static random mutation**, defined similarly to that of the classical version: if $x_i^t = \langle v_1, \ldots, v_n \rangle$ 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 $\langle v_1, \ldots, v'_k, \ldots, v_n \rangle$, with $k \in 1..n$, and v'_k a random value from the domain of the corresponding parameter $domain_k$.

- **dynamic mutation** is one of the operators responsible for the fine tuning capabilities of the system. It is defined as follow: if $x_i^t = \langle v_1, \ldots, v_n \rangle$ 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 $\langle v_1, \ldots, v'_k, \ldots, v_n \rangle$, with $k \in 1..n$,

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

  and UB and LB defined from: $domain_k = (LB, UB)$. The dynamic function $\triangle(t, x)$ returns a random value in the range $\langle 0, x \rangle$, with a non–uniform probability: the distribution is uniform for $t = 0$, and denses close to 0 as t increases. This property causes the space to be searched very locally at later stages of the population life.

Crossover group:

- **crossover**, defined in the usual way, but with the only permissible split points between v's, for a given chromosome x.

- **multiple crossover**, defined as variation of the crossover with multiple split points. The number and position of such split points are randomly controlled by probability p_m^{mc}, which makes this number proportional to the length of the chromosome.

- **arithmetical crossover**, defined as follow: if $x_i^t = \langle v_1, \ldots, v_n \rangle$ and $x_j^t = \langle w_1, \ldots, w_n \rangle$ are to be crosses, the resulting offspring are $x_i^{t+1} = \langle v_1, \ldots, v'_k, \ldots, v_n \rangle$ and $x_j^{t+1} =$

$\langle w_1, \ldots, v'_k, \ldots, w_n \rangle$, for some $k \in (1, n)$, and $v'_k = a * v_k + (1 - a) * w_k$ for some $a \in (0, 1)$. In particular, for $a = 0.5$, the resulting element represents an average of the corresponding two parent elements.

- **multiple arithmetical crossover**, defined analogously to multiple crossover, as a variation of the above, and controlled by the probability p_m^{mar}.
- **whole arithmetical crossover**, a special case of the above, with p_m^{mar} fixed at one.

The **inverse group** consists of only one operator, defined as follow: if $x_i^t = \langle v_1, \ldots, v_n \rangle$ is selected for invesrion, then $x_i^{t+1} = \langle v_1, \ldots, v_{k-1}, v_j, v_{j-1}, \ldots, v_{j-k}, v_{j+1}, \ldots, v_n \rangle$, for some $j \in (1, n)$ and some $k \leq j$.

4.3 Other Enhancements

There are a number of other enhancements implemented in the algorithm, which deal with the convergence problem. A more detailed description is presented in [Michalewicz et al., 1990]. Here, we will only briefly describe some of them, along with the reason and need for their application.

There are two, somehow related, problems with the numerical parameter optimization by genetic algorithms: premature convergence and variance in the average function evaluation.

The premature convergence problem is strongly related to the existence of local optima, and depends on the distribution of evaluation values in some population $P(t)$. For example, assume that $x_i^t \in P(t)$ is close to some local optimum, and $f(x_i^t)$ is much greater than the average evaluation $\overline{f}(x^t)$. Also, assume that there is no x_j^t close to the global maximum sought. This might be the case for many multi–optimum, non–smooth functions. In such a case, there is a fast convergence toward that local optimum. Because of that, the population $P(t+1)$ becomes over–saturated with elements close to that solution, decreasing the chance of more global exploration, needed to search for other optimal solutions. While such a behavior is permissible at the later evolutionary stages, and even desired at the very final stages, it is quite disturbing at the early ones ($t \cong 0$). Our approach diminishes this problem by decreasing the speed of convergence during the early stages of population existence. This is achieved by changing the way a selection probability, normally proportional to $f(x_i^t)/\overline{f}(x^t)$, is computed. In here, it becomes proportional to $(t/T)^{1/q} \cdot f(x_i^t)/\overline{f}(x^t)$, where q is a small positive integer, and T is the anticipated number of generations. Then, the initial selection ($t = 0$) is random, and it quickly (depending on the magnitude of q) becomes closer related to the traditional, environment (evaluation) guided, approach.

The other problem, also reflecting on the convergence, is related to shifts in the average population fitness. Consider two functions: $f_1(x)$, and $f_2(x) = f_1(x) + const$. Since they are both basically the same functions, one would expect that both can be optimized with similar degree of difficulty. However, if $const \gg \overline{f_1}(x)$, then the function $f_2(x)$ will

suffer from much slower convergence than the function $f_1(x)$. In fact, in the extreme case, the second function will be optimized using a totally random search. Some of the previous approaches to this problem used rank instead of actual values $f(x_i^k)$ to guide the selective process. Such an approach suffers from many drawbacks. First, it puts the responsibility on the user to decide on the best selection mechanism. Second, it ignores the information it holds about the relative evaluations of different chromosomes. Third, it treats all cases uniformly, regardless of the magnitude of the problem. In our approach, we used a dynamic measure of evaluation variance, defined as

$$s_t = \sqrt{\sum_{i=1}^{n} \left(f^2(x_i^t) - \overline{f}^2(x^t) \right)} \Big/ \sum_{i=1}^{n} f(x_i^t)$$

Moreover, we experimentally determined that $s^*_{t=0} = 0.10$ gives the best trade-off between convergence speed and space exploration. Now, given a new function to be optimized, and $s_{t=0}$ too far away from $s^*_{t=0}$, the system finds parameters a, b, such that mapping $F = a \cdot f + b$ gives the desired variance. Thereafter, the system switches to optimizing function F. In addition, experiments suggest that, in general, it is not necessary to update the mapping parameters dynamically, as the initial population $P(t = 0)$ represents a good sampling of the solution space.

5 Experiments and Results

In this section we present the results obtained from the specialized genetic algorithm for optimal control problems.

For each case, we have repeated 3 separate runs of 40,000 generations, and the best of those are reported in Table 2, along with intermediate results at some generation intervals. For example, the values in column "10,000" indicate the partial results 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 generation, due to the nature of some genetic operators. For all cases, the population size was fixed at 100. In the next section we compare these results with the exact solutions, those obtained from the computational package GAMS, and from the classical genetic algorithm.

Problem (3)–(5) differs from the first one due to the constraint $x_0 = x_n$. To produce a feasible initial population, we have generated a random sequence of $u_0, \ldots, u_{N-2}$, and then set $u_{N-1} = a \cdot x_{N-1} - x_N$. For negative u_{N-1}, we discarded this particular sequence and generated a new one: this happened in less than 10% of cases.

This constraint also introduced some difficulty during the reproduction — an offspring (after some genetic operations) need not constitute a feasible solution. If this was the case, we replaced the last component of the offspring vector u using the formula: $u_{N-1} = a \cdot x_{N-1} - x_N$. However, if u_{N-1} turned out to be negative, the new element was discarded (which, again, did not happen in more than 10% of the cases). As mentioned before, we

	Generations							Factor
Case	1	100	1,000	10,000	20,000	30,000	40,000	
I	17807.4	3.27985	1.74689	1.61866	1.61825	1.61804	1.61803	10^4
II	13670.4	5.33177	1.45968	1.11349	1.09205	1.09165	1.09163	10^5
III	17023.8	2.87485	1.07974	1.00968	1.00126	1.00104	1.00103	10^7
IV	15077.3	8.64310	3.75530	3.71846	3.70812	3.70165	3.70160	10^4
V	5956.43	12.2559	2.89769	2.87727	2.87646	2.87570	2.87569	10^5
VI	16657.7	5.07047	2.05314	1.61869	1.61830	1.61806	1.61806	10^4
VII	2680666	19.2684	7.02566	1.63464	1.62412	1.61888	1.61882	10^4
VIII	116.982	67.1758	1.92764	1.00009	1.00005	1.00005	1.00005	10^4
IX	7.18263	4.42849	4.37093	4.31504	4.31024	4.31004	4.31004	10^5
X	9870352	138132	16096.0	1.38244	1.00041	1.00010	1.00010	10^4

Table 2. Specialized Genetic Algorithm for problem (1)–(2).

are currently working on a more general algorithm, able to deal with such constraints automatically.

Table 3 summarizes the results. Problem (3)–(5) was solved for the following values of N: $N = 2$, $N = 4$, $N = 10$, $N = 20$, and $N = 45$. The population size was fixed at 100. We also present the intermediate values after the first, 100th, 1000th, 10,000th, 20,000th and 30,000th generations. It appears that, in this case, 10,000 generations is quite sufficient: the improvement in the next 30,000 generations is insignificant. This result generates the conclusion that this algorithm performs quite reasonable space exploration during the early stages of the evolutionary simulation, and then shifts the efforts toward local fine exploitation. As the results of the next section will show, this fine tuning is much better, and more efficient, than that of the classical genetic algorithm.

	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

Table 3. Specialized Genetic Algorithm for problem (3)–(5).

6 Specialized Genetic Algorithms vs. Other Methods

In this section we compare the above results with the exact solutions as well as with those obtained from the computational package GAMS and from the classical genetic algorithm. Exact solutions of the test problems for the values of the parameters specified in Table 1 have been obtained using formulas (6) and (7). The comparison may be regarded as not totally fair for the genetic algorithms, 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 algorithms proved to be competitive, or close to, there would be an indication that they should behave satisfactorily in general. Tables 4 and 5 summarize the results, where columns D refer to the percentage of the relative error.

	Exact solution	Specialized GA		GAMS		Classical GA	
Case	value	value	D	value	D	value	D
I	16180.3399	16180.3939	0.000%	16180.3399	0.000%	16318.3233	0.853%
II	109160.7978	109163.0278	0.002%	109160.7978	0.000%	110235.2444	0.989%
III	10009990.0200	10010391.3989	0.004%	10009990.0200	0.000%	10100003.4515	0.899%
IV	37015.6212	37016.0806	0.001%	37015.6212	0.000%	37152.5652	0.397%
V	287569.3725	287569.7389	0.000%	287569.3725	0.000%	288214.4587	0.223%
VI	16180.3399	16180.6166	0.002%	16180.3399	0.000%	16322.2135	0.875%
VII	16180.3399	16188.2394	0.048%	16180.3399	0.000%	16353.8502	1.061%
VIII	10000.5000	10000.5000	0.000%	10000.5000	0.000%	10078.5689	0.783%
IX	431004.0987	431004.4092	0.000%	431004.0987	0.000%	432997.9771	0.464%
X	10000.9999	10001.0045	0.000%	10000.9999	0.000%	10001.2695	0.001%

Table 4. Comparison of solutions for the linear-quadratic model.

As seen above (and as expected), the GAMS's performance for the linear–quadratic problem is perfect. The specialized genetic algorithm is only slightly worse: the relative error is always less than 0.05%, and in most cases less than 0.001%. On the other hand, the relative error for the classical genetic algorithm is slightly bigger, although relatively competitive, as well. This results supports the claim that our algorithm performs at least the same global search as the classical one, while quite significantly improving the local tuning characteristics.

The differences between algorithms are even stronger for the second test problem, and the individual systems' qualities change. 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 in Table 5, and for $N > 4$ the system failed to find any solution. It appears that GAMS is sensitive to non-convexity 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"[1], 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.

On the other hand, both genetic algorithms performed about as well (in fact, even slightly better) than for the first test problem. Therefore, they are, as a class of algorithms, much less dependent on the characteristics of the function being optimized. It can also be seen that our specialized version outperformed the classical one again by a small margin (relatively small, but quite respectable under the assumption that high precision solutions are being sought). These results support those of the first test case in showing that this new specialized implementation performs much more respectable local space exploitation.

In addition, it is worthwhile to point out that our new version outperforms the old one by at least one order of magnitude on the time scale, for comparable implementations.

N	Solution	GAMS		GAMS+		Specialized GA		Classical GA	
		value	D	value	D	value	D	value	D
2	6.331738	4.3693	30.99%	6.3316	0.00%	6.3317	0.000%	6.3317	0.00%
4	12.721038	5.9050	53.58%	12.7210	0.00%	12.7210	0.000%	12.7131	0.08%
8	25.905710	*		18.8604	27.20%	25.9057	0.000%	25.8502	0.22%
10	32.820943	*		22.9416	30.10%	32.8209	0.000%	32.7045	0.38%
20	73.237681	*		*		73.2376	0.000%	73.0243	0.29%
45	279.275275	*		*		279.2714	0.001%	278.2168	0.38%

Table 5. Comparison of solutions for the linear-quadratic model. The symbol "*" means that the GAMS failed to find any reasonable value and gave a warning: "The final point is not close to an optimum".

7 Conclusions and Future Work

In this paper we have presented a specialized genetic algorithm for numerical optimization. We applied such an algorithm to two rather simple classes of optimal control problems. The results, as compared with those obtained from a search-based computational package (GAMS), and from a classical genetic algorithm, show that, as a generic tool, it outperforms both competitors. First of all, it is applicable to much broader class of such problems than the commercial GAMS. Secondly, it obtains much higher precision solutions than the classical genetic approach, and with respectably lower time complexity. Therefore, it should be a very powerfull tool in hands of a person not familiar with the problem being optomized. Moreover, it can also be seen as an strong alternative in hands of a knowledgable engineer.

Currently, work is under way to extend the applicability of this algorithm to much broader class of parameter optimization problems: those with linear constraints ([Michalewicz

[1]This is "unfair" from the point of view of the genetic algorithm which works without such help.

et al., 1990]). Such contraints, given to the system in a user–friendly manner, would not increase the complexity of the genetic algorithm, as it was the case for previous approaches. Instead, they would decrease the search space, when coupled with powerful, closed–space operators. Another extension that is being implemented involves combining both real and two kinds of discrete valued domains: linearly ordered integer subrange and nominal (unordered) set of integers. The real (or continuous) variables are being represented by the floating point type, the integer ranges by the integer type, and the nominal sets by a binary coding. Such combinations would be desired for some parameter optimization problems, where different parameters are of different type, *eg.* boolean, integers, and real.

Present experiments indicate that, after incorporating all the extentions, a package implementing this algorithm should be the most powerfull commercially available generic tool for high precision general parameter optimization problems.

References

[**Bellman, 1957**] Bellman, R., *Dynamic Programming*, Princeton University Press, Princeton, N.J., 1957.

[**Bertsekas, 1987**] Bertsekas, D. P., *Dynamic Programming. Deterministic and Stochastic Models*, Prentice Hall, Englewood Cliffs, N.J., 1987.

[**Brooke et al, 1988**] Brooke, A., Kendrick, D., and Meeraus, A., *GAMS: A User's Guide*, The Scientific Press, 1988.

[**Davis, 1987**] Davis, L., (editor), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.

[**De Jong, 1985**] De Jong, K.A., *Genetic Algorithms: A 10 Year Perspective*, Proceedings of an International Conference on genetic Algorithms and Their Applications, Pittsburgh, pp.169–177.

[**De Jong, 1990**] De Jong, K.A., private communication.

[**Goldberg, 1989**] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, 1989.

[**Grefenstette, 1986**] Grefenstette, J.J. *Optimization of Control Parameters for Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-16, No.1, January/February 1986, pp.122–128.

[**Holland, 1975**] Holland, J., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

[**Holland, 1986**] Holland, J., *Escaping Brittleness*, in *Machine Learning II*, ed. R. Michalski, J. Carbonell, T. Mitchel, Morgan Kaufmann Publ., Los Altos, CA.

[**Holland et al., 1986**] Holland, J.H., Holyoak, K.J., Nisbett, R.E., and Thagard, P.R., *Induction*, The MIT Press, 1986,

[**Michalewicz, 1989**] Michalewicz, Z., *EVA Programming Environment*, submitted for publication,

[**Michalewicz et al., 1989**] Michalewicz, Z., Vignaux, G.A., Groves, L. *Genetic Algorithms for Approximation and Optimization Problems*, Proceedings of the 11th New Zealand Computer Conference, Wellington, August 16–18, 1989, pp.211-223.

[**Michalewicz et al., 1990**] Michalewicz, Z., Janikow, C.Z., Vignaux, G.A., *GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints*, submitted to Communications of the ACM.

[**Vignaux & Michalewicz, 1989a**] Vignaux, G.A., Michalewicz, Z., *Genetic Algorithms for the Transportation Problem*, Proc. 4th International Symposium on Methodologies for Intelligent Systems, Charlotte, October 12–14, 1989, pp.252–259.

[**Vignaux & Michalewicz, 1989b**] Vignaux, G.A., Michalewicz, Z., *A Genetic Algorithm for the Linear Transportation Problem*, submitted to IEEE Transactions on Systems, Man, and Cybernetics.

[**Vignaux & Michalewicz, 1989c**] Vignaux, G.A., Michalewicz, Z., *A Genetic Algorithm for the Nonlinear Transportation Problem*, in preparation.