The MATLAB Genetic Algorithm Toolbox

A. J. Chipperfield and P. J. Fleming[1]

1. Introduction

Genetic algorithms (GAs) are stochastic global search and optimization methods that mimic the metaphor of natural biological evolution [1]. GAs operate on a population of potential solutions applying the principle of survival of the fittest to produce successively better approximations to a solution. At each generation of a GA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and reproducing them using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals from which they were created, just as in natural adaptation.

GAs have been shown to be an effective strategy in the off-line design of control systems by a number of practitioners. For example, Krishnakumar and Goldberg [2] and Bramlette and Cusin [3] have demonstrated how genetic optimization methods can be used to derive superior controller structures in aerospace applications in less time (in terms of function evaluations) than traditional methods such as LQR and Powell's gain set design. Porter and Mohamed [4] have presented schemes for the genetic design of multivariable flight control systems using eigenstructure assignment, whilst others have demonstrated how GAs can be used in the selection of controller structures [5].

2. The MATLAB GA Toolbox

Whilst there exist many good public-domain genetic algorithm packages, such as GENESYS [6] and GENITOR [7], none of these provide an environment that is immediately compatible with existing tools in the control domain. The MATLAB Genetic Algorithm Toolbox [8] aims to make GAs accessible to the control engineer within the framework of an existing CACSD package. This allows the retention of existing modelling and simulation tools for building objective functions and allows the user to make direct comparisons between genetic methods and traditional procedures.

2.1 Data Structures

MATLAB essentially supports only one data type, a rectangular matrix of real or complex numeric elements. The main data structures in the GA Toolbox are chromosomes, phenotypes, objective function values and fitness values. The chromosome structure stores an entire population in a single matrix of size $N_{ind} \times L_{ind}$, where N_{ind} is the number of individuals and L_{ind} is the length of the chromosome structure. Phenotypes are stored in a matrix of dimensions $N_{ind} \times N_{var}$ where N_{var} is the number of decision variables. An $N_{ind} \times N_{obj}$ matrix stores the objective function values, where N_{obj} is the number of objectives. Finally, the fitness values are stored in a vector of length N_{ind}. In all of these data structures, each row corresponds to a particular individual.

2.2 Toolbox Structure

The GA Toolbox uses MATLAB matrix functions to build a set of versatile routines for implementing a wide range of genetic algorithm methods. In this section we outline the major procedures of the GA Toolbox.

Population representation and initialisation: `crtbase`, `crtbp`, `crtrp`

The GA Toolbox supports binary, integer and floating-point chromosome representations. Binary and integer populations may be initialised using the Toolbox function to create binary populations, `crtbp`. An additional function, `crtbase`, is provided that builds a vector describing the integer representation used. Real-valued populations may be initialised using `crtrp`. Conversion between binary and real-values is provided by the routine `bs2rv` that also supports the use of Gray codes and logarithmic scaling.

Fitness assignment: `ranking`, `scaling`

The fitness function transforms the raw objective function values into non-negative figures of merit for each individual. The Toolbox supports the offsetting and scaling method of Goldberg [9] and the linear-ranking algorithm

---

1. Department of Automatic Control and Systems Engineering, University of Sheffield, PO Box 600, Mappin Street, Sheffield, England. S1 4DU
From IEE Colloquium on Applied Control Techniques Using MATLAB, Digest No. 1995/014, 26/01/95

of Baker [10]. In addition, non-linear ranking is also supported in the routine `ranking`.

Selection functions: `reins`, `rws`, `select`, `sus`

These functions select a given number of individuals from the current population, according to their fitness, and return a column vector to their indices. Currently available routines are roulette wheel selection [9], `rws`, and stochastic universal sampling [11], `sus`. A high-level entry function, `select`, is also provided as a convenient interface to the selection routines, particularly where multiple populations are used. In cases where a generation gap is required, i.e. where the entire population is not reproduced in each generation, `reins` can be used to effect uniform random or fitness-based re-insertion [9].

Crossover operators: `recdis`, `recint`, `reclin`, `recmut`, `recombin`, `xovdp`, `xovdprs`, `xovmp`, `xovsh`, `xovshrs`, `xovsp`, `xovsprs`

The crossover routines recombine pairs of individuals with given probability to produce offspring. Single-point, double-point [12] and shuffle crossover [13] are implemented in the routines `xovsp`, `xovdp` and `xovsh` respectively. Reduced surrogate [13] crossover is supported with both single-, `xovsprs`, and double-point, `xovdprs`, crossover and with shuffle, `xovshrs`. A general multi-point crossover routine, `xovmp`, that supports uniform crossover [14] is also provided. To support real-valued chromosome representations, discrete, intermediate and line recombination are supplied in the routines, `recdis`, `recint` and `reclin` respectively [15]. The routine `recmut` performs line recombination with mutation features [15]. A high-level entry function to all the crossover operators supporting multiple subpopulations is provided by the function `recombin`.

Mutation operators: `mut`, `mutate`, `mutbga`

Binary and integer mutation are performed by the routine `mut`. Real-value mutation is available using the breeder GA mutation function [15], `mutbga`. Again, a high-level entry function, `mutate`, to the mutation operators is provided.

Multiple subpopulation support: `migrate`

The GA Toolbox provides support for multiple subpopulations through the use of high-level genetic operator functions and a function for exchanging individuals amongst subpopulations, `migrate`. A single population is divided into a number of subpopulations by modifying the data structures used by the Toolbox routines such that subpopulations are stored in contiguous blocks within each data element. The high-level routines, such as `select` and `reins`, operate independently on each subpopulation contained in a data structure allowing each subpopulation to evolve in isolation from the others. Based on the *Island* or *Migration* model [16], `migrate` allows individuals to be transferred between subpopulations. Uni- and bi-directional ring topologies as well as a fully interconnected network are selectable via option settings as well as fitness-based and uniform selection and re-insertion strategies.

2.3 A Simple GA in MATLAB

Fig.1 shows the MATLAB code for a Simple GA. The first few lines of the code set the parameters that the GA uses, such as the number and length of the chromosomes, the crossover and mutation rates, the number of generations and, in this case, the binary representation scheme. Next, an initial uniformly distributed random binary population, `Chrom`, is created using the GA Toolbox function `crtbp`. The objective function, `objfun`, is then evaluated to produce the vector of objective values, `ObjV`. Note that as we do not need the phenotypic representation inside the GA, the binary strings are converted to real values within the objective function call.

The initialisation complete, the GA now enters the generational loop. First, a fitness vector, `FitnV`, is determined using the ranking scheme of Baker [11]. Visualisation and preference articulation can be incorporated into the generational loop by the addition of extra functions. In this example, the routine `plotgraphics` displays the performance of the current best controller allowing the user to asses the state of the search. Individuals are then selected from the population using the stochastic universal sampling algorithm, `sus`, with a generation gap, `GGAP = 0.9`. The 36 (`GGAP` $\times$ `NIND`) selected individuals are then recombined using single-point crossover, `xovsp`, applied with probability `XOV = 0.7`. Binary mutation, `mut`, is then applied to the offspring with probability `MUTR = 0.0175`, and the objective function values for the new individuals, `ObjVSel`, calculated. Finally, the new individuals are re-inserted in the population, using the Toolbox function `reins`, and the generation counter, `gen`, incremented.

The GA terminates after `MAXGEN` iterations around the generational loop. The current population, its phenotypic representation and associated cost function values remain in the users workspace and may be analysed directly using MATLAB commands.

```
LIND = 15;                                % Length of individual vars.
NVAR = 2;                                 % No. of decision variables
NIND = 40;                                % No. of individuals
GGAP = 0.9;                               % Generation gap
XOV = 0.7;                                % Crossover rate
MUTR = 0.0175;                            % Mutation rate
MAXGEN = 30;                              % No. of generations
                                          % Binary representation scheme
FieldD = [LIND LIND; 1 1; 1000 1000; 1 1; 0 0; 0 0; 0 0];
% Initialise population
Chrom = crtbp(Nind, Lind*NVAR);           % Create binary population
ObjV = objfun(bs2rv(Chrom, FieldD));      % Evaluate objective fn.
Gen = 0;                                  % Counter
% Begin generational loop
while Gen < MAXGEN
    % Assign fitness values to entire population
    FitnV = ranking(ObjV);
    % Visualisation
    plotgraphics
    % Select individuals for breeding
    SelCh = select('sus', Chrom, FitnV, GGAP);
    % Recombine individuals (crossover)
    SelCh = recombin('xovsp', SelCh, XOV);
    % Apply mutation
    SelCh = mut(SelCh, MUTR);
    % Evaluate offspring, call objective function
    ObjVSel = objfun(bs2rv(SelCh, FieldD));
    % Reinsert offspring into population
    [Chrom ObjV]=reins(Chrom, SelCh, 1, 1, ObjV, ObjVSel);
    % Increment counter
    Gen = Gen+1;
end
% Convert Chrom to real-values
Phen = bs2rv(Chrom, FieldD);
```

Figure 1: MATLAB Code for a Simple GA

3. Applications and Further Developments

The GA Toolbox has been beta-tested at approximately 30 sites world-wide in a wide range of application areas including:

- parametric optimization
- controller structure selection
- mixed-mode modelling
- real-time and adaptive control
- fault identification
- multiobjective optimization
- nonlinear system identification
- neural network design
- parallel genetic algorithms
- antenna design

In future releases we plan to incorporate support for multiobjective optimization. Multi Objective GAs (MOGAs) evolve a population of solution estimates thereby conferring an immediate benefit over conventional MO methods. Fonseca and Fleming [17] have demonstrated how, using rank-based selection and niching techniques, it is feasible to generate populations of non-dominated solution estimates without combining objectives in some way. This is advantageous because the combination of non-commensurate objectives requires precise understanding of the interplay between those objectives if the optimization is to be meaningful. The use of rank-based fitness assignment permits different non-dominated individuals to be sampled at the same rate thereby according the same preference to all Pareto-optimal solutions. Because MOGAs are susceptible to unstable converged populations, due to the potential

for very different genotypes to result in non-dominated individuals, a particular problem is the production of *lethals* when fit members of the population are mated. The search then becomes inefficient and the GA is likely to converge to some suboptimal solution. In general, a combination of mating restriction, niche formation and redundant coding may be appropriate.

4. Concluding Remarks

Together with MATLAB and SIMULINK, the GA Toolbox described in this paper presents a familiar and unified environment for the control engineer to experiment with and apply GAs to tasks in control systems engineering. Whilst the GA Toolbox was developed with the emphasis on control engineering applications, it should prove equally as useful in the general field of GAs, particularly given the range of domain-specific toolboxes available for the MATLAB package.

5. Acknowledgements

The authors gratefully acknowledge the support of this research by a UK SERC grant on “Genetic Algorithms in Control Systems Engineering” (GR/J17920).

6. References

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

[2] K. Krishnakumar and D. E. Goldberg, “Control System Optimization Using Genetic Algorithms”, *Journal of Guidance, Control and Dynamics*, Vol. 15, No. 3, pp. 735-740, 1992.

[3] M. F. Bramlette and R. Cusin , “A Comparative Evaluation of Search Methods Applied to Parametric Design of Aircraft”, *Proc. ICGA 3*, pp213-218, 1989.

[4] B. Porter and S. S. Mohamed, “Genetic Design of Multivariable Flight-Control Systems Using Eigenstructure Assignment”, *Proc. IEEE Conf. Aerospace Control Systems*, 1993.

[5] A. Varsek, T. Urbacic and B. Filipic, “Genetic Algorithms in Controller Design and Tuning”, *IEEE Trans. Sys. Man and Cyber*., Vol. 23, No. 5, pp1330-1339, 1993.

[6] J. J. Grefenstette, “A User’s Guide to GENESIS Version 5.0”, Technical Report, Navy Centre for Applied Research in Artificial Intelligence, Washington D.C., USA, 1990.

[7] D. Whitley, “The GENITOR algorithm and selection pressure: why rank-based allocations of reproductive trials is best,” in *Proc. ICGA 3*, pp. 116-121, 1989.

[8] A. J. Chipperfield, P. J. Fleming and C. M. Fonseca, “Genetic Algorithm Tools for Control Systems Engineering”, *Proc. Adaptive Computing in Engineering Design and Control*, Plymouth Engineering Design Centre, 21-22 September, pp. 128-133, 1994.

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

[10] J. E. Baker J, “Adaptive Selection Methods for Genetic Algorithms”, *Proc. ICGA 1*, pp. 101-111, 1985.

[11] J. E. Baker, “Reducing bias and inefficiency in the selection algorithm”, *Proc. ICGA 2*, pp. 14-21, 1987.

[12] L. Booker, “Improving search in genetic algorithms,” In *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed.), pp 61-73, Morgan Kaufmann Publishers, 1987.

[13] R. A. Caruana, L. A. Eshelman and J. D. Schaffer, “Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover”, In *Eleventh Int. Joint Conf. on AI*, Sridharan N. S. (Ed.), Vol. 1, pp 750-755, Morgan Kaufmann, 1989.

[14] G. Syswerda, “Uniform crossover in genetic algorithms”, *Proc. ICGA 3*, pp. 2-9, 1989.

[15] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm”, *Evolutionary Computation*, Vol. 1, No. 1, pp. 25-49, 1993.

[16] C. B. Petty, M. R. Leuze. and J. J. Grefenstette, “A Parallel Genetic Algorithm”, *Proc. ICGA 2*, pp. 155-161, 1987.

[17] C. M. Fonseca and P. J. Fleming, “Genetic Algorithms for Multiple Objective Optimization: Formulation, Discussion and Generalization”, *Proc. ICGA 5*, pp. 416-423, 1993.