

**Extended Compact Genetic Algorithm in Matlab**

**Kumara Sastry, and Albert Orriols-Puig**  
**IlliGAL Report No. 2007009**  
**March, 2007**

Illinois Genetic Algorithms Laboratory  
University of Illinois at Urbana-Champaign  
117 Transportation Building  
104 S. Mathews Avenue Urbana, IL 61801  
Office: (217) 333-2346  
Fax: (217) 244-5705

# Extended Compact Genetic Algorithm in Matlab

Kumara Sastry<sup>1</sup>, Albert Orriols-Puig<sup>2</sup>

<sup>1</sup>Illinois Genetic Algorithms Laboratory (IlliGAL)  
Department of Industrial and Enterprise Systems Engineering  
University of Illinois at Urbana-Champaign, Urbana IL 61801  
ksastry@uiuc.edu

<sup>2</sup>Group of Research in Intelligent Systems  
Computer Engineering Department  
Enginyeria i Arquitectura La Salle — Ramon Llull University  
Quatre Camins, 2. 08022, Barcelona, Catalonia, Spain.  
aorriols@salle.url.edu

March 22, 2007

## Abstract

This report provides documentation for matlab<sup>®</sup> implementation of the extended compact genetic algorithm (eCGA). The implementation works for integer decision variables where each variable can be of differing cardinality.

## 1 Introduction

In this report we briefly describe how to download and run the matlab<sup>®</sup> implementation of the extended compact genetic algorithm (eCGA) (Harik, 1999; Harik, Lobo, & Sastry, 2006). However, in contrast to original eCGA, in the current implementation each decision variable can have a different user-specified cardinality. That is, we are not restricted to optimizing binary variables.

This report also explains how to modify the objective function that comes with the distribution of the code. The source is written in matlab and basic knowledge of matlab is sufficient to modify the fitness function and run eCGA on user-defined problems.

## 2 How to download the code?

The code is available from `ftp://ftp-illigal.ge.uiuc.edu/pub/src/ECGA/eCGAmatlab.zip`. After downloading it, uncompress the file by typing

```
unzip eCGAmatlab.zip
```

At this point you should have in your directory the following files:

<code>buildModel.m</code>	<code>createCombinations.m</code>	<code>displayResults.m</code>
<code>eCGA.m</code>	<code>elitistReplacement.m</code>	<code>evaluateIndividuals.m</code>
<code>evaluateModel.m</code>	<code>restrictedTournamentReplacement.m</code>	<code>sampleModel.m</code>
<code>tournamentSelection.m</code>	<code>truncationSelection.m</code>	<code>userDefinedParameters.m</code>

### 3 How to run the code?

We assume you have matlab properly installed on your computer. We have tested the code under Linux and Windows operating systems. The main function is `eCGA.m` which reads user defined parameter values such as the number of variables, population size from `userDefinedParameters.m`. Sample parameter values are provided as an example. The file is straightforward to understand and is fully commented. Type `eCGA` at the matlab prompt to run `eCGA`.

At the end of every generation, the marginal product model built in that generation, the best solution found so far and the current best fitness value is displayed on the screen.

The objective function that comes with the distribution of the code is a concatenation of 6 copies of a 4-bit trap function. The trap function has fitness  $0.75(1 - u/3)$  if the string has  $u$  ones, except when the string is `1111`, in which case the fitness is 1. Thus, for each of the 6 blocks, the global optima is at `1111`, each with fitness 1, and the local (deceptive) optima is at `0000`, and has a fitness of 0.75. Therefore, for the overall problem, the optimal solution is the string with all ones and has a fitness of 6.

### 4 How to plug-in your own objective function?

The code for the objective function is in the file `evaluateIndividuals.m`. Along with `userDefinedParameters.m`, this is the only function that you need to rewrite in order to try your own problem. Note that you need to correctly input the number of variables and alphabet cardinalities in `userDefinedParameters.m` as dictated by your problem formulation. Almost no error checking is implemented in this version of `eCGA`, and we will consider it in future versions.

The function header is as follows:

```
fitness = evaluateIndividuals(ranges, population)
```

It takes as input argument arguments an array of alphabet cardinalities, and the population of individuals that need to be evaluated. The population is a matrix of dimension  $n \times \ell$ , where  $n$  is the population size and  $\ell$  is the number of variables. The function returns an array of real numbers: the fitness values of the individuals in the population.

### 5 About the matlab eCGA code

The implementation of the `eCGA` doesn't use advanced features of matlab. This means that you don't need to be a matlab expert in order to modify the code. This also means that the current implementation is not as fast as it could be made. Implementation efficiencies would be enhanced in the future versions of the code.

We give a brief description of each of the implemented functions.

`buildModel.m` builds the marginal product model using greedy search heuristic described in (Harik, Lobo, & Sastry, 2006).

`createCombinations.m` Given the current marginal product model (MPM), this function returns an array of MPMs created by all possible pairwise merges of subsets of the given MPM.

`displayResults.m` displays the marginal product model, current best solution and its fitness value at any given generation.

`eCGA.m` contains the main loop of the eCGA.

`elitistReplacement.m` performs Lambda+Mu elitist replacement by combining parent and offspring population and choosing the best n individuals.

`evaluateIndividuals.m` defines the fitness function and evaluates the fitness values of individuals in a given population. If you want to try the eCGA on your own problem, you should modify this function.

`evaluateModel.m` computes the MDL score of a given marginal product model. In the process this function also computes the subsolutions and the associated frequencies.

`restrictedTournamentReplacement.m` performs restricted tournament replacement (Harik, 1995; Pelikan, 2005).

`sampleModel.m` samples a new population based on the given MPM.

`tournamentSelection.m` performs tournament selection without replacement (Goldberg, Korb, & Deb, 1989; Sastry & Goldberg, 2001) and returns an index of selected individuals.

`truncationSelection.m` performs truncation selection (Mühlenbein & Schlierkamp-Voosen, 1993) and returns an index of selected individuals.

`userDefinedParameters.m` contains all the user-defined eCGA parameters

## 6 Disclaimer

This code is distributed for academic purposes only. It has no warranty implied or given, and the authors assume no liability for damage resulting from its use or misuse. If you have any comments or find any bugs, please send an email to [kumara@illigal.ge.uiuc.edu](mailto:kumara@illigal.ge.uiuc.edu).

## 7 Commercial use

For the commercial use of this code please contact Prof. David E. Goldberg at [deg@uiuc.edu](mailto:deg@uiuc.edu)

## Acknowledgments

This work was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, the National Science Foundation under grant ITR grant DMR-03-25939 at the Materials Computation Center. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

We thank Prof. David E. Goldberg for encouraging us to write this report.

## References

- Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5), 493–530. (Also IlliGAL Report No. 89003).
- Harik, G. (1999, January). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign.
- Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *Proceedings of the Sixth International Conference on Genetic Algorithms*, 24–31. (Also IlliGAL Report No. 94002).
- Harik, G. R., Lobo, F. G., & Sastry, K. (2006). Linkage learning via probabilistic modeling in the ECGA. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications* (Chapter 3, pp. 39–61). Berlin: Springer. (Also IlliGAL Report No. 99010).
- Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. continuous parameter optimization. *Evolutionary Computation*, 1(1), 25–49.
- Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithm*. Berlin: Springer Verlag.
- Sastry, K., & Goldberg, D. E. (2001). Modeling tournament selection with replacement using apparent added noise. *Intelligent Engineering Systems Through Artificial Neural Networks*, 11, 129–134. (Also IlliGAL Report No. 2001014).