A Blueprint for a Genetic Meta-Algorithm

Spyros Raptis and Spyros Tzafestas

Intelligent Robotics and Automation Laboratory
National Technical University of Athens
Zografou Campus, 15773, Athens, GREECE
Phone: +30-1-772 2489, Fax: +30-1-772 2490
E-mail: sraptis@robotics.ece.ntua.gr

ABSTRACT: The importance of adapting the parameters of genetic operators throughout the algorithm run, is a truism in GAs. It can be argued that it is equally important to adapt various other parameters of a population to achieve even higher efficiency. In the current work, we present a conceptual and functional framework for a family of self-adapting genetic algorithms, we call genetic meta-algorithms. We revisit previous work on the so-called meta-level architectures and reformulate it to a derive a two-level model where the meta component dynamically interferes with the progress of the nested genetic algorithm. This framework provides the means for efficiently adapting various important GA parameters during the algorithm run through the competition of nested populations and the domination of the "best" with respect to on-line and off-line performance criteria. The first results obtained were promising indicating that efficient self-adaptation of various genetic parameters can, indeed, be based on the proposed scheme.

I. INTRODUCTION

It is common place in GA research that efficient search requires varying the parameters of the operators as the mutation and crossover. Although their mechanics are rather simple, theoretical studies and results of wide scope concerning the choice of appropriate rates and adaptation schedules for these parameters are still missing. These choices still remain something of an art during the design of GAs.

The role of changing the behavior of the various operators involved in genetic search has been early recognized. It has been extensively used as a means for controlling the flavor of the population from generation to generation: to avoid premature convergence and the early domination of a highly fit individual, to maintain competition among individuals, to maintain diversity within the population, to capture the effects of tight linkage for some specific problems etc.

Numerous ad-hoc (self-)adaptation methodologies have been presented to adjust the parameters of the genetic operators. Before Holland's theory of schemata [Holland 1975] which justifies the use of binary encoding, complicated mutation operators needed to be invented to conform with the high-cardinality alphabets used. Even these operators usually had self-adapting capabilities which are still used in current research. They involve: encoding of the operator values along with the parameter set in the chromosome itself (introduced by Bagley, 1967, and recently investigated in [Bäck 1992; Smith, Fogatry 1996]), use of global rates of improvement data to centrally adjust them, etc.

Only a limited number of heuristics are available for the choice of efficient parameter values for the various operators. For example, it is now widely acceptable that relatively large values should be assigned to the mutation rate which should decrease with the number of generations. This has been confirmed by both practice and theoretical treatment for numerous families of test-problems in GAs. It is also clear that the choice of an efficient mutation rate should take into account the population size and the chromosome length.

But, how fast should parameters be changed during the execution of the GA? Should this change be strictly monotonic during the entire GA run? These and many more questions are still to be answered and, most probably, there are no answers of global applicability.

II. META-LEVEL GAs: FROM WEINBERG TO GREFENSTETTE

The problem of selecting and maintaining appropriate values for the various operators' parameter throughout the GA execution, clearly constitutes a separate optimization task itself and as such it could be treated by a separate "meta-level" GA. This is the key idea behind the work of Weinberg [Weinberg 1970] later to be extended by Mercer (1977) and Grefenstette [Grefenstette 1986]. An optimization problem mainly consists of two task levels: (i) a class of optimization algorithms must be chosen, and (ii) the parameters of the selected algorithm need to be tuned. Meta-level GAs emerge by employing GAs at both levels.

In his study “Computer Simulation of a Living Cell,” Weinberg proposed multilevel GAs for the adjustment of 15 parameters of his cell. The lower level GA was responsible for treating the problem as usual, but the adaptation of its parameters was left to a higher level GA. But, in the absence of Holland’s theory of schemata at the time, Weinberg used non-binary coding for the GAs and did not provide any simulation results.

Following De Jong’s paradigm [De Jong 1980], Grefenstette performed experiments across a five-function test bed of numerical optimization problems [Grefenstette 1986]. His aim was to determine optimal combinations of GA parameters for the given environment which, hopefully, could be successfully ported to other environments too. Instead of using any other method for producing and testing sets of GA parameters, he employed a meta-level GA for this task. Using this “automated” way, he was able to examine a large number of parameter combinations and export heuristic rules concerning both the way they affect the population evolution and the way they are correlated.

Grefenstette considered six parameters of the lower-level GA for optimization, namely the population size, the crossover rate, the mutation rate, the generation gap, the scaling window size, and the selection strategy.

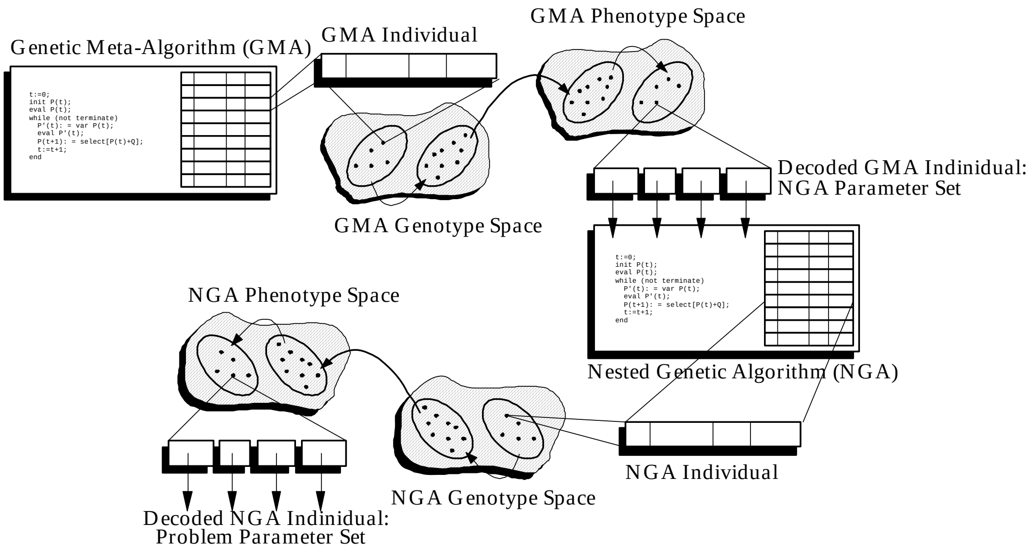
Grefenstette’s concern was not to use the meta-level GA as a dynamic component of the overall system to assist the lower-level GA throughout its search, but rather as a form of an “experiment generator”. Each low-level GA was initiated with a set of parameters provided by the meta-level GA which were held constant throughout its execution. Thus, the problem treated was to determine a near-optimal set of constant parameters rather than to dynamically adapt them during the algorithm run. It is this latter role of multilevel GAs that we investigate in this paper.

III. THE PROPOSED MODEL

The term “*genetic meta-algorithm*” was chosen over the existing term “*meta-level genetic algorithm*” since it seems to be more successful in capturing the idea of the higher-level algorithm being actively involved in the run-time progress of the optimization task by continuously adapting the parameters of the lower-level algorithm.

The proposed genetic meta-algorithm consists of two levels: the lower level genetic algorithm(s) henceforth called *nested genetic algorithms* (NGAs) that addresses the current optimization task itself, and the higher level genetic algorithm henceforth called *genetic meta algorithm* (GMA) whose task is to adapt the parameters of NGAs. The individuals of the *meta population* are encoded candidate optimal sets of parameters that are passed down to the nested GAs. The nested GAs execute for a specified number of iterations, its performance is monitored and the required feedback is returned to the meta algorithm so that a fitness can be assigned to each individual. When all individuals of the meta population have been evaluated, typical genetic procedures are followed by GMA to produce a new meta population to be evaluated. A representation of the flow of data is provided in Fig. 1 as an extension of [Fogel 1995, p. 39] for the case of the proposed meta algorithm. The two constituent parts of each GA, namely the algorithm itself and the population it operates upon, are explicitly depicted for clearness.

The structure of the genetic meta-algorithm is captured by the abstract code fragment of Fig. 2. Symbols `P`, `Q`, and `g`, represent the population, the intermediate population after selection and before application of genetic operators, and the generation number respectively. The prefixes `M` (or `m`) and `N` (or `n`) stand for “meta-” and “nested-” respectively.



```
t:=0;
init P(t);
eval P(t);
while (not terminate)
  P'(t): = var P(t);
  eval P'(t);
  P(t+1): = select[P(t)+Q];
  t:=t+1;
end
```

```
t:=0;
init P(t);
eval P(t);
while (not terminate)
  P'(t): = var P(t);
  eval P'(t);
  P(t+1): = select[P(t)+Q];
  t:=t+1;
end
```

Fig. 1. The various transformations involved (based on [Fogel 1995, p. 39])

```
mg := 0;
initialize( MP[mg] );
while (not terminate-meta) do
 for (each meta individual i) do
  ng := 0;
  NP[ng] := NBest[mg];
  parameters := decode( MP[mg][i] )
  while (not terminate-nested) do
   evaluate( NP[ng] );
   NQ := select( NP[ng] );
   NP[ng+1] := variation( NQ );
   ng := ng + 1;
  end
  if (NP[ng] better than NBest[mg+1])
   NBest[mg+1] := NP[ng];
  end
 end
 MQ := select( MP[mg] );
 MP[mg+1] := variation( MQ );
 mg := mg + 1;
end
```

Fig. 2: The genetic meta-algorithm

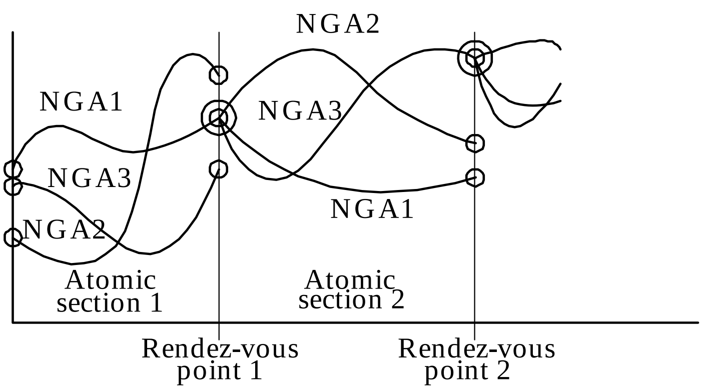


Fig. 3: The evolution of the GMA and the nested algorithms

The execution of the overall algorithm progresses in phases. At each phase all the nested GAs execute for the same pre-specified number of iterations. At the end of the phase, the nested population that was found to be best (in the sense of a criterion) is used to initialize all nested population of the following phase. This way, all the nested GAs of a phase have the same starting-point but different parameters leading to different evolution as depicted in Fig. 3. Although qualitative, Fig. 3 provides the basis for various points to be clarified.

The initial nested populations at the first iteration of the meta algorithm are chosen all randomly since no previous iteration exists to have a best nested population available. An alternative would be to create a single nested population randomly and pass it to all nested GAs. The current approach seems more advantageous since the more times we "throw the dice" the better chances we have to find a good starting population.

The end of each iteration of the meta algorithm introduces a *rendez-vous point* which is a point where all NGAs have been evaluated and the best nested population is determined to provide the kick-off for the subsequent iteration. This point is used for "synchronization" of the NGAs and is the only point where data from all the NGAs are available for use. Between such points lies an *atomic section*, i.e. a section where no data exchange can take place and each NGA completes a pre-specified number of iterations. The size of the atomic sections, referred to as *number of evaluation (nested) iterations*, was purposely chosen to be different implying that it may be subject to change by the meta algorithm.

THE NESTED PARAMETERS UNDER ADAPTATION

While numerous parameters of the nested GA can be selected for optimization by the meta algorithm, those with the greater impact on the overall system performance seem to include the following:

- Crossover rate
- Mutation rate
- Size of nested population
- Number of nested iterations
- The stress of the fitness scaling

The importance of continuously adapting on-line the parameters of the genetic various operators of the nested GAs, has been continuously stressed up to now. We argue that other nested parameters need to be treated similarly, i.e. to be dynamically redefined along the execution of the meta algorithm.

THE POPULATION SIZE

The population size, should not be treated as a parameter whose "optimal" value needs to be determined and kept constant throughout the algorithm run. Starting off with a small population, a GA can quickly discover approximate solutions. To maintain diversity and prevent valuable schemata from disappearing in the way, we need to progressively increase the population size. Put in different words, the only "memory" that a GA has at its disposal resides at the schemata stored in the population. As the algorithm progresses, its "memory requirements" are continuously increasing demanding thus for larger populations. Thus, the population size is also a parameter to be continuously traced during the GA run.

It is possible that the best nested population chosen at a rendez-vous point needs to be copied to a population of different size (smaller or greater) for each of the successive atomic sections of the nested GAs. To switch from a population $Q(t)$ of size n, to a population $Q(t+1)$ of size m, where $n \neq m$, we have, at least, the following alternatives:

- if $n>m$, we may select m out of the n individuals (without replacement), similarly to the way we select for mating,
- if $n \leq m$, we may:
  - produce an offspring population of size $m-n$ and get the union of the two populations,
  - use selection with replacement,
  - create and randomly initialize $m-n$ individuals.

NUMBER OF EVALUATION ITERATIONS

Usually, a GA progresses to fitter individuals logarithmically: there is a fast bootstrapping which tends to degrade along the run. At later stages of the execution it increasingly harder for the meta algorithm to determine the fittest nested population on the basis of their progress since some of the nested GAs may not succeed in finding a globally fitter individual in the specified number of nested iterations. At that point, nested population fitness resides more and more to luck.

Knowing in advance the logarithmic nature of the progress in genetic search, we may hardwire it to the meta algorithm instead of including it in the list of parameters of each nested GA to be optimized. Put in other words, we roughly know when to expect a globally best individual to come by and we may use this prediction (increased with a safety interval) to determine the number of atomic iterations for the following cluster of nested GAs.

DETERMINING THE FITNESS OF A NESTED ALGORITHM

In the work of Grefenstette the two quantities proposed by De Jong have been used to evaluate the progress of each lower level GA, namely the *on-line performance* and the *off-line performance*. Since they seem to be quite expressive, the obvious choice was to adopt them in out investigation too as a means of calculating the fitness of a nested GA. Based on this, we may then select the best nested population as the population produced by the fittest nested GA and then use this to initialize the populations of the NGAs at the subsequent meta iteration.

An alternative could be to define some “population metrics,” i.e. quantities to characterize the “goodness” of a population, probably based on the fitness of its best individual, its average fitness, its diversity, etc. An interesting criterion could also be defined based on [Altenberg 1994], namely the evolvability of each population.

IV. EXPERIMENTAL RESULTS

The proposed meta-algorithm was implemented and compared with the *standard genetic algorithm* SGA. SGA was proposed by Grefenstette based on the work of De Jong relatively to a set of parameters that produced good results for a number of situations. SGA has frequently proven quite hard to outperform and uses the following parameters: population size equal to 40, crossover rate equal to 0.6, mutation rate equal to 0.001, generation gap equal to 1 (no individual of a population survives as is to the next population), no fitness scaling, and an elitist selection strategy.

It is easy to predict that for trivial problems and problems of small dimensionality, the algorithms do not differ much. Moreover, in some extreme cases GMA may appear to be “overqualified”.

THE TASK ENVIRONMENT

Representing one of the standard test beds for GAs, the well known De Jong 5-function test environment was chosen to be used (Table 1). Results in various cases have proven to be remarkable, which in some extent is credited to the graceful adaptation of the crucial nested GA parameters.

V. CONCLUSIONS—DISCUSSION

Besides of its enhanced performance, the proposed genetic meta-algorithm also possesses other important advantages, including the following:

- It directly lends itself for parallel implementation.

- Compared to the scheme proposed in [Bäck 1992; Smith, Fogatry 1996], and numerous other references where crossover and mutation rates were coded in the chromosomes themselves, the current method for handling crossover and mutation rates provides a more straightforward encoding in the "supervising" meta algorithm. It is easier to identify the effect of the values selected for the rates and ensures that a parameter value is attained long enough (throughout an atomic section) to be objectively evaluated.
- Moreover, it is easy to include various other parameters of the nested GA in the optimization process of the meta-algorithm.

From the viewpoint of multilevel GAs adopted here, it doesn't seem to be very promising or even make much sense to proceed with the definition of additional higher GA levels.

It can be argued that the current approach lacks of any analogy to the biological phenomena observed and studied from natural systems. Although that is true, we argue that the main concern when designing artificial systems inspired by natural behavior is, from engineering point of view, higher efficiency and not a one-on-one copy.

Table 1: De Jong Five-Function Test Bed

$$f_1(x_i) = \sum_1^3 x_i^2$$

$$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_i)^2$$

$$f_3(x_i) = \sum_1^5 \mathrm{int}(x_i)$$

$$f_4(x_i) = \sum_1^{30} i x_i^4 + \mathrm{Gauss}(0,1)$$

$$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$

REFERENCES

Altenberg, L. 1994. The Evolution of Evolvability in Genetic Programming, in Advances in Genetic Programming, Cambridge, MA: MIT Press, pp. 47-74

Bäck, T. 1992. Self-Adaptation in Genetic Algorithms, in Proc. 1st European Conf. on Artificial Life, Varela, F. J.; Bourgine, P., Eds., Cambridge, MA: MIT Press, pp. 263-271

Bäck, T.; Hammel, U.; Schwefel, H. -P. 1997. Evolutionary Computation: Comments on the History and Current State, IEEE Trans. Evolutionary Computation, Vol. 1, No. 1, pp. 3-17

De Jong, K. 1980. Adaptive System Design: A Genetic Approach, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 10, No. 9, pp. 566-574

Dorigo, M.; Schnepf, U. 1993. Genetics-Based Machine Learning and Behavior-Based Robotics: A New Synthesis, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 23, No. 1, pp. 141-154

Fogel, D. B. 1995. Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, Piscataway, NJ: IEEE Press

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning, Reading, MA: Addison-Wesley

Grefenstette, J. J. 1986. Optimization of Control Parameters for Genetic Algorithms, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 16, No. 1, pp. 122-128

Holland, J. H. 1975. Adaptation in Natural and Artificial Systems, Ann Arbor: MI: Univ. of Michigan Press

Smith, J.; Fogatry, T. C. 1996. Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm, in Proc. 3rd IEEE Conf. on Evolutionary Computation, Piscataway, NJ: IEEE Press, pp. 318-323

Weinberg, R. 1970. Computer Simulation of a Living Cell, Doctoral Dissertation, University of Michigan