

Evolutionary Algorithms with on-the-fly Population Size Adjustment

A.E. Eiben E. Marchiori V.A. Valkó

Department of Artificial Intelligence, Vrije Universiteit Amsterdam
{gusz, elena, valko}@cs.vu.nl

Abstract. In this paper we evaluate on-the-fly population (re)sizing mechanisms for evolutionary algorithms (EAs). Evaluation is done by an experimental comparison, where the contestants are various existing methods and a new mechanism, introduced here. These comparisons consider EA performance in terms of success rate, speed, and solution quality, measured on a variety of fitness landscapes. These landscapes are created by a generator that allows for gradual tuning of their characteristics. Our test suite covers a wide span of landscapes ranging from a smooth one-peak landscape to a rugged 1000-peak one. The experiments show that the population (re)sizing mechanisms exhibit significant differences in speed, measured by the number of fitness evaluations to a solution and the best EAs with adaptive population resizing outperform the traditional genetic algorithm (GA) by a large margin.

1 Introduction

The effects of setting the parameters of EAs has been the subject of extensive research by the EA community and recently there is much attention paid to self-calibrating EAs¹ that can adjust their parameters on-the-fly (see e.g., [4, 6] for a review). The most attention and most publications have been devoted to the adjustment of parameters of variation operators. Adjusting population size is much less popular, even though there are biological and experimental arguments to expect that this would be rewarding. In natural environments, population sizes of species change and tend to stabilize around appropriate values according some factors such as natural resources and carrying capacity of the ecosystem [17, 21]. Looking at it technically, population size is the most flexible parameter in natural systems: It can be adjusted much more easily than, for instance, mutation rate. In evolutionary computing, however, population size is traditionally a rigid parameter. This is not only true in the sense that for the huge majority of EAs the population size remains constant over the run, but also for the EC research community that has not spent much effort on EAs with variable population sizes. Recently, Bäck *et al.* [2] have provided strong indications that adjusting the population size during a run could be more rewarding than varying

¹ We avoid the term “self-adaptive” since the common EC terminology strongly relates it to a particular way of adjusting parameters on-line, cf. [6, Chapter 8].

the operator parameters. This forms an important motivation for the present investigation. The primary technical objective of this study is to evaluate a number of adaptive population sizing strategies on abstract fitness landscapes. The main contributions of this paper are the following:

- drawing attention to and initiating further research on varying population size in EAs
- giving a brief overview of existing approaches and introducing a new population resizing technique,
- presenting an experimental comparison for a number of these techniques,
- providing freely available Java code for these methods and our test suite, allowing reproduction of our results and further research.

The relevance of this study, and possible future efforts in this direction, lies in the potential of self-calibrating EAs. Ultimately, it would be highly desirable to utilize the inherent adaptive power of an evolutionary process for adapting *itself* to a certain problem instance, while solving that very problem instance. We believe that the extra computational overhead (i.e., solving the self-calibration problem additionally to the given technical problem) will pay off, but a solid judgment of this hypothesis requires more research.

1.1 Related Work

A few researchers provided a theoretical analysis of the optimal population size in EAs. Goldberg described two methods for optimally sizing populations in GAs. In the first one [8, 9] he sized the population for optimal schema processing, in the second one [10], optimization was performed for accurate schema sampling. An overview of both methods can be found in [20]. Reeves [16] tried to specify the minimal population size for GA applications based on a theoretical background. The adopted principle was that every possible point in the search space should be reachable from the initial population by crossover only. The results show that the minimal size is depending on the alphabet cardinality and the string-length. Specifically, for binary representations with string size of 100 the minimal size should be about 15 – 18. Hansen, Gawelczyk and Ostermeier [11] gave a theoretical analysis of sizing the populations in $(1, \lambda)$ -Evolution Strategies with respect to the local progress.

There is also a number of empirical studies on population sizing. The Genetic Algorithm with Variable Population Size (GAVaPS) from Arabas [1], [15, p. 72–80] eliminates population size as an explicit parameter by introducing the age and maximum lifetime properties for individuals. The maximum lifetimes are allocated at birth depending on fitness of the newborn, while the age (initialized to 0 at birth) is incremented at each generation by one. Individuals are removed from the population when their ages reach the value of their predefined maximal lifetime. This mechanism makes survivor selection unnecessary and population size an observable, rather than a parameter. The Adaptive Population size GA (APGA) is a variant of GAVaPS where a steady-state GA is used, and the lifetime of the best individual remains unchanged when individuals grow older [2].

In [12, 14] Harik and Lobo introduce a parameter-less GA (PLGA) which evolves a number of populations of different sizes simultaneously. Smaller populations get more function evaluations, where population i is allowed to run four times more generations than the population $i + 1$. If, however, a smaller population converges, the algorithm drops it. The Random Variation of the Population Size GA (RVPS) is presented by Costa *et al.* in [3]. In this algorithm, the size of the actual population is changed every N fitness evaluations, for a given N . Hinting, Michalewicz and Peachey [13] presented an adaptive mechanism, in which three sub-populations with different population sizes are used. The population sizes are adapted at regular intervals (*epochs*) biasing the search to maximize the performance of the group with the mid-most size. The criterion used for varying the sizes is fitness diversity. Schlierkamp-Voosen and Mühlenbein [18] use a competition scheme between sub-populations to adapt the size of the sub-populations as well as the overall population size. There is a quality criterion for each group, as well as a gain criterion, which dictates the amount of change in the group’s size. The mechanism is designed in such a way that only the size of the best group can increase. A technique for dynamically adjusting the population size with respect to the probability of selection error, based on Goldberg’s research [10], is presented in [19].

2 Population resizing mechanisms in this comparison

For the present comparison we have selected a number of existing population resizing mechanisms to be implemented in our library. In particular, the GAVaPS from [1], the GA with adaptive population size (APGA) from [2], the parameter-less GA from [12], and three variants of the GA with Random Variation of Population Size (RVPS) from [3]. Initial testing has shown that GAVaPS was very sensitive for the *reproduction ratio* parameter and the algorithm frequently increased the size of the population over several thousand individuals, which resulted in unreliable performance. For this reason we removed it from further experimentation. Furthermore, we added a traditional genetic algorithm (TGA) as benchmark and introduced a new technique.

The new population resizing mechanism we introduce is based on improvements of the best fitness in the population. On fitness improvement the algorithm becomes more biased towards exploration increasing the population size, short term lack of improvement makes the population smaller, but stagnation over a longer period causes populations to grow again. The pseudo-code for the Population Resizing on Fitness Improvement GA (PRoFIGA) is given below. The intuition behind this algorithm is related to (a rather simplified view on) exploration and exploitation. The bigger the population size is, the more it supports explorative search. Because in early stages of an EA run fitness typically increases, population growth, hence exploration, will be more prominent in the beginning. Later on it will decrease gradually. The shrinking phase is expected to “concentrate” more on exploitation by reducing the genetic diversity in the decreasing populations. The second kind of growing is supposed to initiate re-

```

procedure PProFIGA
begin
  INITIALIZE population with random individuals
  EVALUATE each individual
  while not stop-condition do
    SELECT parents from the population
    RECOMBINE pairs of parents
    MUTATE the resulting offspring
    EVALUATE each of the offspring
    REPLACE some parents by some offspring
    if BEST_FITNESS_IMPROVED then
      GROW_POPULATION_1
    elsif NO_IMPROVEMENT_FOR_LONG_TIME then
      GROW_POPULATION_2
    else
      SHRINK_POPULATION
    fi
    EVALUATE the new individuals
  od
end

```

newed exploration in a population stuck in local optima. Technically, PProFIGA applies three kinds of changes in the population size:

1. If the best fitness in the population increases, the population size is increased proportionally to the improvement and the number of evaluations left until the maximum allowed. The formula used for calculating the growth rate X for GROW_POPULATION_1 is:

$$X = \text{increaseFactor} \cdot (\text{maxEvalNum} - \text{currEvalNum}) \cdot \frac{\text{maxFitness}_{\text{new}} - \text{maxFitness}_{\text{old}}}{\text{initMaxFitness}}$$

where *increaseFactor* is an external parameter from the interval (0, 1), *maxEvalNum* and *currEvalNum* denote the given maximum number of fitness evaluations and the current evaluation number, *maxFitness_{new}*, *maxFitness_{old}* and *initMaxFitness* are the best fitness values in the current generation, the same in the preceding generation and the best fitness value in the initial population. (Note that we assume the existence of *maxEvalNum*, which is very often present indeed. In case it is not given, a very large number can be used instead.)

2. The population size is increased by a factor Y if there is no improvement during the last V number of evaluations. In principle, the mechanism to

increase the population size in this step can be defined independently from the previous one, but in fact we use the same growth rate X for GROW_POPULATION_2 as for GROW_POPULATION_1.

3. If neither 1. nor 2. was executed, then the population size is decreased. For the decrease factor Z in SHRINK_POPULATION a little percentage of the current population size is used, e.g. (1–5%).

The new members of the population can be chosen by different strategies, like cloning some individuals from the population, or random generation of new individuals. In this study we use cloning of good individuals that are chosen by tournament selection from the actual population.

The individuals to be replaced can be selected by, for instance, an “anti-tournament” selection. The size is not decreased further after a certain minimal population size is reached. Note that PRoFIGA uses tournament selection and delete worst replacement, together with elitism, hence the best fitness of the population cannot decrease, only stagnate.

3 Test suite: Spears’ multimodal problems

When choosing the test suite we deliberately avoided popular, but ad hoc collections of objective functions for reasons outlined in [5] and [6, Chapter 14]. We have chosen the multimodal problem generator of Spears [22] that has been designed to facilitate systematic studies on GA behavior. This generator creates random problems with a controllable size and degree of multi-modality. The random bit-string multi-modality problem generator constructs a number (the degree of multi-modality) of random L -bit strings, where each string represents the location of a peak in an L -dimensional space. The problem consists of identifying the location of the highest peak. The heights of the peaks can be generated using different functions: constant (all peaks have the same height), linear (the heights of peaks belong to a line), 1–square root, and logarithm-based.

The difficulty of the problem depends on the number of peaks, the height of the lowest peak (the higher it is, the more difficult the problem is), the distribution of the peak-heights (the more peaks have heights close to the global optimum, the more difficult the problems is), and the random layout of the peaks in the search space (the more isolated the peaks are, the more difficult the problem is). The first three features can be controlled externally by parameters, the distribution is created randomly in each run of the generator. To calculate the fitness of the individuals, first the nearest peak to an individual is determined: for a given string \bar{x} let $Peak_{near}(\bar{x})$ be such that

$$Hamming(\bar{x}, Peak_{near}(\bar{x})) = \min_{i=1}^{\mathcal{P}}(Hamming(\bar{x}, Peak_i)),$$

in case of multiple peaks at the same distance, choose the highest neighboring peak. Then, the fitness value of a binary string chromosome in the population is determined by taking the number of bits the string has in common with the nearest peak, divided by L , and scaled by the height of the nearest peak:

$$f(\bar{x}) = \frac{L - \text{Hamming}(\bar{x}, \text{Peak}_{\text{near}}(\bar{x}))}{L} \cdot \text{height}(\text{Peak}_{\text{near}}(\bar{x})).$$

Note that the fitness assumes values between 0 and 1.

Our test suite consists of 10 different landscapes, where the height of the lowest peak is always 0.5, the distribution of the peak-heights is linear and the number of peaks ranges from 1 to 1000 through 1, 2, 5, 10, 25, 50, 100, 250, 500, and 1000.

4 Performance measures

In order to analyze and assess the performance of the algorithms considered in our study, we perform 100 independent runs on each problem instance, and consider three statistics to measure algorithm efficiency and effectivity.

- The first effectivity measure is Success Rate (SR) that gives the percentage of runs in which the optimum (the highest peak) was found.
- The second effectivity measure is Mean Best Fitness (MBF). It is the average of the best fitness in the last population over all runs.
- Efficiency (speed) is measured by the Average number of Evaluations to a Solution (AES), which shows the number of evaluations it takes on average for the successful runs to find the optimum. If a GA has no success ($SR = 0$) then the AES measure is undefined.

5 Algorithm setups

We use a traditional GA (TGA) as baseline algorithm for our comparison. The TGA’s parameters are shown in Table 1. The population size $N = 100$ has been “optimized” through conventional hand-tuning comparing a number of different population sizes. All EAs in this comparison follow this setup, hence the algorithms differ only in the population (re)sizing mechanism they apply.

| | |
|-----------------------|-----------------------------------|
| GA model | steady-state |
| Representation | bit-string |
| Chromosome length (L) | 100 |
| Population size (N) | 100 |
| Recombination | 2-point crossover ($p_c = 0.9$) |
| Mutation | bit-flip ($p_m = 1/L$) |
| Selection | 2-tournament |
| Replacement | delete worst 2 |
| Max. no. of evals | 10.000 |

Table 1. TGA setup

Further details, specific for the particular algorithms, are listed below. For APGA [2] we consider the variant which assigns lifetime values according to the following bi-linear function:

$$\begin{cases} MinLT + \eta \frac{fitness[i] - MinFit}{AvgFit - MinFit} & if AvgFit \geq fitness[i] \\ \frac{1}{2}(MinLT + MaxLT) + \eta \frac{fitness[i] - AvgFit}{MaxFit - AvgFit} & if AvgFit < fitness[i] \end{cases}$$

with $MinLT$ and $MaxLT$ equal to 1 and 11, respectively. The parameter-less GA [12] is run in parallel with the following 8 population sizes: 2, 4, 8, 16, 32, 64, 128 and 256. For RVPS (variant RW) [3] is used with insertion of randomly generated individuals and removal of the worst individuals. The minimal population size is set to 15, the maximal size to 300, the time between two resizing is 100 evaluations (the evaluation of the newly generated individuals in RVPS RW is not counted into this number). PRoFIGA (variant TMW²), where the population sizing mechanism uses 2-tournament selection for selecting individuals from the actual population, and where the following values are used for the other parameters of the population sizing mechanism: *increaseFactor* of 0.1, *decreaseFactor* of 0.4, *minPopSize* of 15 and *maxPopSize* 1000.

The algorithms have been implemented in PEA (Programming library for EAs), a new EA software developed for the present investigation (the Java code is available at <http://www.cs.vu.nl/ci>). PEA is written in Java, hence it can be run on all platforms that have a Java Runtime Environment installed.

6 Results

The results of our main experimental series are given in the graphs with a grid background in Figure 1 and the left hand side of Figure 2.

The AES plots are exhibited in Figure 1 (left). These graphs show clear differences between the algorithms. There are, however, no significant differences between the problem instances when only looking at the speed curves (except for the parameter-less GA). Apparently, finding a solution does not take more evaluations on a harder problem that has more peaks. (Although it should be noted that for harder problems the averages are taken over fewer runs, cf. the SR figures below, which reduces the reliability of the statistics.) This is an interesting artifact of the problem generator that needs further investigations. The increasing problem hardness, however, is clear from the decreasing average quality of the best solution found (MBF), cf. Figure 1 (right) and the decreasing probability of finding a solution (SR), cf. Figure 2 (left).

We can rank the population (re)sizing methods based on the AES plots: APGA is significantly faster than the other methods, followed by PRoFIGA. The traditional GA comes third. The parameter-less GA is only competitive for easy problems and the RVPS RW is clearly inferior to the other methods.

² See [23] for details

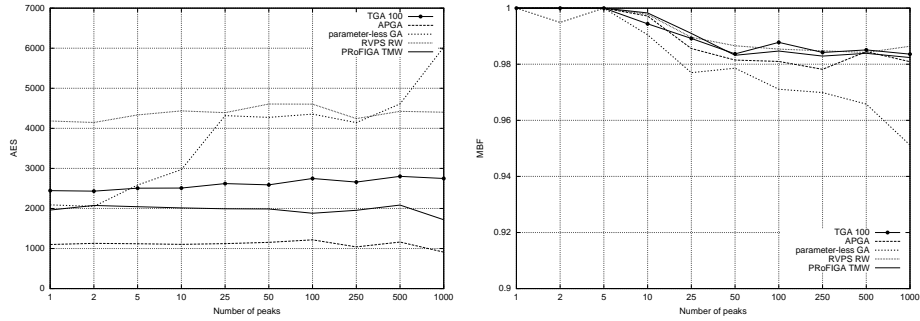


Fig. 1. AES (left) and MBF (right) of TGA, APGA, the parameter-less GA, RVPS and PRoFIGA with max-eval = 10000

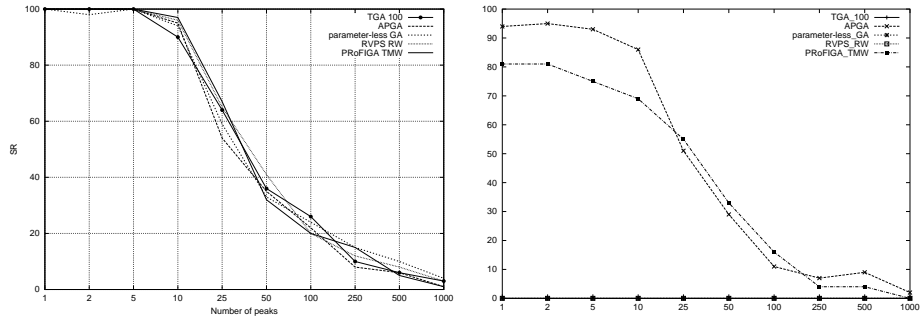


Fig. 2. SR of TGA, APGA, the parameter-less GA, RVPS and PRoFIGA with max-eval = 10000 (left) and with max-eval = 1500 (right)

The SR and MBF results are quite homogeneous, with only one negative outlier, the parameter-less GA. It seems that we cannot rank the algorithms by their effectivity. However, this homogeneity is a consequence of our choice of the maximum number of fitness evaluations in the termination criterion. Apparently it is “too” high allowing all contestants to reach the performance of the champions – be it slower. As a control experiment, we repeated all runs with the maximum number of fitness evaluations set to 1500. The resulting success rates are given in Figure 2 (right), showing great differences. APGA and PRoFIGA obtain somewhat worse, but comparable SR results as before, but the other algorithms never find a solution yielding SR = 0 over all peaks.

Forced by space limitations we cannot provide an analysis, nor a graphical illustration of population size dynamics for the algorithms. In summary, the experiments indicate that each algorithm has a “preferred” range of population size (except the parameter-less GA), rather independently from the hardness of the problem instance.

7 Conclusions

Looking at the results we can conclude that adapting population sizes in an EA can certainly pay off. The gains in terms of efficiency, measured by the number of fitness evaluations needed to find a solution, can be significant: the winner of our comparison (APGA) achieves the same success rate and mean best fitness as the traditional GA with less than half of the work, and even the second best (PRoFIGA) needs 20% fewer evaluations. Our second series of experiments shows that such an increase in speed can be converted into increased effectivity, depending on the termination condition. Here again, the winner is APGA, followed by PRoFIGA. It should be noted that we do not claim that on-the-fly population (re)sizing is necessarily better than traditional hand-tuning of a constant population size. Two GAs from this comparison (the parameter-less GA and RVPS RW) are much slower than the traditional GA.

Finding a sound explanation for the observed differences in algorithm behavior is a hard nut to crack. Our most plausible hypothesis is that the superior performance of APGA is due to the lifetime principle that eliminates explicit survivor selection and makes population size an observable instead of a parameter. However, it should be noted that using this idea does not mean that the number of EA parameters is reduced. In fact, it is increased in our case: instead of N in the TGA, the APGA introduces two new parameters, *MinLT* and *MaxLT*.

The present results can be naturally combined with those of Bäck *et al.* who worked on a test suite containing commonly used objective functions and found that APGA outperformed TGA and other GAs that used adaptive crossover and mutation mechanisms [2]. Our findings here amplify their conclusions on the superiority of APGA. Of course, highly general claims are still not possible about APGA. But these results together form a strong indication that incorporating on-the-fly population (re)sizing mechanisms based on the lifetime principle in EAs is a very promising design heuristic definitely worth trying and that APGA is a successful implementation of this general idea.

Acknowledgement The authors acknowledge the valuable contribution of M. Jelasity and T. Buresch in performing the investigations reported here.

References

1. J. Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS – a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 73–78. IEEE Press, Piscataway, NJ, 1994.
2. T. Bäck, A.E. Eiben, and N.A.L. van der Vaart. An empirical study on GAs "without parameters". In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the 6th Conference on Parallel Problem Solving from Nature*, number 1917 in Lecture Notes in Computer Science, pages 315–324. Springer, Berlin, 2000.
3. J. Costa, R. Tavares, and A. Rosa. An experimental study on dynamic random variation of population size. In *Proc. IEEE Systems, Man and Cybernetics Conf.*, volume 6, pages 607–612, Tokyo, 1999. IEEE Press.

4. A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
5. A.E. Eiben and M. Jelasity. A critical note on experimental research methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)*, pages 582–587. IEEE Press, 2002.
6. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
7. S. Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, 1993.
8. D.E. Goldberg. Optimal population size for binary-coded genetic algorithms. *TCGA Report*, No. 85001, 1985.
9. D.E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 70–79. Morgan Kaufmann, San Francisco, 1989.
10. D.E. Goldberg, K. Deb, and J.H. Clark. Genetic Algorithms, Noise, and the Sizing of Populations. *IlligAL Report*, No. 91010, 1991.
11. N. Hansen, A. Gawelczyk, and A. Ostermeier. Sizing the population with respect to the local progress in $(1,\lambda)$ -evolution strategies – a theoretical analysis. In *Proceedings of the 1995 IEEE Conference on Evolutionary Computation*, pages 80–85. IEEE Press, Piscataway, NJ, 1995.
12. G.R. Harik and F.G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 258–265, Orlando, Florida, USA, 1999. Morgan Kaufmann.
13. R. Hinterding, Z. Michalewicz, and T.C. Peachey. Self-adaptive genetic algorithm for numeric functions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science, pages 420–429. Springer, Berlin, 1996.
14. F.G. Lobo. *The parameter-less Genetic Algorithm: rational and automated parameter selection for simplified Genetic Algorithm operation*. PhD thesis, Universidade de Lisboa, 2000.
15. Z. Michalewicz. *Genetic Algorithms + Data structures = Evolution programs*. Springer, 3th edition, 1996.
16. C.R. Reeves. Using genetic algorithms with small populations. In Forrest [7], pages 92–99.
17. J. Roughgarden. *Theory of Population Genetics and Evolutionary Ecology*. Prentice-Hall, 1979.
18. D. Schlierkamp-Voosen and H. Mühlenbein. Adaptation of population sizes by competing subpopulations. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1996.
19. R.E. Smith. Adaptively resizing populations: An algorithm and analysis. In Forrest [7].
20. R.E. Smith. *Population sizing*, pages 134–141. Institute of Physics Publishing, 2000.
21. J. Song and J. Yu. *Population system control*. Springer, 1988.
22. W.M. Spears. *Evolutionary Algorithms: the role of mutation and recombination*. Springer, 2000.
23. V.A. Valkó. Self-calibrating evolutionary algorithms: Adaptive population size. Master’s thesis, Free University Amsterdam, 2003.