An empirical study on GAs “without parameters”

Th. Bäck[1,2] and A.E. Eiben[1,3] and N.A.L. van der Vaart [1]

[1] Leiden University, [2] ICD Dortmund, [3] Free University Amsterdam

Abstract. In this paper we implement GAs that have one or more parameters that are adjusted during the run. In particular we use an existing self-adaptive mutation rate mechanism, propose a new mechanism for self-adaptive crossover rates, and redesign an existing variable population size model. We compare the simple GA with GAs featuring only one of the parameter adjusting mechanisms and with a GA that applies all three mechanisms - and is therefore almost “parameterless”. The experimental results on a carefully designed test suite indicate the superiority of the parameterless GA and give a hint on the power of adapting the population size.

Introduction

Traditional genetic algorithms have parameters that must be specified before the GA is run on a problem. The most important parameters are the strategy or control parameters: population size (N), mutation rate (p_m), and crossover rate (p_c). The optimum setting of these parameters can be different for each problem. Moreover, the optimal values can be different in different phases of one single run on a given problem. Setting parameter values correctly is, therefore, a hard task. In general, there are two major forms of setting parameter values: parameter tuning and parameter control [EHM99]. Parameter tuning is the usual approach of finding good parameter values *before* the run of the GA and then these static values are used during the whole run. Parameter control is different because it changes initial parameter values *during* the run. Parameter control itself has three variants: deterministic, adaptive and self-adaptive. Deterministic means that parameters are changed according to a rule which uses no feedback from the GA, usually it is some sort of time-varying scheme. For adaptive control, feedback does take place and the values are changed in direction or magnitude depending on this feedback. Self-adaptivity is the idea of evolution of the parameters of evolution; the values are encoded into the chromosomes and are also subject to mutation, crossover and selection. The better parameter values will survive because they produce better offspring. Throughout this paper we maintain this terminology and use the term self-adjusting if we do not want or cannot specify which form of parameter control is used.

The main objective of this paper is to investigate the feasibility of eliminating the three main parameters, N, p_m, and p_c. That is, we are looking for experimental evidence on the performance of a parameterless GA. The race is not won in advance by any of the GA variants, for we are dealing with a clear trade-off situation. On the one hand, on-line parameter adjustment causes a learning overhead, that is the GA is solving a problem and is learning good parameter values at the same time. This might cause a performance decrease w.r.t. using steady parameters. On the other hand, adjustable parameters provide the GA with valuable flexibility. If the GA can adjust its parameters (sub)optimally it might cause increased efficiency. A priori it cannot be predicted which effect will influence the GA performance more.

Previous research on the mutation rate (p_m), crossover rate (p_c) and population size (N), has focused on either adapting/self-adapting *one* of these parameters or on adjusting two or all of these parameters in a non self-adaptive fashion (i.e. adaptive or dynamic parameters). As for crossover there has not been previous research on self-adapting the crossover rate. The technical research objectives here are threefold.

1. Designing and investigating a new method for self-adapting the crossover rate.
2. Investigating the effect of self-adjusting the parameters p_m, p_c and N separately; a self-adaptive p_m, a self-adaptive p_c and an adaptive N. (Notice that a self-adaptive population size wouldn't make much sense.)
3. Investigating the combined use of self-adjusting p_m, p_c and N within a completely self-adjusting GA.

Test suite

To gain relevant experimental feedback we have carefully chosen a number of test functions. For selecting the test function we followed the guidelines after [WMRD95, BäMi97] and required that the test suite should

1) include a few unimodal functions for comparison of efficiency (convergence velocity),
2) contain nonlinear, nonseparable problems,
3) contain scalable functions,
4) have a standard form,
5) include a few multimodal functions of different complexity with a large number of local optima,
6) not have only multimodal functions that have a regular arrangement of local optima, (because this might favour GA-operators that exploit the regularity),
7) contain high-dimensional functions, because these are better representatives of real-world applications.

The following test suite of five functions conforms to the rules listed above: f_1 is the sphere model after De Jong [DeJo75], f_2 is the generalized Rosenbrock function [Rose60,HoBä91], f_3 is the generalized Ackley function [Ackl87,BäSc93], f_4 is the generalized Rastrigin function [TöZi89,HoBä91], and f_5 is the fully deceptive six-bit function [Deb97] (this is the only function in the test set that uses 6 bits/variable). In order to comply to a standard form; they all have dimension n=10 and use 20 bits/variable (except f_5).

$$f_1(\bar{x}) = \sum_{i=1}^{n} x_i^2$$

$$f_2(\bar{x}) = \sum_{i=1}^{n-1} \left(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\right)$$

$$f_3(\bar{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

$$f_4(\bar{x}) = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$$

$$f_5(\bar{x}) = n - \sum_{i=1}^{n} \begin{cases} \frac{0.92}{4}(4 - x_i) & if\ x_i \leq 4 \\ \frac{1.00}{4}(x_i - 4) & if\ x_i \succ 4 \end{cases} \quad , x_i\ is\ \#of\ 1\text{'}s \in gene\, i$$

Algorithms

The basic traditional GA used for this research is a steady-state GA using Gray coding, a mutation rate of $p_m = 1/l$, and a crossover rate of $p_c = 0.90$. The population size is N = 60. The chromosome length will be 220 bits (80 bits for f_5), which consists of 10x20=200 bits (10x6=60 bits for f_5) for the ten function variables (dimension n=10) and at the end 2x10=20 bits for the two self-adaptive parameters p_m and p_c. Although the p_m and p_c are not used by the TGA, they are added and calculated for two reasons. The first reason is that, compared to the other GA's, any (dis)advantage from a different length of the bitstring is ruled out. Second reason is that now, at the end of a TGA run, one can check if the values of p_m and p_c are totally random, as they should be. We use uniform crossover and parents are selected through tournament selection with tournament size 2. The replacement strategy is delete worst; the two worst members of a population are deleted to make place for two new members. The initialization of the population for each function comes from random bits stored in 30 files. These files will be used for each different GA. This guarantees that every GA and every function starts with the same population and no (dis)advantage is caused by a different initialization. The GA terminates when the (known) optimum is found or the maximum number of fitness evaluations is reached. The maximum number of evaluations is 100.000 for f_1, f_3 and f_5 and 500.000 for f_2 and f_4.

Self-adaptive mutation

This mechanism is implemented after Bäck [Bäck92a, Bäck92b] and Fogarty and Smith [FoSm96]: a self-adaptive mutation rate between 0.001 ($< 1/l$) and 0.25 is encoded in extra bits at the tail of every individual. For each member in the starting population the rate will be completely random within that range. Mutation then takes place in two steps. First only the bits that encode the mutation rate are mutated and immediately decoded to establish the new mutation rate. This new mutation rate is applied to the main bits (those encoding a solution) of the individual. The GA using this mechanism is called self-adaptive mutation GA (SAMGA).

Self-adaptive crossover

To our knowledge a stand-alone self-adaptive p_c on a GA has not been used before. Previous research was done on adaptive p_c e.g. [Davi89] or on self-adapting p_c simultaneously with self-adapting the p_m in a trade-off [Juls95]. In our method a self-adaptive crossover rate between 0 and 1.0 is coded in extra bits at the tail of every individual (initialized randomly). When a member of the population is selected for reproduction by the tournament selection, a random number r below 1 is compared with the member's p_c. If r is lower than p_c, the member is ready to mate. If both selected would-be parents are ready to mate two children are created by uniform crossover, mutated and inserted into the population. If it is not lower, the member will only be subject to mutation to create one child which undergoes mutation and survivor selection immediately. If both parents reject mating, the two children are created by mutation only. If one parent is willing to mate and the other one does not, then the parent that is not in for mating is mutated to create one offspring, which is inserted in the population immediately. (Note that in such a case we deviate from the (N+2) scheme and use (N+1) instead.) The willing parent is on hold and the next parent selection round only picks one other parent.

Adaptive population size

Following the ideas of Arabas, Michalewicz and Mulawka with GAVaPS [Mich94, p.70], every new individual is allocated a (remaining) lifetime or life span according to the individuals fitness by a bi-linear scheme. Each cycle, the remaining lifetime (RLT) of all the members in the population is decremented by one. However, we make an exception for the fittest member, whose RLT is left unchanged. If the RLT of an individual reaches zero it is removed from the population. The bi-linear formula in GAVaPS is for functions to be maximized. Therefore we adjust this formula for the functions in this investigation. In this formula MinLT and MaxLT stand for the allowable minimum and maximum lifetime of an individual. The other variables are linked with the current state of the search. These variables are fitness(i), AvgFit, BestFit and WorstFit. They stand for the fitness of individual i, average fitness, best fitness and worst fitness of the current living population. The individual for which the remaining lifetime (RLT) is being calculated, is considered to be already inserted into the population in this scheme and therefore (may) influence AvgFit, WorstFit and BestFit.

$$RLT(i) = \begin{cases} MinLT + \eta \cdot \dfrac{WorstFit - fitness(i)}{WorstFit - AvgFit} & \text{if } fitness(i) \geq AvgFit \\ \frac{1}{2}(MinLT + MaxLT) + \eta \cdot \dfrac{AvgFit - fitness(i)}{AvgFit - BestFit} & \text{if } fitness(i) < AvgFit \end{cases}$$

$$\eta = \tfrac{1}{2}(MaxLT - MinLT)$$

The values for MinLT and MaxLT are set to 1 and 11 in this study, because initial runs with different values indicated that MaxLT=11 delivers good performance. (One could say that choosing a population size N is now shifted to choosing a maximum lifetime MaxLT.) The function RLT(i) gives a (remaining) lifetime of 1 to 11 to each new individual i that enters the population. If its fitness(i) is better than the average fitness, it gets a life span of 7 to 11; if it is worse it gets a life span of 1 to 6. In the higher subrange the much-better-than-average individuals get higher values than the somewhat-better-than-average individuals. The same procedure applies for the lower subrange: the worst individuals get much lower lifetimes than the almost-average individuals. The initial population consists of 60 individuals.

Note that our GA is different from GAVaPS in that it is a steady-state GA instead of a generational GA. The evolution process is long, so it is likely that eventually every individual will die of old age.

Performance measures

With every one of the five different GA’s, 30 runs will be done per test function, which will make a total of 750 runs. In all GAs the best fitness and the average fitness of the population is monitored until a run terminates. For the self-adjusting GA’s the relevant parameters (p_m, p_c and population size) will also be monitored during each run. The speed of optimization will be measured by the Average number of Evaluations on Success (AES), how many evaluations did it take - on average - for the successful runs to find the optimum.. The Success Rate (SR) shows how many of the runs were successful in finding the optimum. If the GA is somewhat unsuccessful (SR < 30), the measurement MBF (Mean Best Fitness) shows how close the GA can come to the optimum. If the SR=30, then the MBF will be 0, because every run found the optimum 0. The MBF includes the data of all 30 runs in it, the successful and the unsuccessful ones.

Experimental results

In this section we present the main results of five algorithm variants. A full overview of experiments can be found in [vdVaart99]. The simple TGA will serve as a benchmark for the other variants. Three GAs use one extension to the TGA: the GA with self-adaptive mutation rate only (SAMGA), the GA with self-adaptive crossover rate only (SAXGA), and the GA with adaptive population size only (APGA). Finally, we study the GA featuring all extensions, called SAMXPGA.

Traditional Genetic Algorithm					
	f_1	f_2	f_3	f_4	f_5
SR	30	0	30	30	0
AES st. dev.	14,669 ±1,934	-	16,042 ±6,502	163,498 ±61,614	-
MBF St. dev.	0	0.281 ±0.317	0	0	0.64 ±0.103
Avg p_m st.dev.	0.121 ±0.076	0.126 ±0.061	0.131 ±0.065	0.123 ±0.054	0.13 ±0.048
Avg p_c st.dev.	0.454 ±0.223	0.473 ±0.25	0.559 ±0.259	0.545 ±0.222	0.504 ±0.18

Table 1. SR, AES, MBF, avg. p_m and avg. p_c for the TGA at the end of all runs. The self-adaptive p_m and p_c are not used by the TGA, but are added to the table to see if they are truly random.

Note that the interval for p_m is [0.001..0.25] and for p_c [0..1.0]. If they are really randomly divided, the average p_m and p_c should be respectively around 0.1245 and 0.5. Table 1 indicates that they are indeed random and therefore have no influence on, or are influenced by, the TGA.

Genetic Algorithm with Self-Adaptive Mutation (SAMGA)					
	f_1	f_2	f_3	f_4	f_5
SR	30	0	30	14	0
AES st.dev.	18,345 ±2,847	-	17,042 ±2,392	378,178 ±84,301	-
MBF st.dev.	0	0.376 ±0.463	0	1.758 ±2.869	0.688 ±0.1
Avg p_m st.dev.	0.0021 ±0.0017	0.0029 ±0.0021	0.0024 ±0.0015	0.002 ±0.0014	0.0023 ±0.0016
Avg p_c st.dev.	0.471 ±0.247	0.47 ±0.241	0.467 ±0.285	0.458 ±0.259	0.492 ±0.254

Table 2. SR, AES, MBF, avg. p_m and avg. p_c for the SAMGA at the end of all runs. The self-adaptive p_c is not used by the SAMGA, but is added to the table to see if it is truly random.

Genetic Algorithm with Self-Adaptive Crossover (SAXGA)					
	f_1	f_2	f_3	f_4	f_5
SR	30	0	30	30	0
AES st.dev.	16,033 ±1,878	-	21,486 ±14,050	170,820 ±75,891	-
MBF st.dev.	0	0.355 ±0.342	0	0	0.581 ±0.09
Avg p_m st.dev.	0.1191 ±0.0592	0.12 ±0.0628	0.1358 ±0.0646	0.1319 ±0.067	0.1404 ±0.0459
Avg p_c st.dev.	0.68 ±0.188	0.57 ±0.266	0.565 ±0.25	0.617 ±0.246	0.514 ±0.161

Table 3. SR, AES, MBF, avg. p_m and avg. p_c for the SAXGA at the end of all runs. The self-adaptive p_m is not used by the SAXGA, but is added to the table to see if it is truly random.

Genetic Algorithm with Adaptive Population Size (APGA)					
	f1	f2	f3	f4	f5
SR	30	0	30	30	0
AES st.dev.	9,645 ±1,240	-	19,919 ±14,903	163,494 ±58,433	-
MBF st.dev.	0	0.133 ±0.186	0	0	0.587 ±0.136
Avg pm st.dev.	0.1409 ±0.0703	0.1291 ±0.0777	0.0964 ±0.072	0.1412 ±0.0704	0.1089 ±0.0606
Avg pc st.dev.	0.558 ±0.258	0.586 ±0.257	0.432 ±0.278	0.496 ±0.252	0.485 ±0.293
Avg remaining lifetime st.dev.	6.36 ±0.73	5.86 ±0.75	6.23 ±0.57	6.41 ±0.7	5.04 ±1.55
Avg population size st.dev.	14.1 ±2.66	13.5 ±2.6	12.7 ±2.02	12.8 ±2.45	7.8 ±2.84

Table 4. SR, AES, MBF, avg. p_m, avg. p_c, avg. RLT and avg. population size for the SAPGA at the end of all runs. The self-adaptive p_m and p_c are not used by the SAPGA, but are added to the table to see if they are truly random.

All-in-one Genetic Algorithm (SAMXPGA)					
	f_1	f_2	f_3	f_4	f_5
SR	30	0	30	30	0
AES st.dev.	16,505 ±4,906	-	16,821 ±8,869	143,933 ±94,496	-
MBF st.dev.	0	0.199 ±0.244	0	0	0.568 ±0.11
Avg pm st.dev.	0.0028 ±0.0023	0.0205 ±0.0261	0.0038 ±0.0057	0.0032 ±0.0024	0.024 ±0.0403
Avg pc st.dev.	0.495 ±0.297	0.622 ±0.27	0.563 ±0.213	0.629 ±0.286	0.597 ±0.291
Avg remaining lifetime st.dev.	6.13 ±0.92	5.75 ±1.47	6.32 ±0.96	6.17 ±0.92	4.8 ±1.83
Avg population size st.dev.	13.4 ±5.36	10.8 ±4.33	11.4 ±3.83	11.6 ±5.09	9.2 ±5.7

Table 5. SR, AES, MBF, avg. p_m, avg. p_c, avg. RLT and avg. population size for the SAMXPGA at the end of all runs

The ranking of the GAs on each function					
	TGA	SAMGA	SAXGA	APGA	SAMXPGA
f_1	2	5	3	1	4
f_2	3	5	4	1	2
f_3	1	2½	5	4	2½
f_4	2½	5	4	2½	1
f_5	4	5	2½	2½	1
Total	12½	22½	18½	11	10½
End Ranking	***3***	***5***	***4***	***2***	***1***

Table 6. Global comparison of GA versions on the test suite

In order to give a clear overview on the performance of all GA variants we rank the GAs for each function. In particular, we award the fastest GA, or the GA closest to the minimum, one point, the second fastest GA two points, and so on, so the worst performing GA for a given function gets five points. If for a particular function two GA’s finish real close to each other, we award them equally: add the points for both those rankings and divide that by two. After calculating these points for each function and each GA variant we add the points for all the functions to form a total for each GA. The GA with the least points has the best overall performance.

Conclusions

The performance of the self-adapting parameters p_m and p_c is disappointing when they are on their own (in SAMGA and SAXGA). The most likely reason is that time spent on searching good parameter values is time taken away from finding the optimum. Another important reason for the SAMGA is that it does not use the 1/5 success rule as Fogarty and Smith did [FoSm96]. They generate five individuals with mutation and choose only to insert the best one into the population. This gives extra survival pressure and that seems to give better values for p_m at a faster rate. The SAXGA is not as bad as the SAMGA. It does perform quite well on the relatively easy functions f_1 and f_5, but on the other three functions it finishes one but last or last. Adapting the population size in the SAPGA, on the other hand, is very effective. The maximum lifetime of 11 sets high pressure on survival and this keeps the population size small. This clearly benefits the search, for the SAPGA is the second best GA in this research and not far from being the best.

The overall competition ends in a close finish with the SAMXPGA as number one and the SAPGA right on its heels. It did surprise us using adaptive population sizes proved to be the key feature to improve the basic TGA. Alone, or in combination with the self-adaptive variation operators, the mechanism to adjust the population size during a run causes a consequent performance improvement w.r.t. the benchmark GA.

The main objective of this paper has been the study of a GA that has no parameters – or at least is freed from the three main parameters p_m, p_c and N. Our experiments showed that such a GA outperforms the four other variants we examined in this study. Nevertheless, it could be hypothesized that the main source of this victory is the adaptation of the population size (N). The self-adaptive p_m and p_c do contribute to its good performance (compare APGA with SAMXPGA) but we also conjecture that it works the other way around too. Namely, that the high selective pressure in the relatively small adaptive populations helps to get better p_m and p_c values.

These outcomes give a very strong indication that, in contrary to past and present practice (where quite some effort is devoted to tuning or on-line controlling of the application rates of variance operators), studying control mechanisms for variable population sizes should be paid more attention.

Our future research is directed to verifying these outcomes on a larger test suite, including real-world problems and a detailed analysis of varying population sizes (comparing deterministic time-varying schemes with adaptive control).

References

[Ackl87] Ackley, D.H.: A Connectionist Machine for Genetic Hillclimbing, Kluwer, Boston, 1987.

[Bäck92a] Bäck, T.: The Interaction of Mutation Rate, Selection, and Self-Adaptation within a Genetic Algorithm, in [MäMa92], pp.85-94.

[Bäck92b] Bäck, T.: Self-Adaptation in Genetic Algorithms, in [VaBo92], pp.263-271.

[BFM97] Bäck, T., Fogel, D.B., & Michalewicz, Z. (editors): Handbook of Evolutionary Computation, Institute of Physics Publishing & Oxford University Press, New York, 1997.

[BäMi97] Bäck, T., & Michalewicz, Z.: Test Landscapes, in [BFM97], Chapter B2.7, pp.14-20.

[BäSc93] Bäck, T., & Schwefel, H-P.: An Overview of Evolutionary Algorithms for Parameter Optimization, Evolutionary Computation, Vol.1, No.1, pp.1-23, 1993.

[Davi89] Davis, L.: Adapting Operator Probabilities in Genetic Algorithms, in [Scha89], pp. 61-69.

[Deb97] Deb, K.: Deceptive Landscapes, in [BFM97], Chapter B2.7, pp.1-5.

[DeJo75] De Jong, K.A.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. Ph.d. dissertation, Department of Computer Science, University of Michigan, Ann Arbor, Michigan, 1975.

[EHM99] Eiben, A.E., Hinterding, R., & Michalewicz, Z.: Parameter Control in Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, Vol.3, No.2, pp.124-141, 1999.

[Eshe95] Eshelman, L.J. (editor): Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Francisco, California, 1995.

[FoSm96] Fogarty, T.C., & Smith, J.: Self Adaptation of Mutation Rates in a Steady-State Genetic Algorithm, in [ICEC96], pp.318-323.

[HoBä91] Hoffmeister, F., & Bäck, T.: Genetic Algorithms and Evolution Strategies: Similarities and Differences, in [ScMä91], pp.455-470.

[Juls95] Julstrom, B.A.: What have you done for me lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm, in [Eshe95], pp.81-87.

[Mich94] Michalewicz, Z.: Genetic Algorithms + Datastructures = Evolution Programs, 2nd extendend edition, Springer-Verlag, New York, 1994.

[Rose60] Rosenbrock, H.H.: An Automatic Method for Finding the Greatest or Least Value of a Function, The Computer Journal, Vol.3, No.3, pp.175-184, 1960.

[Scha89] Schaffer, J.D. (editor): Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, 1989.

[ScMä91] Schwefel, H-P., & Männer, R.: Parallel Problem Solving from Nature - PPSN 1 (or Lecture Notes in Computer Science; Vol.496), Springer-Verlag, Berlin, 1991.

[TöZi89] Törn, A., & Zilinskas, A.: Global Optimization (or Lecture Notes in Computer Science; Vol.350), Springer-Verlag, Berlin, 1989.

[vdVaart99] van der Vaart, N.A.L.: : Towards Totally Self-Adjusting Genetic Algorithms: Let Nature Sort it Out, Masters Thesis, Leiden University, 1999.

[WMRD95] Whitley, L.D., Mathias, K.E., Rana, S., & Dzubera, J.: Building Better Test Functions, in [Eshe95], pp.239-246.