A parameter-less genetic algorithm

Georges R. Harik
950 High School Way
Mountain View, CA
gharik@earthlink.net

Fernando G. Lobo
Dept. Environmental Engineering
Universidade Nova de Lisboa
2825 Monte da Caparica, Portugal
fgl@mail.fct.unl.pt

Abstract

From the user's point of view, setting the parameters of a genetic algorithm (GA) is far from a trivial task. Moreover, the user is typically not interested in population sizes, crossover probabilities, selection rates, and other GA technicalities. He is just interested in solving a problem, and what he would really like to do, is to hand-in the problem to a blackbox algorithm, and simply press a start button. This paper explores the development of a GA that fulfills this requirement. It has no parameters whatsoever. The development of the algorithm takes into account several aspects of the theory of GAs, including previous research work on population sizing, the schema theorem, building block mixing, and genetic drift.

1 INTRODUCTION

When Holland (1975) proposed genetic algorithms, he envisioned them as methods that were going to be efficient, easy to use, and applicable to a wide range of problems. But one thing that stands out from the current literature is that genetic algorithms seem to require quite a bit of expertise in order to make them work well for a particular application. The expertise is needed because users are generally clueless on how to decide among the various codings and operators, as well as on deciding on a good set of parameter values for the GA. In the end, instead of being a robust and an easy-to-use method, the genetic algorithm turns out to be a method that needs a lot of tuning and parameter fiddling. This state of affairs is a kind of a paradox and contradicts Holland's original goals.

The decisions that a user must make before applying a GA can be grouped in two categories. The first, is the choice of an appropriate coding and operators. The second, is the choice of appropriate parameter settings. This paper addresses the second category. Not that the first one is unimportant or irrelevant. It is indeed a very important topic that has been subject of extensive research. Specifically, one of the motivations of the work in linkage learning GAs, is to relieve the user from having to come up with special purpose codings and operators. Linkage learning GAs are able to detect important similarities (building blocks) in the encoding, and are able to come up with "smart" operators that are able to efficiently process building blocks (Harik, 1997). The work contained herein has a similar flavor regarding the second category of choices. The outcome is an algorithm that relieves the user from having to set the parameters of the GA. The resulting parameter-less GA makes life easier to users and goes one step closer in the direction of an algorithm that is robust, efficient, and simple to use.

The remainder of the paper is organized as follows. The next section reviews work related to the topic of parameter setting in GAs. Section 3 describes the parameter-less genetic algorithm and is the core of this paper. Section 4 presents computer simulations showing the parameter-less GA in action. Finally, the paper ends by suggesting a couple of extensions to this work.

2 RELATED WORK

This section reviews the most important research efforts for understanding the relationship among the various parameters, how they affect performance, and attempts to eliminate some of them from the GA. There is a substantial amount of research in these areas and in this paper we can only briefly mention a small part of it. For a better exposition, we divide the various works in three classes: empirical studies, facetwise theoretical studies, and parameter adaptation.

2.1 EMPIRICAL STUDIES

One of the first systematic studies to understand the relationship among the GA parameters, was conducted by De Jong (1975). He tried various combinations of parameters on a set of five functions. Based on those experiments, De Jong concluded that for his test functions, the following set of parameters gave good performance: population size of 50-100, crossover probability of 0.6, and mutation probability of 0.001. After De Jong published his dissertation, these parameter settings have been adopted by many researchers and practitioners. They have been used so often, that they are sometimes referred to as "standard" settings. De Jong's work was very important as it gave practical guidelines for subsequent GA applications and research. The downside is that many people have taken his results too seriously and have blindly applied them to solve all kind of problems.

2.2 FACETWISE THEORETICAL STUDIES

Because the dynamics of the GA are so complex, it is hard to study the effect of all the parameters simultaneously. Therefore, many research studies have analyzed the effect of one or two parameters in isolation, and ignored the others. These facetwise studies have proven to be very useful to get insights about the GA dynamics. Among the most relevant ones, are the works on selection alone (Goldberg & Deb, 1991), mutation (Mühlenbein, 1992; Bäck, 1993), population sizing (Goldberg, Deb, & Clark, 1992), (Harik et al., 1997), and control maps (Goldberg, Deb, & Thierens, 1993; Thierens & Goldberg, 1993). The work on population sizing is of special relevance, resulting in models that clearly show that setting the population size in the range 50-100 for all problems is a huge mistake. The work on control maps is also key for the development of the parameter-less GA. A control map gives a region of the parameter space where the GA is expected to work well. We will talk more about these issues later in the paper.

2.3 PARAMETER ADAPTATION

Parameter adaptation has to do with techniques that change or adapt the parameter values as the search progresses. This topic has been investigated since the early days of the evolutionary computation (EC) field. The techniques used include both centralized and decentralized control methods. Centralized methods change the parameter values based on a central learning rule (Cavicchio, 1970; Davis, 1989; Smith & Smuda, 1995). Decentralized methods have no central control and have been used more often in a GA-related class of algorithms called *Evolution Strategies* (ES) (Bäck & Schwefel, 1995). The idea behind decentralized methods is to encode the parameters (typically crossover and mutation rates) in the individual strings themselves. As a consequence, the parameters are also subject to the rules of evolution. Yet another approach for parameter adaptation is the work on Meta-GAs. Here, the idea is to run a higher level GA that searches for a good set of parameters for a lower level GA. The higher level GA, or Meta-GA, runs on a population whose individuals are encodings of parameter settings, while the lower level GA is a regular GA that uses the settings found by the higher-level GA (Mercer & Sampson, 1978; Grefenstette, 1986).

Most of the work in parameter adaptation have focused on adapting mutation and crossover rates, but there has been very little work on adaptive population sizing schemes. The work of Smith and Smuda (1995) is an exception. We will address it in more detail in section 3.2.

3 DEVELOPMENT OF A PARAMETER-LESS GA

In the previous sections we have explained the motivation for removing the parameters of the genetic algorithm. Now we are ready to present you the parameter-less GA. Our work ignores mutation for the time being. This doesn't mean that mutation is unimportant. There has been extensive debates in the evolutionary computation community regarding the usefulness of crossover versus mutation and vice-versa. We recognize that mutation can be quite helpful in certain situations and that it shouldn't be discarded lightly. But in this work we ignore it for the sake of simplicity. We will revisit the topic on mutation at the end of the paper. For now, you have to be satisfied with a parameter-less crossover-based GA.

There are three parameters that affect the performance of a crossover-based GA: population size, selection rate, and crossover probability. In the remainder of the section we explain how we can eliminate these parameters. The exposition proceeds in two steps. First, we get rid of selection rate and crossover probability. Next, we get rid of population sizing.

3.1 GETTING RID OF SELECTION RATE AND CROSSOVER PROBABILITY

The selection rate and crossover probability can be eliminated in a single shot if one has a proper understanding of the roles of these two parameters. The selection rate s allows the user to control the amount of bias towards better individuals. The crossover probability p_c allows the user to control the amount of recombination or mixing. Together the two parameters work for the same purpose, which is to ensure that building blocks grow from generation to generation. A closer look at the schema theorem reveals that the survival probability of a schema can be made to increase by either raising the selection rate or lowering the crossover probability. Previous research (Goldberg, Deb, & Thierens, 1993) have shown that the GA is quite robust regarding the settings of these two parameters. Goldberg, Deb, and Thierens observed that when the building blocks are compact[1], the GA works well for a wide range of combinations of p_c and s. The important thing is to respect the schema theorem and not fall in the extreme cases of very low and very high selection pressures. At very low selection pressures the GA is not able to discriminate between the good and bad individuals. At very high selection pressures the GA only pays attention to the very best individuals, resulting in an immediate loss of diversity and little recombination to be done. The two extreme cases are easily excluded by setting the selection rate to a value that is neither too low nor too high. Apart from that, the GA just needs to obey the schema theorem by making sure that the growth ratio of building blocks is greater than 1. Let $\phi(H,t)$ be the effect of the selection operator on schema H at generation t, and $\epsilon(H,t)$ be the disruption factor on schema H due to the crossover operator. Then the overall net growth ratio on schema H at generation t is given by:

$$\phi(H,t)[1-\epsilon(H,t)]$$

The above expression is nothing but a simplification of the schema theorem. Under the conservative hypothesis that a schema is destroyed during the crossover operator, and substituting s and p_c in the formula above, we obtain that the growth ratio of a schema is given by the expression:

$$s(1-p_c)$$

Thus, setting $s=4$ and $p_c=0.5$, gives a net growth factor of 2, and ensures that the necessary building blocks will grow. Whether they will be able to mix in a single individual or not is now a matter of having an adequate population size. If the building blocks are compact, the population sizing requirements will be quite reasonable (Goldberg, Deb, & Clark, 1992), (Harik et al., 1997). If they are not compact, then, unless the GA is able to learn linkage on the fly, the population sizing requirements will be extremely large (Thierens & Goldberg, 1993). In either case, we can get rid of selection rate and crossover probability by setting $s=4$ and $p_c=0.5$ for all problems.

[1] A building blocks is compact when its genes are located close to each other in the chromosome string

3.2 GETTING RID OF POPULATION SIZING

The population size is a critical parameter in a GA. Too small and the GA will converge to sub-optimal solutions. Too large and the GA will spend unnecessary computational resources. Theoretical models have addressed the topic of population sizing in GAs (Goldberg, Deb, & Clark, 1992), (Harik et al., 1997). The essence of those models is the recognition that the GA can make mistakes when deciding between a building block and one of its competitors. By increasing the size of the population, the GA can get a better sampling of building blocks and reduce the error in decision-making. In simple words, the models dictate that the population size has to be large enough so that the GA can correctly decide, in a statistical sense, between a building block and its competitors. The result is an equation that says that the population size should be proportional to the problem length, and to building blocks's signal-to-noise ratios. Although accurate, those models are difficult to apply in practice because they rely on several assumptions that may not hold for real-world problems. For example, in order to apply the population sizing equation, the user needs to know or estimate, the maximum level of deception in a problem, and the selective advantage (signal) of a building block over its most tough competitor. This information is usually unknown and is also hard to estimate. Moreover, the equation is only accurate when there is a proper mixing of building blocks. Due to these reasons, it becomes difficult to apply the equation in practice. If it was easy to apply it, then the whole process could be automated in the algorithm itself. Smith and Smuda (1995) did such an attempt. In order to that, the authors required the user to specify a desired accuracy level for selection loss (something equivalent to the building block's signal in Goldberg et al.'s equation). Then, they suggested an algorithm that does online estimates of schema fitness variances, and sizes the population so that the selection loss approximates the user's specified target loss. However, the resulting algorithm had a few limitations. First, it is hard to relate the user's specified selection target loss with the actual accuracy of the GA in solving the problem. Second, the estimation of schema fitness variances is very noisy because the GA samples many partitions simultaneously. Third, the population sizing theory assumes that the building block mixing is going to be nearly perfect, which may not occur in practice. Nonetheless, the work of Smith and Smuda is a significant one, especially because there has been very few studies that have attempted to automatically adapt the population size.

So far we have been looking at the difficulties that exist in order to apply the population sizing theory in practice. Let's step back for a moment and observe what a user typically does to solve a problem with a GA. Most likely, he starts by trying a small population size. Then, he might try a larger one. In the end, perhaps he has tried 5 to 10 different population sizes to have a feeling of how the problem responds to different population settings. This situation is certainly familiar to many GA users around the world, ourselves included. But why should every user need to do this kind of experimentation for every single problem? Why not have the algorithm do the experimentation automatically? That's more or less what we are about to propose here. Next, we present two approaches for getting rid of population sizing. The first is simpler, but has a major drawback. The second comes as a logical consequence of the first one, and constitutes an effective mechanism for eliminating the population sizing parameter once and for all.

3.2.1 First attempt: double the population size at convergence.

This approach consists of iteratively running the GA with different population sizes. Starting with a small population size, say 4, let the GA run until it converges.[2] Following that, the GA fires a new population, twice as large as the previous one, and again let it converge. And the process repeats *ad eternum*. Each time the population converges, the population size is doubled. The GA works forever and only stops if the computer runs out of memory, or the user is happy with the solution quality, and presses a stop button. This technique, although simplistic, is powerful. It roughly mimics what the user typically does in an ad-hoc way, except this time, the GA is doing it in a systematic way. By doubling the population size each time the population converges, there is a guarantee that the time needed by the parameter-less GA in order to reach a certain solution quality, will be within a factor of 2 of the time needed by a GA that starts with an optimal population size. Moreover, if the user doesn't want to wait that long, and presses the stop button early on the run, the parameter-less GA does probably better than a GA with an optimal population size.

This technique has a drawback that is not immediately obvious. The occurrence of genetic drift may lead to long convergence times. Genetic drift occurs when there is not enough pressure to discriminate between two or more distinct solutions. The simplest way to illustrate it is with a one-bit problem where both individual 0 and individual 1 have the same fitness. Consider a population of N individuals, half of them are like individual 0, and half are like individual 1. When a GA runs on such a population, there's no selection pressure to discriminate between the two different solutions, and eventually, the population will converge to all zeros or all ones. But the whole process looks like a random walk, and the waiting times until convergence can be quite long. This roaming around of the population is called *drift*. The same process can occur in our proposed parameter-less GA. The consequences are that the waiting times before the population size is doubled can be quite long and unacceptable. One solution to avoid these long waiting times is to detect if a population is drifting, and in the affirmative case, fire a new population twice as large, even before the old one converges. However, drift detection is a difficult task. One could think of stopping the population from running if the fitness variance of the population members falls within a certain threshold ϵ, but such an approach introduces two problems. First, there's the introduction of a parameter, which goes against the whole philosophy of this work. Second, many real-world problems have noisy fitness functions. On such problems, the threshold ϵ might never be reached. Fortunately, there's another way to get around the drift problem, and without the need of additional parameters.

3.2.2 Second attempt: establish a race among multiple populations.

An alternative approach for tackling the drift problem, consists of running multiple populations simultaneously. The idea is to establish a race among populations of various sizes. The parameter-less GA gives an advantage to the smaller populations by giving them more function evaluations. Consequently, the smaller populations have a chance to converge faster than the larger ones. That is, smaller populations get a head start at the beginning, but if they start to drift too much, they will be catched up by a larger population. When that happens, the smaller populations are destroyed. Specifically, if at any point in time, a larger population has an average fitness greater than that of a smaller population, then the parameter-less GA gets rid of the smaller population. The rationale for doing this is as follows. If a larger population has a higher fitness than a smaller population, then there is no point in continuing running the smaller population, since it is very unlikely that the smaller one will produce a fitter individual than the larger one. The drift problem is partially eliminated because if a population starts drifting, it will be catched up by a larger population. The coordination of this array of populations seems complex, but can be easily implemented with a counter as illustrated by table 1.

At each time step, a counter of base 4 is incremented, and the position of the most significant digit that changed during the increment operation is noted. That position indicates which population should be ran. In the example above, the algorithm initially runs population 1 for 4

[2] Convergence means that all the individuals have the same genotype. Notice that this eventually happens because mutation is turned off.

Table 1: Mechanics of the parameter-less GA.

Counter base 4	Action
0	run 1 generation of population 1
1	run 1 generation of population 1
2	run 1 generation of population 1
3	run 1 generation of population 1
10	run 1 generation of population 2
11	run 1 generation of population 1
12	run 1 generation of population 1
13	run 1 generation of population 1
20	run 1 generation of population 2
21	run 1 generation of population 1
22	run 1 generation of population 1
23	run 1 generation of population 1
30	run 1 generation of population 2
31	run 1 generation of population 1
32	run 1 generation of population 1
33	run 1 generation of population 1
100	run 1 generation of population 3
101	run 1 generation of population 1
⋮	⋮

generations, then it runs population 2 for 1 generation, then population 1 for 3 more generations, then population 2 for 1 generation, and so on as illustrated in table 1. Overall, population i is allowed to run 4 times more generations than population $i+1$. Just like in the description of our first attempt, each new population that gets fired is twice the size of the previous population. Taking into account both the number of generations and the population sizes, we observe that population i is allowed to spend twice the number of function evaluations of population $i+1$.

Eventually, a population converges (most likely population 1). At that point the algorithm removes it, and resets the counter. Likewise, if the average fitness of a population is less than the average fitness of a larger population, then the smaller population is removed, and the counter is reset. After some time, the typical state of the parameter-less GA might look like the one shown in table 2. In the example, the parameter-less GA is currently running with 3 different population sizes. The smaller one has size 256 and is at generation 30. The next population has size 512 and is only at generation 6. The larger one has size 1024 and is still at generation 1.

Table 2: Typical state of parameter-less GA.

population index	population size	current generation	average fitness
1	256	30	17.6
2	512	6	11.8
3	1024	1	7.8

4 EXPERIMENTAL RESULTS

This section presents computer simulations on three test problems, and compares the performance of the parameter-less GA with a regular GA with "optimal" parameter settings for those problems. Both GAs are implemented as simple GAs. By simple GA, we mean a GA that uses a fixed coding and non-adaptive operators. Our simple GA implementation is generational, uses tournament selection without replacement, uniform crossover, and no mutation. For each problem, 20 independent runs were performed in order to get results with statistical significance. In order to compare the parameter-less GA with the regular GA, we run both algorithms until a specified target solution is achieved, and record the number of fitness function evaluations needed to do so.

The three test problems are carefully chosen to illustrate specific aspects of the parameter-less GA. The first problem is the onemax problem, also known as the counting ones problem. This is an easy problem for the GA, and we can use the existing population sizing theory in order to compare the parameter-less GA with a GA with "optimal" population size. The second problem, is the noisy onemax problem. We use this problem to illustrate the need of having multiple populations running simultaneously in order to overcome drift. The third problem is a bounded deceptive problem where the user has no knowledge about the location of the deceptive sub-problems or building blocks. For such a problem, the simple GA is unable to efficiently mix the building blocks, and thus the existing population sizing theory does not hold.

4.1 ONEMAX

The onemax function simply returns the number of ones of an individual string. For testing purposes, a string length of 100 bits is used. The target solution is the string with all ones. The parameters of the regular GA are: tournament size 2, probability of crossover equal to 1, and population size of 100. These parameters were chosen because they are nearly optimal for this problem under a crossover-based GA. The population size of 100 was chosen based on the population sizing equation (Harik et al., 1997) shown below:

$$N = \frac{-2^k\sqrt{\pi m \sigma_{BB}^2}}{2d}\ln\alpha$$

The population size N depends on the building block size k, the number of building blocks m, the building block's fitness variance σ_{BB}^2, and the building block's fitness signal d. For a 100 bit onemax problem, these values are $k = 1$, $m = 100$, $\sigma_{BB}^2 = 0.25$, and $d = 1$. Plugging these values in the equation, we get:

$$N = -8.86\ln\alpha$$

where α is the probability that the GA makes a mistake on a building block. Thus, the GA is expected to correctly solve a proportion of $1-\alpha$ of the building blocks. In our case, we would like the GA to get the optimal solution in all the 20 runs. Theoretical, this can only be guaranteed if $\alpha = 0$, which would imply an infinite population size. For practical purposes, we use a population size of 100, which corresponds to a very small error probability (≈ 0.00001).

Figure 1 compares the regular GA with the parameter-less GA. The regular GA takes on average 2500 function evaluations in order to reach an optimal solution, while the parameter-less GA takes on average 7400 function evaluations. As expected, the GA with optimal parameter settings can reach the optimal solution faster than the parameter-less GA. But the parameter-less GA does not bad at all. Without using any knowledge about the problem, the parameter-less GA is able to reach the optimal solution in not more than 3 times the time needed by the optimal GA. The population sizing scheme is perhaps responsible for a factor of 2. The rest is due to the selection rate $s = 4$ and the crossover probability $p_c = 0.5$ used by the parameter-less GA. On an easy problem such as the onemax, a higher crossover rate and a less aggressive selection scheme, would achieve a higher solution quality faster. But this is a price that must be paid in order to have robust settings. That is, to have parameter settings that work more or less well on both easy and difficult problems. As an aside, notice that we are not doing any clever tricks counting the number of function evaluations. We are simply counting them as the product of population size by number of generations. In a careful implementation, the parameter-less GA could actually spend only half of that number simply by not re-evaluating the population members that don't undergo crossover.

4.2 NOISY ONEMAX

Here we want to illustrate the need for having multiple populations running simultaneously in the parameter-less GA. To do so, we test it on a noisy onemax problem.

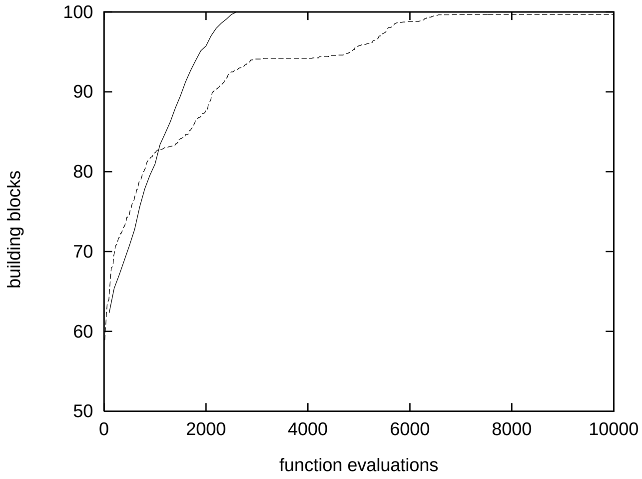

Figure 1: A 100-bit onemax problem. In this, as well as in the other figures in this paper, both lines indicate the best solution found so far by each algorithm averaged over 20 runs. The solid line is for the regular GA. The dashed line is for the parameter-less GA.

In this problem, the fitness f' of an individual is given by:

$$f' = f + noise$$

where f is the fitness of the problem in the absence of noise, and *noise* is a noise component. For practical purposes, the noise component can be simulation by tossing a Gaussian random variable with mean 0 and variance σ^2. We use a 100-bit string length, and a noise variance $\sigma^2 = 1000$. This value corresponds to a large amount of noise. Miller (1997) studied the effects of noise in the population sizing requirements of the GA, and extended Harik et al.'s equation to account for it. The resulting equation is:

$$N = \frac{-2^k\sqrt{\pi}\sqrt{\sigma_f^2 + \sigma_{noise}^2}}{2d}\ln\alpha$$

For the 100-bit onemax problem, the fitness variance without noise is $\sigma_f^2 = 25$. For a noise level of $\sigma_{noise}^2 = 1000$, we obtain a population sizing of:

$$N = -56.7\ln\alpha$$

For the same level of accuracy that was used before ($\alpha = 0.00001$), we get a population size of 650. Figure 2 shows a comparison of the regular GA with optimal parameter settings ($N = 650$, $s = 2$, $p_c = 1$) with the parameter-less GA. The regular GA takes on average 96000 function evaluations in order to discover an individual with 100 building blocks. To do the same thing, the parameter-less GA takes on average 197000 function evaluations, a factor of 2.

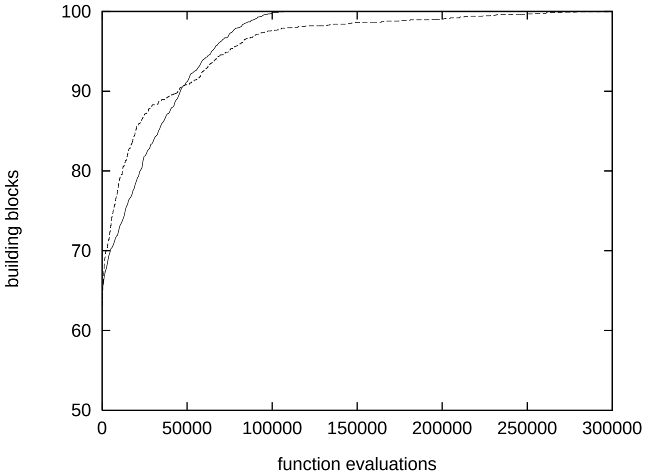

Figure 2: A 100-bit noisy onemax problem.

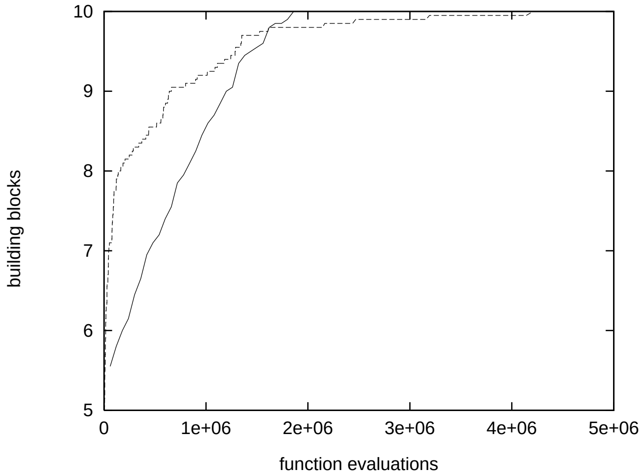

Figure 3: 10 copies of a 4-bit trap function.

Tracing the parameter-less GA, we observed that the smaller populations were not being ran until convergence. Instead, they were often catched up by larger populations. Due to the large amount of noise, there is only a very tiny selection pressure, and the population simply drifts. Notice that drift is not completely eliminated. The populations still drift, but they don't do it too much because they are not ran until convergence.

4.3 BOUNDED DECEPTIVE

The last test case illustrates a problem that is known to be difficult for a simple GA. The problem is the concatenation of 10 copies of a 4-bit trap function. A trap function is a function of unitation u, the number of ones in a 4-bit sub-string. It is defined as follows:

$$f(u) = \begin{cases} 3 - u, & \text{if } u < 4; \\ 4, & \text{if } u = 4. \end{cases}$$

The overall fitness function is the sum of the 10 independent sub-functions. Without previous knowledge about the location of each of the 10 sub-functions or building blocks, it is very hard for the simple GA to combine building blocks from different individuals and assemble them in a single individual. In order to do that, the simple GA needs to obey the schema theorem, and in addition, it requires a very large population size (Thierens & Goldberg, 1993). For this case, the existing population sizing models are unable to predict accurate population sizes, because the assumption that the building blocks mix well does not hold. Our desired target solution is the string with all 10 building blocks solved correctly. In order to obey the schema theorem, the regular GA uses a selection rate of $s = 4$ and a crossover probability $p_c = 0.5$. The minimum population size needed in order to solve all 20 runs to optimality was found empirically to be 60000. Figure 3 compares the two GAs. The regular GA took on average 1.5 million function evaluations to reach the target solution, and the parameter-less GA took about the same number.

5 EXTENSIONS

A couple of extensions can be done on top of this work. One is the integration of mutation. In many problems, a mutation-based GA will outperform a crossover-based GA. More important, many problems will benefit from a combination of both. The question is how to combine them in an efficient way. There is empirical and theoretical evidence in the GA community, that mutation works best with small population sizes, while crossover works best with large population sizes. It seems that a crossover-based GA and a mutation-based GA need totally different strategies. The question of how to integrate mutation with crossover is a difficult one, and is surely an area that needs additional research.

Another extension to this work, is to experiment with other kind of GAs. It is important to mention that the parameter-less GA that we have just proposed is not tied up to a specific implementation. It is valid for any kind of crossover-based GA. Thus, it is possible to have a parameter-less simple GA, a parameter-less linkage-learning GA, and so on. In this paper, we restricted ourselves to a parameter-less simple GA. That was done on purpose in order to make the exposition clear. But we have also conducted experiments with a parameter-less linkage learning GA. The result is a very powerful algorithm that not only relieves the user from having to set the GA parameters, it also relieves the user from having to come up with special purpose codings and operators. Those results will be reported soon.

6 SUMMARY AND CONCLUSIONS

This paper presented a practical approach that eliminates the need of parameters in a crossover-based GA. We illustrated the operation and described how the parameter-less GA can be implemented. The parameter-less GA uses a crossover probability $Pc = 0.5$, and a selection rate $s = 4$ in order to obey the schema theorem, and not fall in the extreme cases of very high and very low selection pressures. At first, fixing the selection rate and the crossover probability for all problems, might look like an approach similar to the one used by those who adopt the so-called "standard" parameter settings. But in this case, the parameter settings are backed up by sound theoretical work, and constitute robust settings. Of course, it is possible to obtain better performance in some problems, by tweaking both s and p_c. But with this work, we are not interested in peak performance. What we are really interested is in robustness and simplicity of use. Regarding the population size, the parameter-less GA tackles it by running multiple populations in a cascade-like manner, and getting rid of the ones that converge, and the ones that drift. The parameter-less GA is a bit slower than a GA that starts with "optimal" parameter settings, but the truth is that nobody knows what are the "optimal" parameter settings for an arbitrary real-world problem. The technique presented here just requires the user to press a start button. The algorithm runs forever, and does the the best that it can until the user is happy with the solution quality, or has waited long enough and decides to press a stop button.

Users of GA technology don't have to be necessary GA specialists in order to use them. To disseminate the usage of these methods, the algorithms have to be accurate, reliable, efficient, and simple to use. Up to this date, the issue on parameter settings has been very confusing for users. The parameter-less GA removes a great part of this confusion. It makes life easier to users, and we strongly believe that it will contribute for more successful applications of GAs. As far as theory is concerned, the parameter-less GA is based on solid foundations, and it helps to narrow the existing gap between GA theory and GA practice.

References

Bäck, T. (1993). Optimal mutation rates in genetic search. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 2–8). San Mateo, CA: Morgan Kaufmann.

Bäck, T., & Schwefel, H.-P. (1995). Evolution strategies I: Variants and their computational implementation. In Winter, G., Périaux, J., Galán, M., & Cuesta, P. (Eds.), *Genetic Algorithms in Engineering and Computer Science* (Chapter 6, pp. 111–126). Chichester: John Wiley and Sons.

Cavicchio, Jr., D. J. (1970). *Adaptive search using simulated evolution.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 61–69). San Mateo, CA: Morgan Kaufmann.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, *1*, 69–93. (Also TCGA Report 90007).

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, *32*(1), 10–16.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. In Sage, A. P. (Ed.), *IEEE Transactions on Systems, Man, and Cybernetics*, Volume SMC–16(1) (pp. 122–128). New York: IEEE.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Doctoral dissertation, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Bäck, T. (Ed.), *Proceedings of the Fourth International Conference on Evolutionary Computation* (pp. 7–12). New York: IEEE Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Mercer, R. E., & Sampson, J. R. (1978). Adaptive search using a reproductive meta-plan. *Kybernetes*, *7*, 215–228.

Miller, B. L. (1997). *Noise, sampling, and efficient genetic algorithms.* Doctoral dissertation, Urbana. Also IlliGAL Report No. 97001.

Mühlenbein, H. (1992). How genetic algorithms really work: I. Mutation and Hillclimbing. *Parallel Problem Solving from Nature- PPSN II*, 15–25.

Smith, R. E., & Smuda, E. (1995). Adaptively resizing populations: Algorithm, analysis, and first results. *Complex Systems*, *9*, 47–72.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.