Parameter-less Genetic Algorithm: A Worst-case Time and Space Complexity Analysis

Martin Pelikan and Fernando Lobo

IlliGAL Report No. 99014
March 1999

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

Parameter-less Genetic Algorithm: A Worst-case Time and Space Complexity Analysis

Martin Pelikan and Fernando Lobo
Illinois Genetic Algorithms Laboratory
104 S. Mathews Avenue, Urbana, IL 61801
University of Illinois at Urbana-Champaign
Phone/FAX: (217) 333-2346, (217) 244-5705
{pelikan,lobo}@illigal.ge.uiuc.edu

Abstract

In this paper, the worst-case analysis of the time and space complexity of the parameter-less genetic algorithm versus the genetic algorithm with an optimal population size is provided and the results of the analysis are discussed. Since the assumptions in order for the analysis to be correct are very weak, the result is applicable to a wide range of problems. Various configurations of the parameter-less genetic algorithm are considered and the results of their time and space complexity are compared.

1 Introduction

A parameter-less genetic algorithm (Harik & Lobo, 1999) is an alternative to a common trial-and-error method of tweaking the values of the parameters of the genetic algorithm in order to find a set-up to accurately and reliably solve a given problem. The algorithm manages a number of independent runs of the genetic algorithm with different population sizes with the remaining parameters set to fixed values according to the theory of genetic algorithms' control maps introduced by Goldberg, Deb, and Thierens (1993) in order to constitute robust settings. So far, however, no theoretical analysis about the relation between the time to convergence or the space required by the parameter-less genetic algorithm and the genetic algorithm with an optimal population size has been done.

The purpose of this paper is to provide a worst-case analysis of the recently proposed parameter-less genetic algorithm. Time to convergence as a function of a time required by the genetic algorithm with an optimal population size is computed. Moreover, space complexity analysis of the parameter-less genetic algorithm is performed. Various configurations of the parameter-less genetic algorithm are analyzed and the results of the analysis are discussed.

The paper starts by providing the basic background and motivation to the proposal of the parameter-less genetic algorithm. Section 3 describes the principle of the algorithm. In Section 4, a worst-case analysis of the method is provided. The results of the analysis are discussed in Section 5. The paper is concluded in Section 6.

2 Background

The genetic algorithm (GA) (Goldberg, 1989) is an optimization method praised for it is easy to use and applicable on a wide range of problems without any prior problem-specific knowledge. However, to make the genetic algorithm work well, the user must specify a number of parameters such as the population size, selection pressure, crossover rate, and mutation probability. Using the schema theorem (Holland, 1975), it is possible to set the crossover rate and selection pressure so that the algorithm achieves good results on a number of problems. The crucial problem is how to size the population in order to get a solution of a required quality. Being unaware of optimal building blocks in a problem, the population sizing theory (Harik, Cantú-Paz, Goldberg, & Miller, 1997) is not directly applicable and the user is usually left to a trial-and-error approach of enlarging the population manually until the results are satisfactory.

The parameter-less genetic algorithm uses results of the schema theorem (Holland, 1975; Goldberg, 1989) and various facet-wise theoretical studies of genetic algorithms (Goldberg, Deb, & Thierens, 1993; Harik, Cantú-Paz, Goldberg, & Miller, 1997) in order to eliminate the GA parameters. It sets all parameters except the population size to fixed, yet reasonable values, and manages multiple independent GA runs with different population sizes.

3 Parameter-less Genetic Algorithm

In the parameter-less genetic algorithm no mutation is used. This does not mean that mutation is not important. However, for the sake of simplicity, in the parameter-less genetic algorithm mutation is ignored. The next few sections discuss how to discard the remaining parameters of the GA, i.e. the selection pressure, crossover rate, and the population size.

3.1 Eliminating the Selection Pressure and Crossover Rate

Selection pressure controls the bias of the algorithm toward high-quality solutions. The stronger the selection pressure, the more pressure is put on the quality of solutions during selection. The crossover rate controls the amount of mixing of promising solutions in the GA. The higher the crossover rate, the more the promising solutions are mixed.

The selection pressure and crossover rate are correlated. The selection has to be strong enough to ensure a proper reproduction of high-quality building blocks. On the other hand, it must not be too strong to preserve the diversity in the population and give the crossover enough time to combine the promising solutions before the population converges. The crossover rate has to be large enough in order to ensure a proper mixing of promising building blocks with respect to the current selection pressure. On the other hand, it must not be too high because a "blind" crossover often breaks long building blocks, and with a weak selection the promising solutions might be lost.

In the parameter-less genetic algorithm, the selection pressure and crossover rate are set according to a simplification of the schema theorem in order to ensure the growth of promising building blocks. The selection pressure s is set to $s = 4$ (the best solution in a population gets 4 copies in the selected set of the same size as the population) and the crossover rate p_c is set to $p_c = 0.5$ (half of the pairs of solutions in the selected set undergo crossover). In this way, the solutions better than a median of the population get at least one full copy in the offspring on average and still about a half of the offspring will be created by combining the selected promising solutions by crossover. Any crossover method can be used.

3.2 Eliminating the Population Size

When the user does not know an optimal population size that guarantees that the GA finds the solution of a particular quality, he usually tries to run the algorithm for a few different population sizes. When the results achieved with bigger populations are similar to the results with smaller ones, there are no more available computational resources to spend, or the user is satisfied with the found solution, he takes the best solution found so far.

The parameter-less genetic algorithm is based on a similar principle. We will describe two possible approaches. The first approach is more or less an automated version of the user's typical way of iteratively enlarging the population. However, in the parameter-less genetic algorithm, another approach is used in order to eliminate a major drawback of the first, simpler approach.

First, let us denote the GA runs to be performed by GA. The runs will be indexed by non-negative integer numbers, starting with a run GA_0. The population size used in the run GA_i will be denoted by N_i. The first run uses a small population size, e.g. $N_0 = 4$. Population sizes for the remaining runs are set inductively by

$$N_{i+1} = 2N_i, \tag{1}$$

i.e. each run uses a population twice as large as the population used by the preceding run. Thus,

$$N_i = 2^i N_0. \tag{2}$$

The following two sections describe two approaches to simulating the runs GA_i with the properties described above.

3.2.1 The First (Iterative) Approach

The first approach (Harik & Lobo, 1999) is based on performing the runs iteratively, starting with the run GA_0. When the current run GA_i converges, the next run GA_{i+1} with a population size twice as big as the previous one is started. The process repeats until the user is satisfied with the solution the algorithm has found or the available computational resources have already been spent.

However, with noisy functions or functions with multiple optima it is far from a trivial task to design effective termination criteria. For many problems, there is simply no way of determining the "right moment" to terminate a particular run, and the run can either wander in a search space for a long time without any qualitative improvement or lose the information it has acquired during the computation because of having been terminated too soon. The success of iteratively enlarging the population is determined by the choice of termination criteria.

3.2.2 The Second (Concurrent) Approach

The second approach (Harik & Lobo, 1999) eliminates a major drawback of the iterative approach described in the previous section by managing a race among multiple populations. The parameter-less genetic algorithm maintains an infinite number of independent GA runs simultaneously.

Runs with larger populations require proportionally more fitness evaluations to process each generation than the runs with smaller populations, and therefore it would be "unfair" to process each of the runs at the same speed with respect to the number of performed generations. Moreover, by running an infinite number of runs at the same speed an infinite loop would be inevitable. The problem can be evaded by allowing each run to use the same number of fitness evaluations

throughout the entire computation. The GA_i with a population of size N_i is allowed to perform twice as many generations as the GA_{i+1} with $N_{i+1} = 2N_i$. Each run gets to perform the next generation when the runs with smaller population sizes have used as many function evaluations as this run needs to perform the next generation. In this fashion, all runs get the same number of fitness evaluations.

In the original parameter-less genetic algorithm, runs with smaller populations get even more fitness evaluations than the runs with larger populations. The GA_i gets twice more fitness evaluations than the GA_{i+1}. In this paper, we will analyze a general case where the run GA_{i+1} gets linearly proportional number of fitness evaluations with respect to the number of evaluations given to the preceding run GA_i.

A particular run is deleted when it converges or when the average fitness in the current population of this run is lower than the average fitness in the current population of some run with a larger population, and therefore it is unlikely that the run with a smaller population succeeds in getting a better result than the run with a bigger population. After deleting the run, all the runs with smaller populations are also deleted because it is again unlikely that they get a result any better than the runs with bigger populations. The indices of the remaining runs are decreased so that the run with the smallest population size will be again indexed as GA_0. By maintaining the runs by the mechanism described above, if the solution can be found with a population big enough, the parameter-less genetic algorithm will always find this solution. The described technique also ensures that in spite of virtually managing an infinite number of runs, at each point in time only a finite number of runs are maintained.

Counter (base $m = 2$)	Activated GA
0	
$\underline{1}$	GA_0
$\underline{1}0$	GA_1
$1\underline{1}$	GA_0
$\underline{1}00$	GA_2
$10\underline{1}$	GA_0
$1\underline{1}0$	GA_1
$11\underline{1}$	GA_0
$\underline{1}000$	GA_3
$\vdots$	$\vdots$
$\underline{1}0000$	GA_4
$1000\underline{1}$	GA_0
$\vdots$	$\vdots$

Table 1: Example run of the parameter-less genetic algorithm.

To implement the above method, an m-ary counter, where m is a positive integer greater than one, can be used. At the beginning, the counter is set to zero. Each iteration the counter is incremented by one and the position of the most significant bit that has changed in the counter determines the index of the run that will be allowed to perform one generation this iteration. The index is given by the position of the changed bit according to the ordering of bits from the least to the most significant (from right to left). In this fashion, the GA_i is allowed to run for m times

more generations than the GA_{i+1}. The larger the m, the closer the race resembles the simple iterative approach. For $m = \infty$, the simultaneous approach is equivalent to the iterative one. If the GA_i converges or its average fitness is lower than the average fitness of the run GA_{i+1}, this run along with the runs with smaller population sizes, i.e. the runs GA_j for $j \leq i$, are deleted and the indexes of the runs are decreased so that the remaining run with the smallest population (i.e. GA_{i+1}) is indexed as GA_0 from now on. The counter is shifted to the right so that the positions of bits corresponding to the runs and the index of the runs are consistent. An example of a few steps of incrementing the counter with $m = 2$ (i.e., a binary counter) is given in Figure 1. The bit on a position determining the number of the activated run (the run which is allowed to perform a generation) is underlined.

4 Worst-case Analysis

This section provides an analysis of the relation between the worst case with the parameter-less genetic algorithm and the GA with an optimal population size. In the worst case, only the runs with a population larger or equal than N_{opt} can find the solution of a required quality (e.g. the solution with at least 90% of correct building blocks with probability 99%), and all the runs that have smaller populations sizes never converge nor are ever overtaken by the runs with larger populations.

The section starts by describing the assumptions made in order to perform the analysis. Thereafter, the worst case with the parameter-less genetic algorithm is described and its analysis is performed.

4.1 Assumptions

Our first assumption is that the genetic algorithm with the selection pressure and crossover rate, as set in the parameter-less genetic algorithm, finds the solution of a required quality (further referred to as *the solution*) with a certain probability with a population larger or equal than some critical population size. This assumption is reasonable. With bigger populations, building-block sampling and decision making is more accurate. If the quality of a found solution does not grow monotonously with the population size, the GA is probably not the best way to approach this problem. We will call the critical population size an optimal one and denote it by N_{opt}.

The average number of generations it takes the GA to find the solution with the optimal population size N_{opt} will be denoted by G_{opt}. When at each generation all solutions have to be evaluated, the number of fitness evaluations until the solution is reached with the optimal population size N_{opt}, denoted by E_{opt}, is given by

$$E_{opt} = G_{opt} N_{opt}. \tag{3}$$

Our second assumption is that with a population size $N \geq N_{opt}$ that is at most twice as big as N_{opt}, the genetic algorithm needs at most $2G_{opt}$ generations to reach the solution on average, i.e.

$$\forall N : N_{opt} \leq N < 2N_{opt} \rightarrow G \leq 2G_{opt},$$

where G is the average number of generations until the GA with a population size N finds the solution. The above two assumptions are in fact weaker than the existing genetic algorithms theory allows them to be. However, they are sufficient in order to make a correct, simple, and very useful worst-case analysis.

4.2 What is the worst case?

Since $N_0 > 0$ and the population size for each of the remaining runs GA_i is given by $N_i = 2^i N_0$, it is easy to see that there exists a minimal k for which the population size in the run GA_k is sufficient in order to find the solution, i.e.

$$N_k \geq N_{opt} > N_{k-1}. \tag{4}$$

By plugging Equation 1 into the right-hand side of the last equation, we get

$$N_k < 2N_{opt} \tag{5}$$

By substituting Equation 2 into Equation 4, we also get

$$2^{k-1}N_0 < N_{opt} \leq 2^k N_0$$

Thus, we can bound the k as follows:

$$k = \left\lceil \log_2 \frac{N_{opt}}{N_0} \right\rceil \leq 1 + \left\lceil \log_2 \frac{N_{opt}}{N_0} \right\rceil \leq 2 + \log_2 \frac{N_{opt}}{N_0} \tag{6}$$

By using the second assumption and Equation 5, we can bound the expected number of generations until convergence of the GA_k, denoted by G_k, by

$$G_k \leq 2G_{opt}. \tag{7}$$

In the worst case, all runs but the GA_k do not reach the solution we are looking for and all computational resources except the ones spent by the GA_k are wasted. The number of generations performed by the run GA_i for $i < k$ in case of success of the GA_k, denoted by G_i, is given by

$$G_i = G_k m^{k-i}. \tag{8}$$

The last equation is valid because we assume that the run GA_k has performed the last generation that has been performed. For $i > k$, we get

$$G_i \leq \frac{G_k}{m^{i-k}}. \tag{9}$$

4.3 Time Complexity Analysis of the Worst Case

In this section, we compute the number of fitness evaluations needed by the parameter-less genetic algorithm as a function of the number of fitness evaluations needed by the GA with an optimal population size N_{opt}. We classify the fitness evaluations the parameter-less genetic algorithm performs into three classes. The first class are the function evaluations performed by the GA_k. The second class are the evaluations performed by the runs with the populations smaller than N_k. The remaining fitness evaluations are classified into the third class. The overall number of fitness evaluations, denoted by E, is given by the sum of the numbers of fitness evaluations in each of the three classes (denoted by E_A, E_B, and E_C, respectively).

4.3.1 E_A: Number of evaluations done by the GA_k

The run GA_k with a population size N_k finds the solution after performing G_k generations on average. When all individuals have to be evaluated each generation, the number of fitness evaluations performed by this algorithm, denoted by E_k, can be computed as

$$E_k = G_k N_k. \tag{10}$$

By applying the equations 7 and 5 to the last equation, we can bound the number of fitness evaluations from the first class, denoted by E_A, as follows:

$$E_A = E_k < 4E_{opt} \tag{11}$$

Thus,

$$E_A = O\left(E_{opt}\right) \tag{12}$$

4.3.2 E_B: Number of evaluations done by the runs with smaller populations

In the worst case, no runs with a population smaller than the N_k find the solution before the GA_k does. Let us denote the number of generations performed by the run GA_i before the run GA_k has found the solution of a required quality by G_i. Using equations 8, 12, and the relation between the population sizes of the runs, we get

$$E_i = G_i N_i = \left(\frac{m}{2}\right)^{k-i} G_k N_k < 4E_{opt}\left(\frac{m}{2}\right)^{k-i}. \tag{13}$$

By summing the fitness evaluations performed by all runs with populations smaller than N_k, we get the overall number of fitness evaluations from the second class, denoted by E_B, as follows:

$$E_B = \sum_{i=0}^{k-1} E_i < \sum_{i=0}^{k-1} 4E_{opt}\left(\frac{m}{2}\right)^{k-i} = 4E_{opt}\sum_{i=0}^{k-1}\left(\frac{m}{2}\right)^{k-i} \tag{14}$$

Using the last equation and Equation 6, we get

$$E_B < \begin{cases} 4E_{opt}(2+\log_2 \frac{N_{opt}}{N_0}) & \text{if } m = 2 \\ \\ 4E_{opt}\frac{\left(\frac{m}{2}\right)^{3+\log_2 \frac{N_{opt}}{N_0}}-1}{\frac{m}{2}-1} & \text{if } m > 2 \end{cases} \tag{15}$$

By simplifying the last equation, we get

$$E_B = \begin{cases} O\left(E_{opt} log_2 N_{opt}\right) & \text{if } m = 2 \\ \\ O\left(E_{opt} N_{opt}^c\right) & \text{if } m > 2 \end{cases} \tag{16}$$

where $c = \log_2 m - 1$ is a positive real number growing with the logarithm of m.

4.3.3 E_C: Number of evaluations done by the runs with bigger populations

Each run GA_i for $i > k$, performed at least m^{i-k} times less generations than the GA_k (see Equation 9), i.e.

$$G_i \leq \frac{G_k}{m^{i-k}}.$$

Therefore, for i such that $m^{i-k} > G_k$, the GA_i has not been allowed to run at all so far in the run, i.e.

$$\forall i : m^{i-k} > G_k \rightarrow G_i = 0.$$

By solving the last equation, we get

$$\forall i : i > k + \log_m G_k \rightarrow G_i = 0.$$

The number of fitness evaluations performed by the run GA_i for $i > k$ is given by

$$E_i = N_i G_i = \left(\frac{2}{m}\right)^{i-k} N_k G_k < 4E_{opt}\left(\frac{2}{m}\right)^{i-k}.$$

The overall number of fitness evaluations performed by the runs GA_i for $i > k$, denoted by E_C, is then given by

$$E_C = \sum_{i>k} E_i = \sum_{i=k+1}^{k+[\log_m G_k]} E_i < \sum_{i=k+1}^{k+[\log_m G_k]} 4E_{opt}\left(\frac{2}{m}\right)^{i-k} = 4E_{opt}\sum_{i=k+1}^{k+[\log_m G_k]}\left(\frac{2}{m}\right)^{i-k}$$

Since $G_k \leq 2G_{opt}$, we get

$$E_C < 4E_{opt}\sum_{i=k+1}^{k+\log_m 2G_{opt}}\left(\frac{2}{m}\right)^{i-k}$$

Thus,

$$E_C < \begin{cases} 4E_{opt}\left(1 + \log_2 G_{opt}\right) & \text{if } m = 2 \\ 4E_{opt}\frac{\left(\frac{2}{m}\right)^{\log_m 2G_{opt}+1}-1}{\frac{2}{m}-1} & \text{if } m > 2 \end{cases} \tag{17}$$

By simplifying the last equation we get

$$E_C = \begin{cases} O\left(E_{opt} log_2 G_{opt}\right) & \text{if } m = 2 \\ O\left(E_{opt} G_{opt}^d\right) & \text{if } m > 2 \end{cases} \tag{18}$$

where $d = \frac{1}{\log_2 m} - 1$ is a negative real number decreasing with a logarithm of m.

4.3.4 The overall number of fitness evaluations

The overall number of fitness of fitness evaluations needed by the parameter-less genetic algorithm to achieve the solution of a required quality, denoted by E, is given by the sum of the number of fitness evaluations from each of the three analyzed classes, i.e.

$$E = E_A + E_B + E_C.$$

Using equations 12, 15, 17, we get the overall number of fitness evaluations E as follows:

$$E < \begin{cases} 4E_{opt}\left(3 + \log_2 \frac{N_{opt}}{N_0} + \log_2 G_{opt}\right) & \text{if } m = 2 \\ 4E_{opt}\left(1 + 2\frac{\left(\frac{m}{2}\right)^{2+\log_2 \frac{N_{opt}}{N_0}} - 1}{m-2} + m\frac{\left(\frac{2}{m}\right)^{\log_m G_{opt} + \log_m 2 + 1} - 1}{2-m}\right) & \text{if } m > 2 \end{cases} \tag{19}$$

By simplifying the above equations, we get

$$E = \begin{cases} O\left(E_{opt} \log E_{opt}\right) & \text{if } m = 2 \\ O\left(E_{opt}(N_{opt}^c + G_{opt}^d)\right) & \text{if } m > 2 \end{cases} \tag{20}$$

where $c = \log_2 m - 1$ and $d = \frac{1}{log_2 m} - 1$ are constants depending on the value of a logarithm of m.

4.4 Space Complexity Analysis of the Worst Case

In this section, we will compute the memory used by the solutions from populations of all initialized runs in the parameter-less genetic algorithm.

The GA with optimal population size has to store at most $2N_{opt}$ solutions at the same time. Therefore, its space complexity with respect to the number of individuals stored at a time, denoted by M_{opt}, is given by

$$M_{opt} = 2N_{opt} \tag{21}$$

Using the results of the previous section, at most the algorithms GA_0 to $GA_{k+[log_m 2G_{opt}]}$ are run at the same time. Using Equation 2, we get that the number of solutions that have to be stored at any time during the computation, denoted by M, is given by

$$M = 2N_0\left(1 + 2^1 + \ldots + 2^{k+[log_m 2G_{opt}]}\right) \tag{22}$$

Thus,

$$M \leq 2N_0\left(2^{k+log_m 2G_{opt}+1} - 1\right) \tag{23}$$

By simplifying the above equation, we get the overall space complexity of the parameter-less genetic algorithm

$$M \leq O\left(N_{opt} G_{opt}^{log_m 2}\right) = O\left(M_{opt} G_{opt}^{\frac{1}{log_2 m}}\right) \tag{24}$$

Therefore, the parameter-less genetic algorithm requires space proportional to the space required by the GA with optimal population and the number of generations required by the GA with optimal population size to find the solution powered to the base-m logarithm of 2. The space requirements therefore decreases with m.

5 Discussion

The worst-case analysis provides an upper boundary for the time required by the parameter-less genetic algorithm to find the solution of the same quality with the same probability as the simple genetic algorithm with the population set up optimally. The resulting complexity depends on a base m that determines the proportion of fitness evaluations given to each of the simulated runs.

The analysis provided in this paper is very useful because it gives estimates of worst-case time and space complexities of the parameter-less genetic algorithm. However, the reader should be very careful in interpreting the results given herein. The analysis does not take into account a cumulative effect of running multiple independent runs. The more runs (however badly configured), the bigger the chance to find the solution. Moreover, in actual practice, the worst-case scenario is very unlikely to occur. For example, regarding the time complexity alone, $m = 2$ is the best choice in the worst-case scenario. However, in an average case scenario this might not be the case. In general, problems that suffer from the effects of genetic drift will benefit from low values of m, while for other problems a larger m will be better.

The issue of choosing the base m seems to go against the purpose of the parameter-less genetic algorithm work. After all, the main purpose of this approach is to relieve the user from having to set the parameters of the GA. In the original parameter-less genetic algorithm (Harik & Lobo, 1999), the value of m has been fixed to $m = 4$. This is a reasonable compromise between problems that are affected by genetic drift and those that are not.

6 Conclusions

The worst-case analysis of the parameter-less genetic algorithm has shown that this algorithm, even in the worst case scenario, does not increase the computational requirements of the GA with an optimal population size significantly and therefore it offers an efficient method to discard the GA parameters. Moreover, the method can be used for any of commonly used evolutionary algorithms, such as the linkage learning genetic algorithm, univariate marginal distribution algorithm, the extended compact genetic algorithm, or the Bayesian optimization algorithm without any significant changes to the method or its complexity analysis.

Acknowledgments

The authors would like to thank Erick Cantú-Paz and David E. Goldberg for discussions and comments that helped to shape this work.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0050. Research funding for this project was also provided by a grant from the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. Martin Pelikan was supported by grants number 1/4209/97 and 1/5229/98 of the Scientific Grant Agency of Slovak Republic.

The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the Air Force of Scientific Research or the U.S. Government.

References

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

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.

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation* (pp. 7–12). Piscataway, NJ: IEEE.

Harik, G., & Lobo, F. (1999). *A parameter-less genetic algorithm* (IlliGAL Report No. 99009). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory. To appear in the Proceedings of Genetic and Evolutionary Computation Conference GECCO-99, July 13-22, Orlando, Florida.

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