Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Sciences 167 (2004) 217–232

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

The parameter-less genetic algorithm in practice

Fernando G. Lobo [a,*], David E. Goldberg [b]

[a] *Area Departamental de Engenharia Electrónica e Computação, FCT, Universidade do Algarve, Campus de Gambelas, 8000 Faro, Portugal*
[b] *Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA*

Received 20 May 2002; received in revised form 27 September 2002; accepted 18 March 2003

Abstract

The parameter-less genetic algorithm was introduced a couple of years ago as a way to simplify genetic algorithm operation by incorporating knowledge of parameter selection and population sizing theory in the genetic algorithm itself. This paper shows how that technique can be used in practice by applying it to a network expansion problem. The existence of the parameter-less genetic algorithm stresses the fact that some problems need more processing power than others. Such observation leads to the development of a problem difficulty measure which is also introduced in this paper. The measure can be useful for comparing the difficulty of real-world problems.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Genetic algorithms; Parameter selection; Problem difficulty; Network optimization problems

1. Introduction

The parameter-less genetic algorithm was proposed recently [7]. Its motivation was to make genetic algorithms (GAs) easier to use and to make them

[*] Corresponding author.
E-mail addresses: flobo@ualg.pt (F.G. Lobo), deg@uiuc.edu (D.E. Goldberg).

0020-0255/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2003.03.029

available to as large an audience as possible. The need for solving problems occur in a variety of domains and GAs can be useful tools for that purpose. What has been difficult up to now is the interaction between the GA and the user; GAs require the specification of a number of parameters for which users usually do not know how to specify. In other words, the user needs to have a substantial amount of GA expertise in order to apply them in a proper way. But most users are not (and should not be required to be) experts in the field of genetic algorithms. An analogy comes handy here. When people use an electrical appliance, say a toaster, they do not have to know about Ohm's law or about the internal workings of the toaster. Yet, people use toasters everyday, and most of them have never heard about Ohm's law. [1] Likewise, with genetic algorithms, users should not need to worry about population sizes, crossover probabilities, and other GA internals. Yet, people should be able to use GAs, even if they do not understand much about how they work.

In the original paper [7], the validity of the parameter-less GA was illustrated on artificial problems. In this paper it is applied to a quasi-real-world problem, a scenario that users are more likely to encounter in practice. Artificial problems are useful for testing GAs under carefully controlled conditions so that specific aspects of the algorithm can be analyzed. If that was not done, it would not be possible to get a better understanding of GAs, and it would not be possible to improve the state of the art in GA technology. However, it is also important to keep an eye on the applications side and see how the GA performs on problems that are not artificially constructed. Moreover, it is important to illustrate how the lessons learned from theory can be transferred to a practical context. That is precisely what we do here.

The paper starts by reviewing the parameter-less genetic algorithm. Then, Section 3 describes the network expansion problem, followed by the application of the parameter-less technique on Section 4. Section 5 presents an empirical measure of problem difficulty. Finally, the paper outlines some extensions to this work.

2. Background

This section briefly reviews the parameter-less GA. With the parameter-less GA, the user does not need to specify the selection rate, crossover probability and the population size parameters.

[1] This analogy is taken from [9] in the context of software design and used here in the context of genetic algorithm usability.

2.1. Selection rate and crossover probability

The selection rate s and crossover probability p_c are preset to fixed values ($s = 4$, $p_c = 0.5$) in order to obey the schema theorem and ensure building block growth.

At first sight, one could argue that setting $s = 4$ and $p_c = 0.5$ for all problems constitutes a similar kind of mistake that other practitioners have done in the past when adopting the so-called "standard parameter settings". However, there is an important difference in this case. Previous theoretical studies [4] have shown that there must be a balance between selection pressure and schema disruption in order to ensure building block growth. This argument comes from a simplified view of the schema theorem. 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:

$$s(1 - p_c)$$

Setting $s = 4$ and $p_c = 0.5$ gives a net growth factor of 2, and ensures that the necessary building blocks will grow. Other values of s and p_c could also be used as long as the net growth factor is somewhat greater than 1, and some care is taken not to fall in the extreme cases of a very high or very low selection pressure [4].

It is important to stress that we are not saying that these parameter values ($s = 4$ and $p_c = 0.5$) are the best ones for all problems. Small variations of these values (for example, $s = 3$ and $p_c = 0.6$) are likely to yield similar GA performance. Since we are interested in simplifying genetic algorithm operation, we decided to set $s = 4$ and $p_c = 0.5$, because doing so relieves the user from having to make guesses on these parameter values. In other words, setting s and p_c to fixed values is a rational decision which is backed up by previous theoretical work.

2.2. Population sizing

Regarding the size of the population, the parameter-less GA uses a technique that keeps increasing the population size in an attempt to reach the right sizing. It does so by establishing a race among multiple populations of various sizes (powers of 2). The different populations are at different stages of evolution, the smaller ones being ahead of the larger ones in terms of generations. For example, a snapshot of the parameter-less GA at a particular point in time could reveal the existence of three populations whose sizes could be 256, 512, and 1024. The population of size 256 could be running its 30th generation, while the population of size 512 could be on generation 6, and the population of size 1024 could still be on generation 1.

As time goes by, the smaller populations are eliminated and larger populations are created. The creation and deletion of populations is controlled by inspecting the average fitness of the populations and taking rational decisions based on those readings. For example, if the population of size 512 has an average fitness greater than that of the population of size 256, then there is no point in continuing running the small population anymore because it is very unlikely that the smaller population will produce a fitter individual than the larger population. Recall that the larger population is at a much earlier stage of evolution but already contains better individuals than those contained in the smaller one, a clear indication that the smaller population is not large enough.

The overall net effect of this strategy is equivalent to a scheme that continuously increases the population size as time goes by. The interested reader should refer to [7,11] for a through description and implementation details of the parameter-less GA. What we would like to stress at this point is the need of large populations, something that still seems to be underestimated by large parts of the evolutionary computation community.

2.3. The need of large populations

All search algorithms have an initial state and perform some sort of inference in order to move from state to state. In the context of genetic algorithms, the state is represented by the population, and the GA moves from population to population. The population can be seen as a collection of data that summarize the past experiences of the GA. Therefore, the larger the population the more accurately the GA will be able to infer the next state. Indeed, what the GA does is nothing but inference from data. Researchers in the field of *Data Mining*, also known as *Knowledge Discovery in Databases*, deal precisely with this type of inference problem. However, in many situations the lack of a sufficient amount of data is the problem. Without sufficient data it is not possible to do accurate inference. Within a GA framework, obtaining more data is not a problem because it is always possible to generate more data; that is simply a matter of working with larger populations. The question is how large is large enough, a topic that is discussed next.

2.4. How large is large enough?

The population size is a critical parameter in a GA. Too small and the GA converges to poor solutions. Too large and the GA spends unnecessary computational resources. There are theoretical models [3,6] that can be used to size populations but they are not easy to apply in practice because they rely on parameters that are usually unknown and are also hard to estimate for real world problems. The intuition behind population sizing in GAs, however, is that it is related to the problem's difficulty; the more difficult a problem is the larger the population should be. However, since problem difficulty is also hard to estimate, GA practitioners usually have no other way to size populations other than do experiments with a number of different sizes and see what works best.

Although not trivial to put in practice, the theoretical models on population sizing are important and crucial for understanding the role of the population in a GA. Among other things, an important lesson of those models is that setting the population size to a fixed value regardless of the problem's size and difficulty, is certainly a mistake.

In practice, the bottom line is that the user has to do some experimentation and guess the population size. But guessing right is pure luck and most likely the user will guess wrong by doing one of the following two mistakes: (1) a population size that is too small, or (2) a population size that is too large.

Type I error: undersized population. If the population size is not big enough, the GA runs out of steam prematurely and converges to sub-optimal solutions; the user pays a *quality penalty*.

Type II error: oversized population. If the population size is too large, the GA wastes unnecessary computational resources spending more time than it is necessary; the user pays a *time penalty*.

These two types of errors are depicted in Fig. 1. The parameter-less GA uses a technique that was developed on purpose to eliminate the need of guessing the population size, and therefore, avoid the errors shown in Fig. 1. The basic idea is to continuously increase the population size in an attempt to reach the right sizing. When to stop this growing process of the population is left to the user. He/she will decide when to stop as soon as he/she realizes that is not worth to wait more for an improvement in solution quality.

2.5. A note about mutation

The parameter-less technique ignores the mutation operator. We recognize that mutation can be important for many problems but have not considered yet how to automate it within the rest of the parameter-less GA framework.

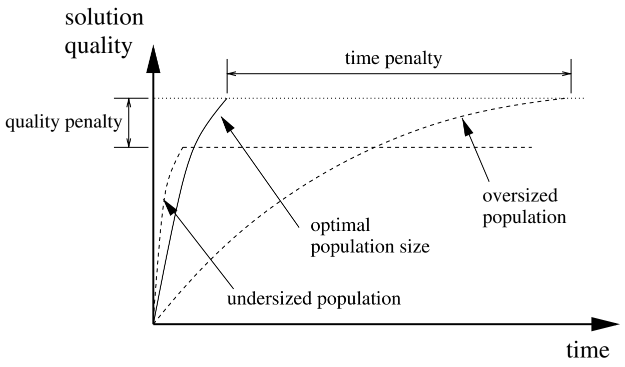

Fig. 1. Population sizing in genetic algorithms. Too small and the user pays a quality penalty. Too large and the user pays a time penalty.

Mutation makes small variations on solutions and thus is not likely to benefit from very large populations. On the contrary, crossover requires large population sizes in order to mix the bits and pieces of the different solutions.

Section 4 shows the application of the technique to a network expansion problem, but before doing that, let us present what the network problem itself.

3. A network expansion problem

This section describes a utility network expansion problem. For its exposition we focus on the particular case of an electrical network. Without further considerations, let us describe what the network expansion problem is with an illustrative example shown in Fig. 2. The figure shows a region that does not

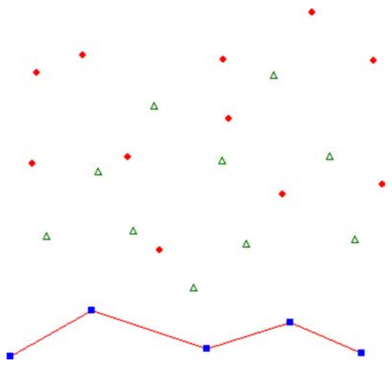

Fig. 2. A hypothetical 10-bit (10-transformer) network expansion problem instance.

have electrical facilities, an hypothetical instance of the network expansion problem.

There are four types of entities depicted in the figure: cables, substations, possible transformer locations, and houses. These entities are represented in the figure by lines, squares, triangles, and dots respectively. In the example there are five substations (squares) connected by cables in a network. The objective of the problem is to expand the network so that the houses (dots) can get electricity. Moreover, the electrical utility company would like to do so with minimum cost.

The substations are the only entities that take part of the electrical infrastructure. The transformers (triangles in the figure) do not exist yet but there are a number of possible locations where they can be built. These locations are given in advance by the electrical utility and may take into account a variety of restrictions.

There are many ways in which the houses can be connected to the existing network. The only restriction is that the end result has to be a tree (a graph with no cycles) and it is allowed to use a subset of the possible location sites to build transformers.

The total cost of expanding the network is the sum of the costs of all the cables and transformers that need to be built. Each transformer that is built has a fixed cost associated and the cost of each cable is proportional to its length. The electrical company must decide which cables and transformers should be built in order to deliver electricity using the minimum amount of money as possible.

In summary, one has to decide which transformers should be built. Once that decision is taken, expanding the network can be done with a straightforward computation.

Figs. 2 and 3 illustrate the fitness function in four steps by showing the construction of the network on a hypothetical 10-transformer network problem. There are 10 possible locations (the 10 triangles in Fig. 2) to build transformers. For each location, a binary decision must be made by the power company: build or not build a transformer in that location. In Fig. 3—part (a), four locations are selected for building transformers and the other six are left out. The example corresponds to solution 0101000110, which is one out of the 2^{10} possible solutions. Then, a graph is constructed in the following way. For each selected transformer node, an edge is added from that node to all the existing substation nodes and to all the other selected transformers. Following that, a minimum spanning tree of the graph is computed. There are a variety of algorithms to find minimum spanning trees (see for example [1]). Notice that for the minimum spanning tree computation, the edges connecting the substations have zero cost because these edges correspond to existing cables and no additional cost is needed to be spent by the utility company. Once the tree is constructed, each house can be connected to the network by adding an edge

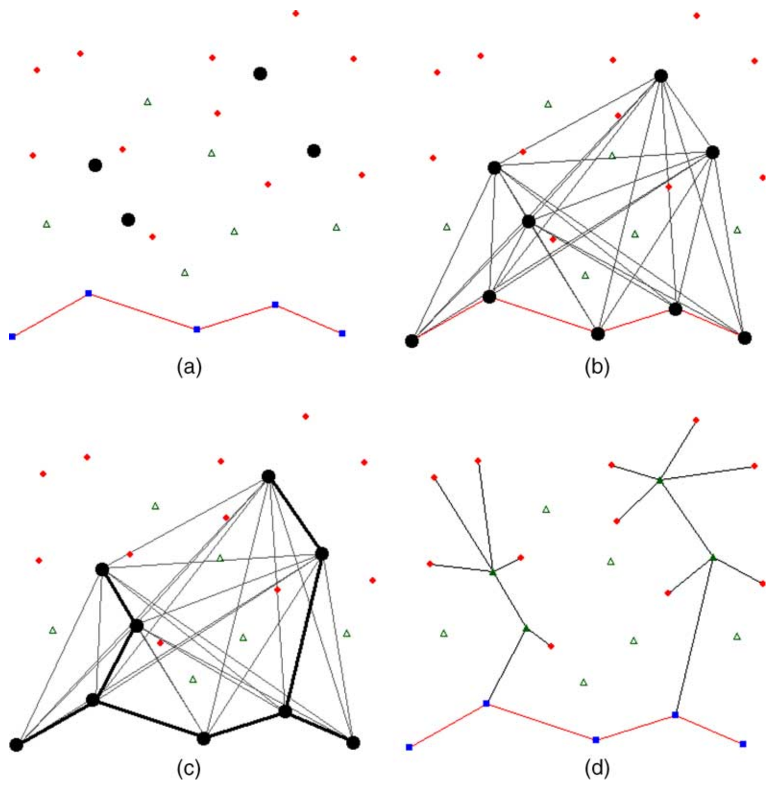

Fig. 3. Example of objective function evaluation in a hypothetical 10-bit (10-transformer) problem instance. (a) Four transformer nodes are selected to be built. They are represented on the figure by the large circles. (b) A graph is constructed by adding edges from every selected transformer to all the other selected transformers and to all the substations. (c) A minimum spanning tree of the graph is computed. (d) Each house connects to the closest node of the minimum spanning tree.

from the house to the closest node of the tree. The final result is shown on Fig. 3—part (d).

This problem has similarities with a well known NP-complete problem from graph theory known as the *minimum steiner tree problem* [2]. The difference is that in the minimum steiner tree problem, the possible location for the transformer nodes are not given in advance. There are a number of approximate algorithms and heuristics to solve the minimum steiner tree problem. In this paper we do not make any comparison between the GA and these other heuristics. Moreover, the GA would not take into consideration any special properties of the problem and it will treat it as a pure black-box function. It should be stressed that the purpose of the paper is not to show that the GA is superior or even competitive with specialized algorithms specifically tuned for this particular problem. Instead, the purpose of the paper is to illustrate how GA technology may be applied in an environment where not much is known other than the objective function values of individual solutions, and also to illustrate the transition from laboratory problems to real-world problems.

We have described a method to expand the existing network once the transformers to be built are specified. The decision variables of the problem are binary variables, one for each possible transformer location, indicating whether that transformer should be built or not. Thus, if the electrical utility company specifies ℓ possible locations to build transformers, the total number of possible network configurations is 2^{ℓ} and the application of a genetic algorithm is straightforward.

The network expansion problem has several characteristics that contrast with those of purely artificial problems. Some of them are listed below:

- The global optima is unknown.
- The problem's structure is unknown.
- We do not know what might constitute building blocks.
- We do not know if the problem is easy or hard.

4. Parameter-less GA application

This section shows the application of the parameter-less GA to the network expansion problem described during the previous section. In particular, the GA is applied to a 60-bit problem instance. It should be stressed that the parameter-less technique relieves the user from having to specify the population size, selection rate, and crossover probability, but it does not relieve the user from specifying the type of GA operators to be used. In fact, the parameter-less technique can be used with any kind of GA.

In a first experiment, we use the parameter-less technique coupled with a simple GA using uniform crossover and with mutation turned off. Table 1 shows a trace of the execution of a single run of the parameter-less GA. The algorithm starts with a population of size 16 and continuously increases (doubles) its size. For each population size, Table 1 shows the best solution quality found. For example, when using a population of size 128, the best solution had a cost of 1990.25.

A number of observations are worth mentioning. First and foremost, the experiment is very simple (of course, there are no parameters!). Larger and larger populations are continuously spawned. By doing that, the parameter-less technique injects more and more power into the GA as time goes by.

The parameter-less GA was stopped when the population of size 65 536 had already converged but the population of size 131 072 was still in its early generations (see [7] and [11] for details of the execution of the algorithm). In

Table 1
The parameter-less GA on a 60-bit problem instance

Population size	Solution quality with parameter-less SGA
16	2131.79
32	2109.33
64	2004.93
128	1990.25
256	1971.50
512	1984.76
1024	1986.49
2048	1967.21
...	...

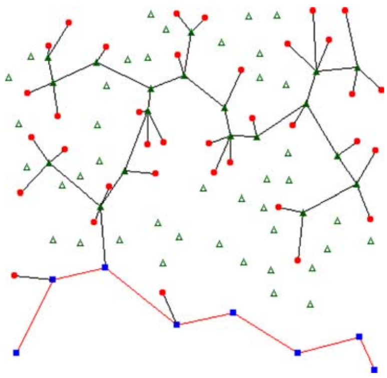

Fig. 4. The best solution found by the parameter-less GA on a 60-bit problem instance.

this specific example, the user (ourselves) stopped the parameter-less GA at that point because we did not want to wait longer for a better solution quality. That of course depends on a problem by problem basis and from user to user. For instance, another user might have been happy with the solution quality of 1990.25 and might have stopped the parameter-less GA by the time that solution was found (population size 128 in the example).

The best solution was found at population size 2048 (see Fig. 4). Larger populations also found that solution but could not get any better. If the user knew that beforehand he/she would have pressed the stop button at population size 2048 and would have saved computational time. Unfortunately, the user does not know that beforehand. In fact, the user cannot predict what would happen if he/she had let the algorithm run for a longer time; it is unknown whether the algorithm can reach a better solution quality or not.

The only thing that we can say is that if the user is not satisfied with the solution quality that he has gotten so far then the best option is to put more power into the GA and that can be done by increasing the population size; the parameter-less GA does that automatically.

Since the population size doubles every time a new population is spawned, it should be expected that using the parameter-less techniques has an overhead when compared with a GA that starts with an optimal population size. A detailed analysis for the worst case scenario has been made elsewhere [14]. In practice, the worst-case scenario is very unlikely to occur, and its actual performance is usually within a factor of 2 or 3 of a GA that starts with optimal parameter settings.

4.1. What about other parameter settings?

It is time to stand back and ask ourselves what option would a user have if the parameter-less technique was not available. Perhaps he/she would have done what many other practitioners have been doing in the past; try different parameter combinations and see what works best, or, use the so-called "standard settings" (population size in the range 50–100, p_c in 0.6–0.9, small p_m, and so on).

We would not be doing exhaustive parameter combinations and compare them with the parameter-less technique, but we can easily experiment with the standard settings just to see what happens. Doing that (population size 100, $p_c = 0.7$, $p_m = 1/\ell$, binary tournament selection), the SGA could not reach the target solution obtained by the parameter-less technique not even after a million function evaluations. On the contrary, various executions of the parameter-less GA were able to consistently get to that target solution of 1967.21 with less than a couple hundred thousand fitness function evaluations. This experiment confirms again that using the so-called "standard settings" is no recipe for good performance. On the contrary, it is a mistake and should be avoided.

4.2. What about other operators?

The parameter-less technique relieves the user from setting parameters such as population size, selection rate, and crossover probability, but says nothing about which GA operators to choose from. That design decision was done on purpose when the parameter-less technique was originally developed. This way, the technique becomes general and may be applied with any kind of GA.

Instead of using a parameter-less simple GA, we could have used a parameter-less *extended compact genetic algorithm* (ECGA) [5] or a parameter-less *Bayesian optimization algorithm* (BOA) [13]. On difficult problems, these more advanced techniques have shown to outperform the traditional simple GA by several orders of magnitude.

The parameter-less ECGA was tested on the same 60-bit network problem instance and the solution quality obtained was the same as the one obtained by the parameter-less simple GA. Moreover, it did so using more or less of the same number of fitness function evaluations. This may be an indication that the particular problem instance is not a very difficult problem after all. On the other hand, the problem is also not that easy because otherwise the simple GA would not require a population of size 2048; a smaller size would have been enough.

This discussion on problem difficulty can be explored further and that is precisely what we do in the next section.

5. Towards an empirical measure of problem difficulty

During the previous sections we have argued that the amount of computing power that needs to be put in order to solve a problem is related to the problem's difficulty. Different facets of problem difficulty have been pointed out by researchers in the evolutionary computation field. Among these difficulties are isolation, deception, multi-modality, bad building block scaling, and noisy fitness functions.

From a practical perspective it would be nice to estimate the degree of difficulty of a given problem because then it would be possible to estimate the amount of computing power (and also time) needed to solve it.

There has been a few studies regarding problem difficulty estimation. Among them, the works of [10,12,15] have proposed different measures that can be useful to predict if one operator is likely to behave better than other.

One thing that can give an overall measure of a problem's difficulty is the amount of effort that is required to solve it. In GAs, the amount of effort can be measured by the minimum number of fitness function evaluations needed by the GA to get reliably to some desired target solution. It is possible to go further and define an empirical measure of a problem's difficulty, call it R_{hard}, which can be defined as the minimum number of function evaluations needed by the parameter-less GA for solving the given problem up to a certain solution quality divided by the minimum number of function evaluations needed by the parameter-less simple GA to solve a onemax problem of the same size. The onemax problem (that of counting 1s in a bit string) is chosen as a reference for comparing the difficulty of problems because in some sense it is the easiest of all problems for the GA—there is no interaction among the decision variables of the problem, the function is completely decomposable, and the scaling is uniform (all bits give the same contribution to the fitness function).

Let $\hat{f}$ be the minimum number of fitness function evaluations needed by the parameter-less GA to solve a problem to a certain solution quality, and let f_{onemax} be the minimum number of fitness function evaluations needed by the parameter-less GA to solve a onemax problem of the same length. The *hardness ratio*, R_{hard}, can be defined as follows:

$$R_{\text{hard}} = \frac{\hat{f}}{f_{\text{onemax}}}$$

This definition is an empirical measure of a problem's difficulty regarding crossover-based GAs. A ratio $R_{\text{hard}} = 1$ indicates that the problem is as difficult as the onemax. A ratio $R_{\text{hard}} = r$ indicates that the problem is r times more difficult than its onemax counterpart. Although this empirical measure of problem difficulty can only be calculated after the problem itself is solved, it is a practical measure that can be used to compare the difficulty of real world problems from a rational basis. Thus, when doing GA applications, users can talk about a real world problem's difficulty and say things like: problem X is $R_{\text{hard}} = 10$, problem Y is $R_{\text{hard}} = 2000$, and so on.

As an illustrative example, let us calculate the R_{hard} ratio for the 60-bit network problem. Doing an average of 20 independent runs, the parameter-less GA needs 161 200 function evaluations to get to the target solution of 1967.21. Likewise, for a 60-bit onemax problem, the parameter-less GA needs an average of 4200 function evaluations in order to get to the optimal solution. Therefore, the R_{hard} ratio for our 60-bit network problem is $161\,200/4200 \approx 38$.

More specific measures of GA difficulty could be defined as well. For example, we could define a mixing hardness ratio, call it $R_{\text{mixingHard}}$, defined as the number of function evaluations needed by a simple GA divided by the number of function evaluations needed by a linkage learning GA (such as the ECGA or BOA) to reach a certain target solution.

6. Extensions

The parameter-less technique is a step forward towards making GAs easier to use. With it, the user does not need to specify parameters such as population size, selection rate, and crossover probabilities. However, there are still some decisions that the user needs to make. For example, the user must specify an encoding for the problem as well as GA operators. In the example shown in this paper we used a bit string encoding and a uniform crossover operator. Other choices could have been made as well but the parameter-less technique says nothing about that.

One of the motivations of the research in linkage learning is to develop techniques that can adapt the GA operators on a problem by problem basis. Some of these advanced techniques have shown to outperform the traditional simple GAs by several orders of magnitude, especially on very difficult problems. Two of such techniques are the ECGA and BOA algorithms that were mentioned awhile ago. These advanced algorithms, however, do not come for free. They do a through analysis of the population and require more effort than the simple GA when moving from population to population (note that the extra effort is needed by the operation of the algorithm itself, not by an extra amount of fitness function evaluations). In many problems the extra operational effort of the ECGA or BOA are well worth it, but for other problems the more traditional crossover operators might suffice.

This discussion leads us back again to the issues of problem difficulty estimation. If we could predict that the problem is very difficult then we could apply a parameter-less ECGA (or BOA). On the other hand, if we could predict that the problem is not so difficult then we could apply a parameter-less simple GA and save ourselves the extra computing power needed for the operation of the more sophisticated GAs.

Since problem difficulty estimation is hard to do, a possible solution to this dilemma could be to run two parameter-less GAs simultaneously (a heavy-weight one such as the ECGA or BOA, and a light-weight one such as the simple GA or the compact GA [8]) using a trick similar to the one used by the parameter-less technique to tackle multiple populations.

7. Summary and conclusions

This paper reviewed the parameter-less genetic algorithm and showed a practical application of it to a utility network expansion problem. The problem has characteristics that contrast with those of pure artificial problems, and constitutes a more representative scenario of what users might encounter in practice. Another important contribution of this paper is the introduction of an empirical measure of problem difficulty. The measure is enabled by the existence of the parameter-less GA, and can very useful for comparing the difficulty of real-world problems.

With the parameter-less GA the user does not have to do trial and error experiments to find adequate parameter settings for the GA. It is our strong belief that GAs should be designed with the user in mind. That is the only way to have more and more people using them. This paper makes an important effort in that direction and we hope it can be useful for practitioners seeking to apply state of the art GA technology to solve real world problems.

Acknowledgements

This work was sponsored in part by the Portuguese Foundation for Science and Technology (FCT/MCT), under grant POSI/SRI/42065/2001 and grant POCTI/MGS/37970/2001.

The work was also sponsored in part by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-00-0163, the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are our own and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, the US Army, or the US Government.

References

[1] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, McGraw-Hill, 1993.

[2] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, 1979.

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

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

[5] G.R. Harik, Linkage learning via probabilistic modeling in the ECGA, IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, 1999.

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

[7] G.R. Harik, F.G. Lobo, A parameter-less genetic algorithm, in: W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Eds.), GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, CA, 1999, pp. 258–267.

[8] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 287–297.

[9] R.E. Johnson, Frameworks = (components + patterns), Communications of the ACM 40 (10) (1997) 39–42.

[10] T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: L. Eshelman (Ed.), ICGA-95: Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, 1995, pp. 184–192.

[11] F.G. Lobo, The parameter-less genetic algorithm: Rational and automated parameter selection for simplified genetic algorithm operation, Doctoral dissertation, Universidade Nova de Lisboa, Lisboa, 2000.

[12] B. Manderick, M. de Weger, P. Spiessens, The genetic algorithm and the structure of the fitness landscape, in: R.K. Belew, L.B. Booker (Eds.), ICGA-91: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 143–150.

[13] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, BOA: the bayesian optimization algorithm, in: W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Eds.), GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, CA, 1999, pp. 525–532.
[14] M. Pelikan, F.G. Lobo, Parameter-less genetic algorithm: a worst-case time and space complexity analysis, IlliGAL Report No. 99014, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, 1999.
[15] E.D. Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference, Biological Cybernetics 63 (1990) 325–336.