Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 36 (2014) 523 – 528

Procedia Computer Science

Complex Adaptive Systems, Publication 4
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2014-Philadelphia, PA

Harnessing Mother Nature: Optimizing Genetic Algorithms for Adaptive Systems

Justin Lovinger[a], Iren Valova[a]*, MacKenzie Rogers[a], Ryan Nadeau[a], Natacha Gueorguieva[b]

[a]*Computer and Information Science Department, University of Massachusetts Dartmouth, MA 02747, USA*
[b]*Computer Science Department, City University of New York, NY 10314, USA*

Abstract

Many adaptive systems require optimization in real time. Whether it is a robot that must maintain its gait regardless of the terrain or multicore systems needing proper scheduling, optimization is of utmost importance. With hundreds of processes created and evaluated every second, real-time performance optimization is a monumental task. Mother nature has proven that evolution is very effective form of adaptation. Through a stochastic search, i.e. GA, computers harness this power. GAs have been developed to utilize many different parameters, which have a significant effect on the efficiency and effectiveness of a GA. If a GA tasked to optimize these parameters, the result is a rapid and automatic optimization. To test our hypothesis we optimize a GA that solves common optimization functions. The GA's effectiveness is determined by the time it takes to find the solution. Cross validation is utilized, and shows an average 947% performance improvement on training sets and 440% on testing sets. This large improvement in the testing sets shows that an optimized genetic algorithm remains general enough to effectively solve similar problems.

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology

Keywords: genetic algorithms; optimization; embedded systems performance

* Corresponding author. Tel.: 5089998502; fax: 5089999144.
E-mail address: ivalova@umassd.edu

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology
doi:10.1016/j.procs.2014.09.033

1. Introduction

Genetic algorithms (GA) have become an integral part of numerous adaptive systems. Although, at their core, GA are simply a form of search, their effectiveness and generalizability have made them very popular. Despite their popularity, GA can be very finicky. Although effective and powerful when parameters are set correctly, a genetic algorithm with improper parameters can be anywhere from very slow to completely ineffective. Of course, human beings are more than able to adjust these parameters manually, but why should we have to? We are talking about as many as half a dozen parameters that all interact with each other in complex ways. Because of these complexities, the process of manually tweaking, testing, and adjusting again can take several hours for a person to complete. Instead, we let the computer perfect these parameters on its own.

We have organized this paper in six sections, presenting our method for automatic optimization of the parameters of a genetic algorithm in section 3; the testing of this method with a GA for function minimization, and proving the effectiveness and generalizability of the method in section 5; and finally, we provide an example of the effects of our method on a real world problem in section 3.

2. State-of-the-art and applications

Since GA are a popular search method in many fields, their optimization has been investigated in several papers. The most common method of optimizing genetic algorithms is the systematic testing of various parameter combinations. This is the method human beings utilize when manually adjusting genetic algorithms. However, it is not uncommon to automate this process[1,2]. Then, a number of parameter sets are created, either by hand or automatically and are tested sequentially, where an evaluation function determines the effectiveness of each set of parameters. The best parameter set is then chosen.

The optimization of one genetic algorithm with another genetic algorithm has also been investigated on occasion. The authors in [2] investigate a genetic algorithm optimization system similar to ours, with a focus on road traffic optimization. Although their system encodes similar parameters, their method of optimization differs greatly, as it is optimizing the final fitness of the genetic algorithm, not the efficiency and time of the genetic algorithm.

Testing the performance of a genetic algorithm with minimization problems is common practice. In [3] minimization problems are utilized to test the effectiveness of various genetic operators.

Just about any adaptive system benefits from an increase in efficiency and performance. However, for some adaptive systems optimization is essential, e.g. systems running on embedded hardware. These are applications where computing power is limited and battery life is a concern. An interesting example of adaptation in embedded hardware is auto adaptive movement for robotics[4,5,6]. This places a robot in a continual state of evolution. Rather than being designed to function well in a specific environment, and with hardware in perfect condition, a robot with auto adaptive movement evolves to its environment and internal state. Even if its environment catches on fire or it loses a leg, this type of robot finds a way to keep going. Since such a robot is chasing an ever-changing optimal movement pattern, it has greater processing and battery requirements than the average robot. Increased efficiency allows this type of robot to function longer and stay ahead of the curve by adapting more rapidly.

Another performance critical adaptive system is process scheduling for computing clusters[7] and multicore machines[1]. These systems must evaluate hundreds of processes and threads in milliseconds, and every time a new process or thread is created, the schedule must be reevaluated. With processes being created every second, this becomes a monumental task. Every increase in efficiency of the process scheduling system leads to increased performance for the computer or computer cluster.

3. Proposed technique

Our technique, at its core, utilizes one genetic algorithm (GA) to optimize another GA. Given a problem that is solved with a GA, in our system, this GA is referred to as the inner genetic algorithm (IGA), and it is the one being optimized. The GA performing the optimization is referred to as the outer genetic algorithm (OGA). The OGA optimizes the parameters for the IGA. The parameters discussed and tested in this paper are: mutation chance, crossover chance, population size, crossover function, selection function.

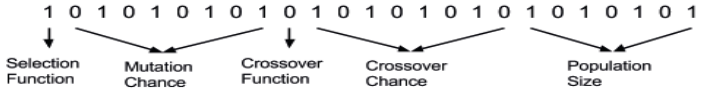


Figure 1 Representation of a chromosome

Indeed, additional parameters are theoretically possible. Our OGA encodes all parameters in binary format. Mutation chance and crossover chance are converted into floating point values during decoding; population size is decoded into an integer; and crossover and selection function are decoded into strings by checking the value of the corresponding bits. The strings for crossover and selection function are used to determine these from a set of preprogrammed functions. The encoding of the IGA is determined by the specific problem it solves.

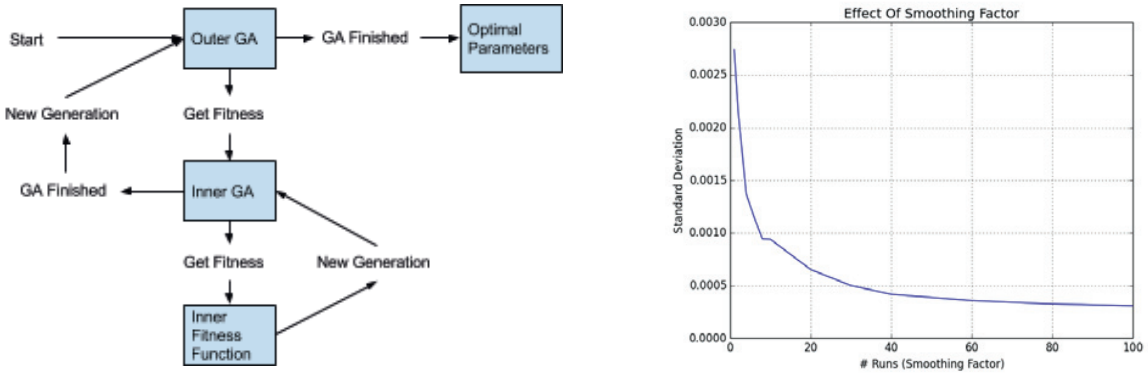


Figure 2 (a) inner/outer GA relationship; (b) effect of smoothing factor

When our optimization system is utilized, it first generates a number (n) of random parameter sets for the IGA, encoded in binary. This is the population of the OGA. The IGA is then run in its entirety n times, each time using one of the generated parameter sets. How long the IGA takes to solve its defined problem determines the fitness of its parameter set. Once each parameter set has a fitness, a new generation of fitness sets can be generated by the OGA, through crossover and mutation. This process (Figure 2a) is repeated to determine the optimal parameter set.

However, the random nature of genetic algorithms leads to large variation in the performance of the IGA, given a set of parameters. This is because genetic algorithms depend on random initial populations, random crossover points, and random mutations. A GA can produce an initial population that includes the solution, which results in a solution being found very quickly, regardless of parameters. A GA can also produce a poor initial population, with very "unlucky" crossovers and mutations, which results in a solution being found very slowly, if at all. GAs always have a very good best performance and a very poor worst performance, but the average performance for a GA can vary greatly. Therefore, we aim to optimize this average performance. We do this by running the IGA several times, and taking the average of their run times. The number of times the IGA is run before averaging is the smoothing factor. Figure 2b shows how the smoothing factor affects the variation in performance. Our goal is to minimize this variation, so that the fitness for a set of parameters approaches the average performance. However, because a greater smoothing factor slows the optimization process, there must be a compromise. Our tests show that at least 20 runs per parameter set is necessary for proper optimization, and more than 100 runs shows little improvement. In addition to the time factor, the number of times the IGA cannot find a solution is used to penalize the fitness for a set of parameters. This ensures that a set of parameters is consistent as well as fast.

The evaluation function for the IGA depends on the problem it is solving. However, in order for this technique to properly function, the IGA must be able to implement a stopping point of some form. This stopping point can either be a known solution, or a fitness threshold that indicates that the GA is doing its job well. The purpose of a fitness threshold in this situation is to provide a time factor. Despite this threshold not being a true solution, the faster the genetic algorithm reaches this fitness threshold, the more efficient it is, and the better its parameters are.

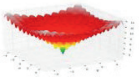
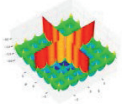
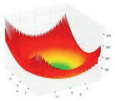
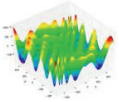
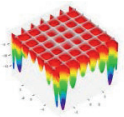
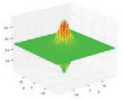
The scheduling of computational tasks on a processor for real time systems is a common problem addressed by genetic algorithms[5]. Efficient allocation of distributed tasks to a CPU results in maximized utilization of the

processor. The application of a genetic algorithm to process scheduling eliminates ramifications generated by exhaustive search, including inconsistency of performance and overall limited speed of processes. Through training such a genetic algorithm, we can minimize the time it takes to optimally schedule presented tasks. While the inner genetic algorithm seeks an optimal scheduling solution for an input of processes, the outer genetic algorithm seeks an optimal performance for the inner genetic algorithm.

Typically, a process-scheduling genetic algorithm will terminate after a maximum number of generations, using the best solution up to that generation as the final one. With our method, we can elicit this solution in a fraction of the time. Our tests show an average speed improvement of 10 times. The scheduling algorithm can therefore explore 10 times the number of generations in the same amount of time, and potentially find an even more favorable solution. Real time systems do not guarantee that all deadlines will be met. Optimization of a genetic algorithm for process scheduling can minimize missed deadlines, and increase process throughput.

4. Testing methodology

Table 1. Our optimization problems

Problem	Equation	Solution
Ackley's Function 	$f(x,y) = 20\exp(-0.2\sqrt{0.5(x^2+y^2)} - \exp\big(0.5(\cos(2\pi x)+\cos(2\pi y))\big) + 20 + e$	0.0
CrossInTray function 	$f(x,y) = -0.0001\left(\left\lvert \sin(x)\sin(y)\exp\left(\left\lvert 100 - \frac{\sqrt{x^2+y^2}}{\pi}\right\rvert\right)\right\rvert + 1\right)^{0.1}$	-2.06261
Lévi Function N.13 	$f(x,y) = sin^2(3\pi x) + (x-1)^2\big(1+sin^2(3\pi y)\big) + (y-1)^2\big(1+sin^2(3\pi y)\big)$	0.0
Eggholder function 	$f(x,y) = -(y+47)\sin\left(\sqrt{\left\lvert y+\frac{x}{2}+47\right\rvert}\right) - x\sin\left(\sqrt{\lvert x-(y+47)\rvert}\right)$	-959.6407
Hölder table function 	$f(x,y) = -\left\lvert \sin(x)\cos(y)\,exp\left(\left\lvert 1 - \frac{\sqrt{x^2+y^2}}{\pi}\right\rvert\right)\right\rvert$	-19.2085
Schaffer function N. 2 	$f(x,y) = 0.5 + \frac{\sin^2(x^2-y^2)-0.5}{\big(1+0.001(x^2+y^2)\big)^2}$	0.0

To test our method, a set of minimization problems (presented in Table 1) are solved with a genetic algorithm. Each problem has some number of floating point values as inputs, and a known solution that must be discovered by a GA. For each problem, the GA encodes a number of floating point values in binary form. The evaluation function for each problem decodes the chromosome, inputs the resulting floating point values into the problems equation,

and, given that these are minimization problems, returns the inverse of the equations output. If the output of the equation is within 0.1 of the problems solution, the problem is considered solved, and the GA terminates. Our goal is to optimize the time it takes this GA to solve these problems. This GA is our inner genetic algorithm.

To test that our method is effective, cross validation is performed as it provides a robust test of our method by utilizing multiple problems. With cross validation, we can measure the degree of optimization for a training set, on which a GA is directly optimized. Additionally, we can test for overtraining by measuring the effects of an optimized genetic algorithm on a testing set on which the genetic algorithm is not directly optimized.

To perform our cross validation, we first construct three training sets with four problems each, and three corresponding testing sets, with the remaining two problems from our problem set in Table 1. For each training set, we utilize our method to determine the optimal parameters for a genetic algorithm that solves the problems. The same parameters are used for all problems in the training set. With these parameters, we time how long, on average, a genetic algorithm takes to solve the problems in the training set, and the corresponding testing set. Additionally, we perform the same process with a standard set of parameters, not obtained with our method. This allows us to compare the effects of our optimization. We also measure how often our optimized genetic algorithms are able to find a solution. This ensures that our optimized genetic algorithms are both efficient and effective.

5. Results

Table 2. GA parameters

	Fold 1	Fold 2	Fold 3	Standard
Mutation Chance	0.0784	0.0901	0.8941	0.02
Crossover Chance	0.5333	0.1725	0.0941	0.7
Selection Function	Stochastic	Stochastic	Stochastic	Roulette
Crossover Function	One Point	Uniform	One Point	One Point
Population Size	6	8	6	20

Table 2 shows the optimized parameters for each fold. A fold is a combination of a training and corresponding testing set. These parameters are discovered by running the OGA on an IGA that solves the training set of equations for the fold. The standard parameters are hand picked, and fall within the range of expected parameters for a canonical genetic algorithm. Mutation chance is the chance of a bit being flipped during mutation. Crossover chance is the chance, for each set of parents, that crossover will occur. Selection function is the function used for selecting parents. Roulette selection is the canonical roulette selection algorithm. Stochastic universal sampling is a selection algorithm, similar to roulette selection, which guarantees that chromosomes with greater than average fitness are selected, at the expense of diversity. Crossover function is the function used for crossing two parents during crossover. One point crossover is the canonical one point crossover algorithm. Uniform crossover randomly swaps bits between two parents. Population size is the number of chromosomes in the population of every generation.

Table 3 shows the results of our performance analysis. These results are obtained by using a genetic algorithm to solve a set of optimization problems, using either the parameters obtained from our method, or the standard parameters. The standard parameters for each fold are the standard parameters in Table 2. The optimized parameters for each fold are the parameters in the corresponding fold in Table 2. The “Time (Seconds)” rows are the result of timing the IGA and taking the average of 500 runs. The “Times Improvement” rows show the improvement from the standard parameters to the optimized parameters. The “% Solutions Found” rows show the percentage of runs that discovered a solution, for the optimized parameters.

Table 3. Performance analysis

	Fold 1	Fold 2	Fold 3	Mean	STD
Time (Seconds): Standard Parameters (Training Set)	0.0462	0.0677	0.0546	0.0562	0.0089
Time (Seconds): Standard Parameters (Testing Set)	0.0587	0.0143	0.0325	0.0352	0.0182
Time (Seconds): Optimized Parameters (Training Set)	0.0053	0.0052	0.0082	0.0062	0.0014
Time (Seconds): Optimized Parameters (Testing Set)	0.0063	0.0072	0.0167	0.0101	0.0047
Times Improvement (Training Set)	8.7894	12.9785	6.6496	9.4725	2.6285
Times Improvement (Testing Set)	9.2631	1.9916	1.9390	4.3979	3.4403
% Solutions Found (Training Set)	%100.0	%100.0	%100.0	%100.0	0.0000
% Solutions Found (Testing Set)	%100.0	%100.0	%100.0	%100.0	0.0000

The large variation between folds is expected because each fold consists of a unique set of equations, and each equation is unique in its requirements and difficulty. The times improvement row shows that, despite the large variation, all folds show a large improvement to both the training set and the testing set. As expected, the largest improvement exists for the training set, but the improvement to the testing sets show that our method does not overtrain the genetic algorithm. Furthermore, our method does not sacrifice speed for accuracy, as the "% solutions found" rows indicate.

Overall, our optimization results in a massive 9.5 times efficiency increase on average. This allows for much faster genetic algorithm searches with lower system requirements. Also, nearly 10 times more generations can be explored in the same amount of time. By exploring more generations, more effective solutions can be found with genetic algorithms.

6. Conclusions

Genetic algorithms represent a powerful tool for adaptive systems. However, their performance can vary greatly, depending on parameters. We have presented a method for optimizing these parameters, with recommendations for implementation. We have explored the effects of this optimization on a process scheduling problem, and tested our method using a set of common optimization problems and cross validation. Our results show an average efficiency improvement of 9.5 times for directly optimized problems, and 4.4 times for indirectly optimized problems. Furthermore, given the problem, "Ackley's Function", our method can optimize a genetic algorithm to solve this problem in 7.5 minutes. Overall, this method is fast, effective, and maintains accuracy and generalizability.

References

1. Myungryun Y., and Mitsuo G., "Multimedia Tasks Scheduling Using Genetic Algorithm", Asia Pacific Management Review (2005) 10(6), 373-380.
2. T. Potuzak, "Feasibility Study of Optimization of a Genetic Algorithm for Traffic Network Division for Distributed Road Traffic Simulation"
3. Z.Michalewicz, T.Logan, and S.Swaminathan, "Evolutionary operators for continuous convex parameter spaces"
4. M. Anthony Lewis, Andrew H. Fagg, and George A. Bekey, "Genetic Algorithms for Gait Synthesis
in a Hexapod Robot", Published in: Zheng, ed. Recent Trends in Mobile Robots, pp 317-331, World Scientific, New Jersey, 1994.
5. L.M.Garder, and M.E.Høvin, "Robot Gaits Evolved by Combining Genetic Algorithms and Binary Hill Climbing", Proceeding of: Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006
6. G.Capi, Y.Nasu, L.Barolli, and K.Mitobe, "Real Time Gait Generation For Autonomous Humanoid Robots: A Case Study For Walking", Robotics and Autonomous Systems 42 (2003) 107-116.
7. Sheng-Wu Xiong, Yong-Xiang Zhao, and Ning Xu, "SAREC-GA: a security-aware real-time scheduling algorithm with genetic algorithm", Proceedings of the 6th International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007.