A General Method for Selection Function Optimization in Genetic Algorithms

Nawar Ismail and Matthew Demers

Abstract Genetic algorithms are often used as a mechanism to solve complicated problems in optimization. In the schemes that we are concerned with, a population of members, which are each defined by a set of parameters, are used with the desire to optimize some value called the *fitness*. The fitness of each member in a population is measured and used during a *selection* process which defines a likelihood for any member to carry on to the subsequent iteration (often called a *generation*) of the algorithm. *Mutations* are then stochastically applied to the population. This alters the parameters of the population members. Combining the effects of selection and mutation tends to increase the average fitness of a population. Our principal concern is in determining how to select members from one iteration to the next. Measuring how well a selection mechanism performs is computationally demanding, making its optimization difficult. We apply an additional genetic algorithm to a simplified model to give an approximate optimization for the selection mechanism. In this paper, we detail the general procedure for this optimization.

Keywords Genetic algorithms · Iterative methods · Optimization
Predictive models

1 Introduction

A genetic algorithm (GA) is, loosely, an iterative scheme designed with the purpose of finding an optimal solution to a potentially very difficult or complex problem. In general, large numbers of difficult evaluations impose time constraints. Many techniques have been developed to improve the utility of the algorithm, such as functional approximation and determining representative simulation run length which both reduce the difficulty of evaluation, and fitness estimation which can reduce

N. Ismail · M. Demers (✉)
University of Guelph, 50 Stone Rd E, Guelph, ON, Canada
e-mail: mdemers@uoguelph.ca

N. Ismail
e-mail: nismail@uoguelph.ca

D. M. Kilgour et al. (eds.), *Recent Advances in Mathematical and Statistical Methods*, Springer Proceedings in Mathematics & Statistics 259,
https://doi.org/10.1007/978-3-319-99719-3_4

evaluation numbers [1–3]. Additionally, the operators used can be designed to produce improvements more efficiently [4]. Finally, the parameters used in the algorithm strongly influence the success of the output but the ideal values are often difficult to determine [5, 6]. In this paper, we are concerned with optimizing a particular parameter of these algorithms, the selection function (see Sect. 3) with a generally applicable technique.

There are many parameters in genetic algorithm to be chosen, such as the number of members in a population, mutation rates, selection probabilities (as well as application specific parameters). Often, these values are chosen through trial and error, or "experimentally" [6]. Finding optimal parameters is difficult due to number of possibilities and the generality of problems tackled by GAs [6]. This difficulty suggests the use of a GA to optimize the parameters of the original GA. However, directly applying a GA to the output of another GA would require an infeasible amount computation time. We propose that a model can instead be used to optimize the parameters of the algorithm.

We apply our investigation to a particular optimization problem; however, we maintain that the procedures presented here are generally applicable. The goal of our optimization problem is to obtain configurations of so-called "creatures" that maximize their displacement on a flat planar surface by the end of a fixed time. These sets of mechanical components operate in a physically simulated environment. Many similar optimizations involving virtual creatures have been studied [7].

2 Optimization Setup

2.1 Creatures

In our framework, we define a creature as a set of mechanical components and how they are connected. There are three possible component types in any creature: pistons, rigid bars, and contact spheres, which will be referred to as *muscles*, *bones*, and *nodes* respectively to remain consistent with the biological naming associated with *genetic* algorithms.

The nodes act as anchors for the connecting muscles and bones, and are the source of environment interaction. The bones will maintain a fixed length, while the muscles oscillate their length sinusoidally with time, at different rates. With these components, each creature will travel a deterministic displacement at the end of a time, $t = t_{max}$. The projection of this displacement on the plane is what we take to be the creature's *fitness*,[1]

$$f = \sqrt{x_{com}(t_{max})^2 + y_{com}(t_{max})^2}, \qquad (1)$$

[1]The goal is to travel across the plane. The height of a creature plays no (direct) role in this. So only the displacement in the x-y plane is considered.

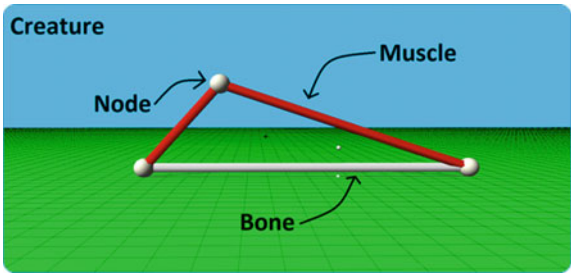

Fig. 1 A creature consists of three components: muscles (red) which provide a potential driving force, bones (white) which provide structure, and nodes which interact with the environment. These define a sufficient set of components that allows a creature to move, provided its configuration allows it. A 2D creature is shown for simplicity, but simulations are run in 3D

where x_{com} and y_{com} are the center of mass coordinates. This fitness is the objective function that we would like to maximize. For simplicity, the muscles and bones are massless, making the nodes the only massive component (Fig. 1).

2.2 Evaluating Fitness

The are six forces responsible for the net force on each node are gravity, muscle forces, bone forces, surface collision, ground friction and drag.

The net force on a node is then the sum of these six forces; namely:

$$\mathbf{F}_{node} = \mathbf{F}_g + \mathbf{F}_m + \mathbf{F}_b + \mathbf{F}_c + \mathbf{F}_f + \mathbf{F}_d. \tag{2}$$

From $\mathbf{F} = m\ddot{\mathbf{r}}$, we get that

$$\mathbf{r}_j(t) = \frac{1}{m_j} \int\int \sum_{forces} \mathbf{F}_{node,j}(t, \mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{max}) dt^2, \tag{3}$$

where $\mathbf{r}_j$ is the position of node j, and $\mathbf{F}_{node,j}$ is the net force acting on that node. This coupled with Eq. 1 defines the mathematical function we are trying to optimize.

We simulate the movement of each node by iteratively evaluating this equation numerically. At the end of each simulation (consisting of 9000 unit time step iterations), the displacement in the plane of the creature's center of mass is recorded as the creature's fitness.

3 Algorithm

3.1 Genetic Algorithm

Our genetic algorithm begins with an initial population of N members.[2] In our scheme, each member of the population is a creature as defined in Sect. 2.1. *Genetic operators* are then iteratively applied to the population. These operators act on a population with the aim of increasing the average fitness of that population [8]. The operators used are the selection function, and the mutations (see Sect. 3.2). An outline of the algorithm used can be found in Algorithm 1.1. The algorithm can terminate on many different end conditions. We mainly end our simulations when improvements become negligible or at the end of a fixed number of generations.

Algorithm 1.1 Genetic Algorithm

```
$P \leftarrow$ INITIATEPOPULATION( )                ▷ with creatures
while end condition not met do
    EVALUATEFITNESSES($P$)
    $P \leftarrow$ SELECT($P$)
    $P \leftarrow$ MUTATE($P$)
end while
```

3.2 Genetic Operators

3.2.1 Selection Function

Selection functions act on a population to select creatures from one iteration of the algorithm to the next [8]. There are several types of selection functions [8]. We specifically investigated a type of rank selection. After fitness evaluation, creatures in the current generation are ordered from highest to lowest fitness. The creature with the greatest fitness would have rank 0, the creature with the second greatest would have rank 1, and so on. We could also consider a rank percentage which is bounded by 0 and 1, regardless of population size.

We must balance the variance of the selection function with how strongly we select for the best creatures. With too little variation, the probability of being trapped at or near a local maximum would be very high [8]. Conversely, selecting uniformly (without regard for fitness) would be a pointless exercise. Determining how to distribute this balance in our selection function is our principal concern.

[2]Values of N between 100, and 1000 are typically used. This was determined through trial and error. Future work may determine an ideal value through the techniques described here.

3.2.2 Mutations

At the conclusion of each generation, all creatures may undergo one or more *mutations*. Mutations alter the properties of a creature, which ultimately affects its fitness. There are three mutation types in our scheme:

1. **Modify Characteristic**: Changes a property of a component. Examples include: random node locations, modifying node mass, and modifying muscle contraction rate.
2. **Add Component**: Add a node, muscle, or bone. Adding a node may generate additional connections.
3. **Remove Component**: Remove a node, muscle, or bone. Removing a node will also remove connected muscles and bones.

These essentially span the set of simple alterations, and provide a mechanism to explore other creature configurations [8]. These are applied to creatures with some small probability. Our simulations indicate that most mutations will decrease performance, but a handful will cause improvements. Coupling mutations with the selection function means that previously successful creatures are being modified, causing improvements in their design over time.

4 Methods

4.1 Overview

To optimize the form of the selection function, it must be assigned a fitness. We use the average fitness of the creatures produced after 300 generations to represent the ability of a selection function.[3]

A statistical model of our creatures will be used to reduce the computational cost required to evaluate the fitness of many selection functions. This model is implemented by replacing creatures with their most representative statistic: their fitness. The model will therefore deal with floating point numbers instead of a complicated structure whose fitness is computationally expensive to measure. This is a sort of *functional approximation*, where we use an alternate expression for the fitness [3].

The selection function acts identically except it considers fitnesses rather than creatures (which would have those *associated* fitnesses). The mutation operator however, is intimately tied to the physical design of a creature. So reducing its structure to a single value requires some careful considerations.

[3]This must be evaluated several times to obtain a proper statistic, which can be demanding.

4.2 Estimating Impact

To emulate the mutation operator, we look at the distribution of how the mutations tend to affect the creature's fitness. It is straightforward to see that the change in fitness after a creature is mutated (or *impact*), correlates to the fitness of the creature. For example, creatures with a greater fitness would be likely to suffer negative effects when a mutation occurs due to disruptions in their more specialized structures.

Combining the fact that each creature has its own distribution, with the fact that these distributions depends on fitness, we conclude that each fitness has its own distribution. To properly mimic it, we collect a sample of genomes with various associated fitnesses, and apply this sampled distribution to our model.

We start by collecting a list of impact statistics. To do this, we first determine a set of fitness levels that span the range of fitnesses typically seen. The genetic algorithm is run until it produces a creature with a fitness within some margin of a desired level. Mutations are applied to several copies of this creature. Measuring the change in fitness for each creature provides us with a sample of impact statistics. If we sample these impacts from many different creatures at each fitness level, we obtain a reasonable sample of impact statistics grouped by fitness level.[4] This is outlined in Algorithm 1.3 and gives us the required statistics for our model.

The fitnesses of the creatures typically range from 0-400. Based on our available computational time, we chose our levels to be at $5 * 1.25^i$ for $i = 0, \ldots, 20$. 1000 impact statistics were collected from each of over 1800 creatures spanning these fitness levels, providing us with nearly two million impact statistics.

Algorithm 1.2 Obtaining Impact Statistics

```
for all Fitness levels do
    for $M$ creatures found at this level do
        $Population \leftarrow N$ copies of creature
        $f_0 \leftarrow$ initial creature's fitness
        for all $N$ creatures do
            apply round of mutations
            $f \leftarrow$ new fitness
            record impact statistic as $f_0 - f$
        end for
    end for
end for
```

[4]Care should be taken as to not sample multiple creatures from the same instance of the algorithm. Otherwise they will not be independent.

4.3 Model

Our model aims to approximate the fitnesses produced by the genetic algorithm that simulates creatures, without actually simulating them. The utility of this is not creature optimization (at least not directly), since we remove any concept of creature, but instead to greatly decrease the computational cost associated with measuring the fitness of a selection function.[5] In our model, when the mutation operator is called on a population of fitnesses, the fitness of each member is looked up and a random impact corresponding to that fitness is applied to their fitness. Since the distribution of impacts is contained in our sampling, this approximates the true nature of impacts.[6] These changes to Algorithm 1.1 are shown in Algorithm 1.3.

Algorithm 1.3 Model Genetic Algorithm

```
function IMPACTMODEL($f$)
    impacts ← dataset with fitness closest to $f$
    return uniformly selected value from impacts
end function

$P$ ← INITIATEPOPULATION( )        ▷ with fitnesses sampled from initial populations of creatures
while end condition not met do
    $P$ ← SELECT($P$)
    for all population members do
        $f \leftarrow f$ + IMPACTMODEL($f$)
    end for
end while
```

To demonstrate the validity of our model, we run both algorithms (the one which simulates creatures, and the one which only considers their fitness) and measure the fitness of the selection function (as described in Sect. 4.1). This is done for a selection function of the form,

$$P(x) = (1 - x)^p \sin(\pi x) \tag{4}$$

for several different values of p. This function was based on our intuition for how the mass of the selection function should be distributed - essentially a reasonable guess at a good selection function. As can be seen in Fig. 2, our simplified model follows the general trend obtained by the algorithm which actually simulates the creatures. Since the trends are similar, the maxima in both the real simulation and our model would likely occur for similar selection functions.

[5]This decreased cost will then be used to optimize the selection function, which in turn improves the creature optimization.

[6]More advanced statistics or added corrections can be implemented to improve the quality of our model. As a basic implementation this will suit our purposes.

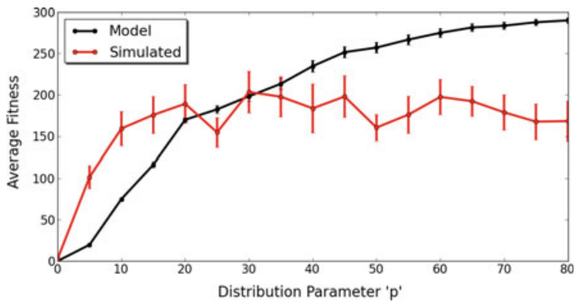

Fig. 2 A comparison between the actual data obtained from simulating the creatures (red) and the models predictions (black) for a selection function of the form, $P(x) = (1-x)^p \sin(\pi x)$. Each point corresponds to the particular selection function's fitness. Notice the relatively large fluctuations found in the simulated data. This results not only from a large variance at each point, but also demonstrates a limitation in acquiring large data sets due to the high computation cost that we aim to eliminate. Under the considerations that our model **greatly** simplifies the problem (by eliminating the creature), the two graphs follow a similar trend and so we validate it as an approximation to the actual selection function fitness

4.4 Selection Function Optimization

We can now approximate the fitness of a selection function in a feasible time scale which allows us to optimize the selection function. To avoid assuming the form of the ideal selection function, we define the function,

$$\sigma_n(a_1, a_2, \dots a_n) = \frac{n}{\sum a_i} \cdot \sum_{i=0}^{n} \begin{cases} a_i & \frac{i}{n} < x < \frac{i+1}{n} \\ 0 & else \end{cases}, \tag{5}$$

which corresponds to a normalized set of n columns of height $a_i n / \sum a_i$ that are equally spaced on the domain [0, 1]. With sufficiently large n, this function can be used to approximate any function we would be concerned with, and so it is used as our selection function.

We use a third genetic algorithm to determine the parameters, a_i of the selection function σ_n, to optimize our model of the fitness of a selection function.[7] We fill our population with 300 members. Each member has a set of n numbers corresponding to a_i in 5. The selection function used in this third algorithm is $P(x) = (1-x)^{27} \sin(\pi x)$, since we have seen its validity when optimizing creatures.[8] Mutations consist of potentially reassigning some numbers with new random values. With this, everything needed to optimize the selection function is set in place.

[7]The other two being: (1) the one used in creature optimization, and (2) the one used in our model.

[8]One could consider optimizing *this* selection function as well. However, they would find themselves optimizing endlessly. At some point an educated guess must be made.

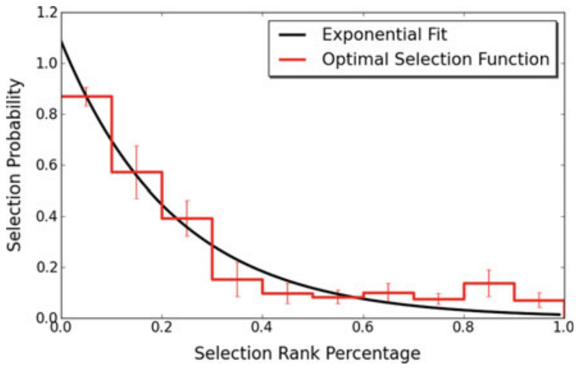

Fig. 3 The average σ_n function outputted by the selection function optimization procedure (red). The least squares exponential fit $P(r_p) = 1.08e^{-4.44r_p}$ was fitted to the σ function (black). Here, $P(r_p)$ is the probability of a creature with rank percentage r_p to be selected. We chose $n = 10$, but higher values of n can be used, although this may make the optimization more difficult. Either the σ function or the fitted curve can be taken to be an optimal selection function

5 Results

Our model aimed to approximate the fitnesses produced by the optimization of creatures. It was able to generate trends similar to those that resulted from actually simulating the creatures. This is shown in Fig. 2.

Taking our model to be a reasonable predictor of a selection function's fitness, it was used in an additional genetic algorithm. This genetic algorithm optimized the parameters in the selection function defined by Eq. 5. The optimal selection function that was produced was fitted to the exponential seen in Fig. 3. When this selection function was used to optimize creatures, the selection function's fitness was 230 ± 10. Although this is likely not the highest possible fitness, it does surpass all other tested selection functions. The best of those selection functions produced a fitness of 200 ± 20. This means that the fitness of our optimized selection function increased by $(15 \pm 2)\%$.

6 Discussion and Conclusion

Our primary concern was optimizing the selection function used in our optimization of creatures. To overcome the excessive computational cost associated with this, we developed a model capable of approximating selection function fitness. Our model removed the need to simulate creatures by only considering their best representative estimator, their fitness. To allow for this simplification, the mutation operator was approximated by using a sample of impacts at various fitness levels. With this, our model matched the general trend obtained when actually simulating the creatures, validating our model as an approximation.

The model was used to optimize a selection function with a general form. An ideal solution was found to be an exponential decay and produced $15 \pm 2\%$ higher creature fitness on average after 300 generations than other tested functions. The generality of our method, would allow it to be applied in many different optimization problems.

The determination of the selection function form would be an unaccessible task without similar considerations to those presented here. Brute force optimizations can still be done, but may be limited to considering a handful of values [9]. The most computationally demanding task in our technique is the acquisition of a sufficient sample of creatures. Collecting the $\sim$2000 creatures used here required around 10 days of computation (on a single machine). However, we observed similar results with $\sim$200 creatures, implying some robustness with regard to quantity. In addition, the collecting of genomes - which amounts to recording the solutions to the problem - has other uses like analysing solution behaviours and their characteristics, and can be collected passively as the problem is studied. It does not have to be the focus of the optimization, and can this technique can be applied after a sufficient set is collected.

Future work could relate to improving the statistical methods applied to our model, the simplicity of which amounts to our largest source of error. One could also consider extensions such as: if and how the selection function should change as the algorithm iterates, optimization of population size, or ideal mutation rates, as considered in [6].

References

1. Sun, C., Zeng, J., Pan, J., Xue, S., Jin, Y.: A new fitness estimation strategy for particle swarm optimization (2013)
2. Branke, J., Asafuddoula, M., Bhattacharjee, K.S., Ray, T.: Efficient use of partially converged simulations in evolutionary. Optimization (2017). https://doi.org/10.1109/TEVC.2016.2569018
3. Regis, R., Shoemaker, C.: Local function approximation in evolutionary algorithms for the optimization of costly functions. IEEE Trans. Evol. Comput. **8**, 490–505 (2004). https://doi.org/10.1109/TEVC.2004.835247
4. Rasheed, K., Hirsh, H.: Informed operators: speeding up genetic-algorithm-based design optimization using reduced models (2000)
5. Eiben, A.E., Smit, S.K.: In Autonomous Search, p. 1536. Springer, Torino (2011)
6. Aine, S., Kumar, R., Chakrabarti, P.P.: Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off (2009)
7. Lehman, J., et al.: The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities (2018)
8. Kumar, R.: Blending roulette wheel selection and rank selection in genetic algorithms. Int. J. Mach. Learn, Comput (2012)
9. Pongcharoen, P., Hicks, C., Braiden, P., Stewardson, D.: Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. Int. J. Prod. Econ. **78**, 311 (2002)