

Optimization of Genetic Algorithms

Travis Hoppe – Drexel University – Summer 2006 Research Paper

The aim of this paper is to propose an extended methodology building upon the basis of genetic algorithms (GA). An introduction to the basic steps of function optimization is given in the form of the method of steepest descents. The limitations to this method and others similar to it are shown. While many algorithms exist for linear systems, we hypothesize a general non-linear result using genetic algorithms. GA's are then introduced in a framework that will be expanded beyond its traditional boundaries. Instead of seeking the solution itself, we propose that new information can be gleaned by maximizing the *method* of solutions to a similar class of problems.

Introduction:

We begin with some terminology. Throughout this paper we will denote $f(\chi)$ to be a function with a set of inputs $\chi = \{x_1, x_2, \dots, x_k\}$. The function itself is not necessarily single-valued. Define a fitness function Ξ , that maps $\Xi \equiv \Xi(f(\chi))$. We can then define optimization in all generality as the maximization of Ξ for a given f by varying χ .

Gradient Method, Method of Steepest Descents:

Often times the problem can be recast by simply letting Ξ be the maximization¹ of f . In an ideal system the form of f can be written down by a series of elementary functions (i.e. e^x , $\sqrt{y^2 + x^2}$, ...). If this is the case, the simplest and most pedagogically useful illustration of optimization can be illustrated by the method of steepest descents. Since the function is known explicitly we can calculate the gradient $\nabla \cdot f$. The physical interpretation of this vector is the direction of the greatest change along the fitness landscape. Simply by making incremental changes in χ , governed by the gradient, we can find a minimum of f .

While technically correct there are several caveats to this method. As long as it exists, a minimum is guaranteed to be found. There is no guarantee however that this point is the *global* minimum, but merely a *local* minimum. Various methods exist to rectify the

¹ Since the fitness function Ξ is user-defined, is not necessarily the maximization of f . It could, for example, be the sum of input terms as well as the function itself, i.e. $\Xi = f(\chi) - \sum x_i$

situation. The simplest merely involves choosing a different initial point to iterate on, with the chance that the solution will settle on different minima. The other methods are variations of simulated annealing. In these methods a perturbation is introduced along the iterations to disturb the solution out of any local minima.

In practice, the method of steepest descents works fairly well but constrains f to be a function whose explicit form is known. The more advanced cousin of steepest descents, the conjugate gradient method, is more robust but requires that that f be expressed in the linear form of a positive definite matrix. In many physical problems the solution sought will be linear (or at the very least, approximated by a linear system). For example, a large class of partial differential equations can be cast in this form. Nevertheless, it is quite possible to encounter a function that is both unknown and decidedly non-linear.

Introduction to Genetic Algorithms:

As a whole, genetic algorithms are only understood in specific well defined contexts. Since they still find novel uses in the field (despite being introduced more than 30 years ago), the nomenclature and techniques are still evolving. This has the result of nonstandard terminology. The notation used in this paper is a case-in-point.

Typically genetic algorithms perform well when the structure of f is unknown. When assumptions about f can be made, i.e. linearity, tailored algorithms can be applied and are typically much faster. Genetic algorithms, as traditionally applied, make no such assumptions and offer extensive flexibility.

Formally, we can introduce genetic algorithms as follows:

An *agent* is a selection of inputs χ to the function f . Each agent has an associated fitness given by the mapping $\Xi_i \equiv \Xi(f(\chi_i))$. There is a *population* of agents $P = \{\chi_1, \chi_2, \dots, \chi_m\}$. The population itself has a fitness defined by $\Xi'(P, \Xi) \equiv \max\{\Xi_1, \Xi_2, \dots, \Xi_m\}$, or in other words, the most fit agent of the population. The population evolves over iterations by applying operators (such as

mutation and *crossover*) governed by a set of parameters Λ . The algorithm terminates when the population has reached a desired level of fitness, $\Xi' > \varepsilon$ or a set number of iterations, n , has been reached.

Mutation and crossover are, in general, tricky to formally define. Crossover, or recombination, takes the various elements of an agent, and recombines them with another agent. Exactly how the crossover is done is highly dependent on the data structure of χ itself. In the first representations of genetic algorithms, agents were simply binary representations of χ , so crossover was simply bitwise switch. If χ contained cardinal numbers and agents reproduced these representations, crossover could involve swapping, averaging (weighted or unweighted), or any number of various combinations. Mutation involves changing an individual agents' expression according to parameters in Λ . Typically a mutation is a random perturbation with an effect similar to simulated annealing.

The parameter set Λ that governs the iteration operators contains all the information the system needs to evolve. A small sampling of Λ parameters in typical GA problem consists of:

Crossover/Mutation rates – Governs number of times crossover and mutation operators are applied each iteration.

Crossover weighting – A selection rule for crossover (i.e. preferential treatment to agents with greater fitness).

Influx of new agents – Often introduced as an extreme form of mutation, helps keep the population from converging to local minima and prevents genetic drift.

Extensions to Genetic Algorithms:

Indeed there are many more parameters that could be included into Λ . Attempts that have been made to classify and generalize Λ have been met with limited success. It seems that there are similarities in certain classes of problems. This, I believe, is a crucial observation for the improvement of genetic algorithms.

I propose that the classifications intrinsically contain information. Furthermore, the classifications themselves should be done by genetic algorithms in a hierarchal form. In other words, a separate set of genetic algorithms (call them *breeders*) should evolve the parameters to help other genetic algorithms evolve. The classifications can be grouped by the parameters governing the breeders.

For clarity, let Λ be an agreed upon set of breeding parameters (crossover/mutation rates, etc...) and Λ_j be a specific selection of these parameters (20% mutation, etc ...). Each iteration of the system can be thought of schematically as $P_i = P_{i-1} \otimes \Lambda_j$, i.e. each iteration is the result of the previous one acted on by the set of breeding criteria.

Since we are allowing for different breeding parameters the population fitness must reflect this, thus $\Xi'_i \equiv \Xi'_i(P_i(P_{i-1}, \Lambda_j), \Xi)$. We can now define a *breeding fitness* which is a measure of population fitness for n fixed iterations, $\Xi'' \equiv \Xi''_n(P_n(P_{n-1}, \Lambda_j), \Xi)$.

With the breeding fitness defined, we now proceed to find an optimal Λ_j . This optimal set of breeding parameters, call it Λ_ω , is defined such that $\Lambda_\omega \equiv \max(\{\Xi'', \forall \Lambda_j\})$. Thankfully since the framework of genetic algorithms are already in place, the problem of finding Λ_ω is simply another optimization problem (where the components of Λ_ω are χ and the fitness function is Ξ''). These *breeders* are then GA's whose purpose is not to optimize f , but to find the set of parameters such that a separate set of GA's can optimize f *in the most efficient way possible*.

Before we proceed, it will help to take a step back and see what we have accomplished. In calculating Λ_ω , we have optimized the optimization of f in the present framework. This clearly is a computationally involved process. Whatever the complexity of a single GA is, the calculation of Λ_ω requires that it is that complexity squared. After the calculation, there is no immediate benefit of Λ_ω , since the problem has been solved

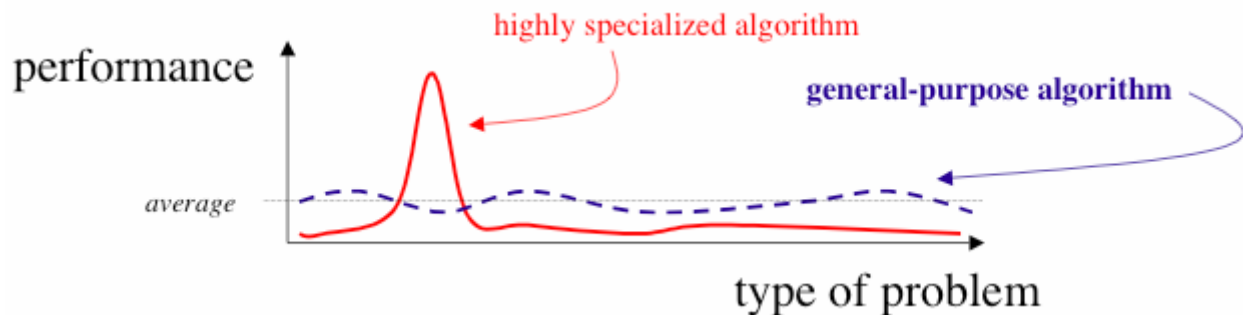
exhaustively at this point. While not explicit, many papers in the literature today attempt to calculate a variant of Λ_ω by adjusting breeding parameters by hand. Their goal is find various qualitative rules that can be applied to other, different functions. This is exactly what I propose with my method as well. The difference is that Λ_ω is a qualitative, repeatable measurement for any specific f .

Suppose now that we have at our disposal hundreds of various optimization problems. In biological physics there are numerous examples such as protein-folding, signal pathways in cellular biology, cellular structure under various transformations and so on. Each problem can be treated as a black-box defined only by its f and Ξ . By running the analysis in the preceding paragraphs $\forall f_i \rightarrow \Lambda_{\omega,i}$. This collection of Λ_ω , call it Ω , is a qualitative measurement defining the optimal breeding characteristics for a wide array of problems. In a comparison of the members of this set, there are two possible observations that would make this extensive analysis worthwhile.

The first such observation of Ω would be uncorrelated results. If each member of Ω bore no resemblance to its peers this has immediate consequences. It would imply that, for all the problems considered in Ω , there does not exist optimal breeding parameters for GA.

The second observation of Ω , and the one that I predict, is a high degree of correlation between small subsets of Ω . If the sample size of Ω is sufficiently large to capture a large spectrum of problems, I believe that certain classes of problems will optimize in similar ways. This is in accordance with the famous “No-free-lunch theorem” by Wolpert and Macready. It states that, on average, a general optimizer will perform just as well as a specialized algorithm when applied to all optimization problems. Even if, on the surface, f_i and f_j bear no resemblance to each other, by having similar Λ_ω , there must be an underlying structure of the fitness landscape that unites the two.

If, optimistically, a large percentage of Ω 's members can be grouped into a few small subsets of similar Λ_ω , we have a tangible, usable result for future research. If a researcher has a new function that is similar to those included in Ω , he need not to choose Λ_j blindly. This ad-hoc method, which seems to be prevalent due to lack of reliable Λ_ω , can be replaced with one of the optimal Λ_ω subsets. Since the number of optimal Λ_ω are small, only a few must be tested and observed. There is a high probability that the researcher will be rewarded with a 'specialized' algorithm, a GA tailored to match the fitness landscape of the problem.



An illustrated view of the No-Free-Lunch thrm. - <http://en.wikipedia.org/wiki/Image:Nflt.png>

Since the process was designed to be as general as possible (the only constraints were that all functions had the same *types* of parameters in its Λ), as more problems are added to Ω , the scope of each generalized Λ_ω grows accordingly. Purely in the speculative region now, one could envision an automated system where a function from a published paper (along with its fitness criteria), could be uploaded onto a central repository. This would allow for collaborative discourse on the relations between various members of a subset of a generalized Λ_ω . By looking at the specific problems in each subset, it is possible that there may be obvious relations to each other (i.e. they all may be 2nd order-differential equations). If there is no obvious relation between the members of a particular subset, the question of a quantifiable similarity in their fitness landscapes becomes interesting and well-defined. Until a large Ω is assembled encompassing a broad spectrum of problems, the question remains. Currently only fragmented answers are spread out over a swath papers with non-optimal Λ_ω and small Ω .