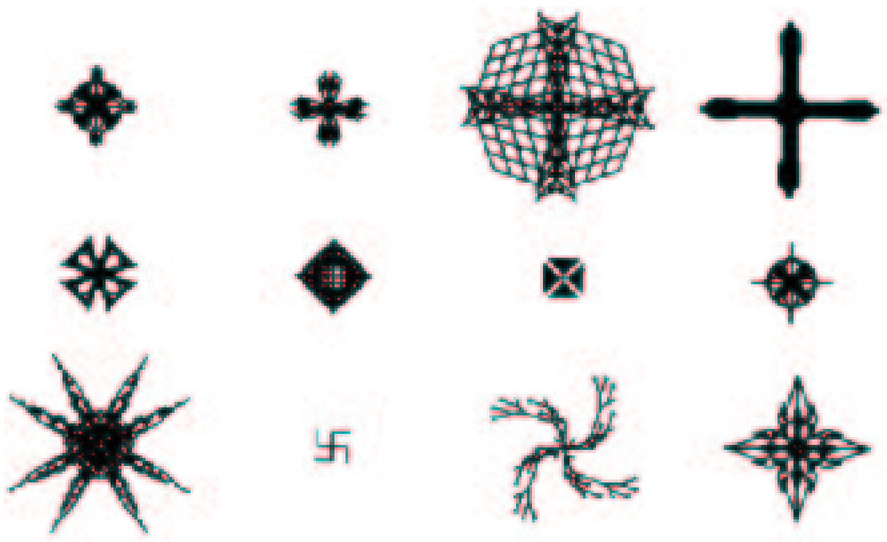


GENETIC OPTIMIZATION

Carl A. Pearson

Acknowledgements

Research overseen by Dr. Robert G. Brown,

Duke University Physics Department

Supported by NSF VIGRE grant number DMS-9983320

Presentation generated by LaTeX

Title picture courtesy of Richard Dawkins' *The Blind Watchmaker*

1 INTRODUCTION

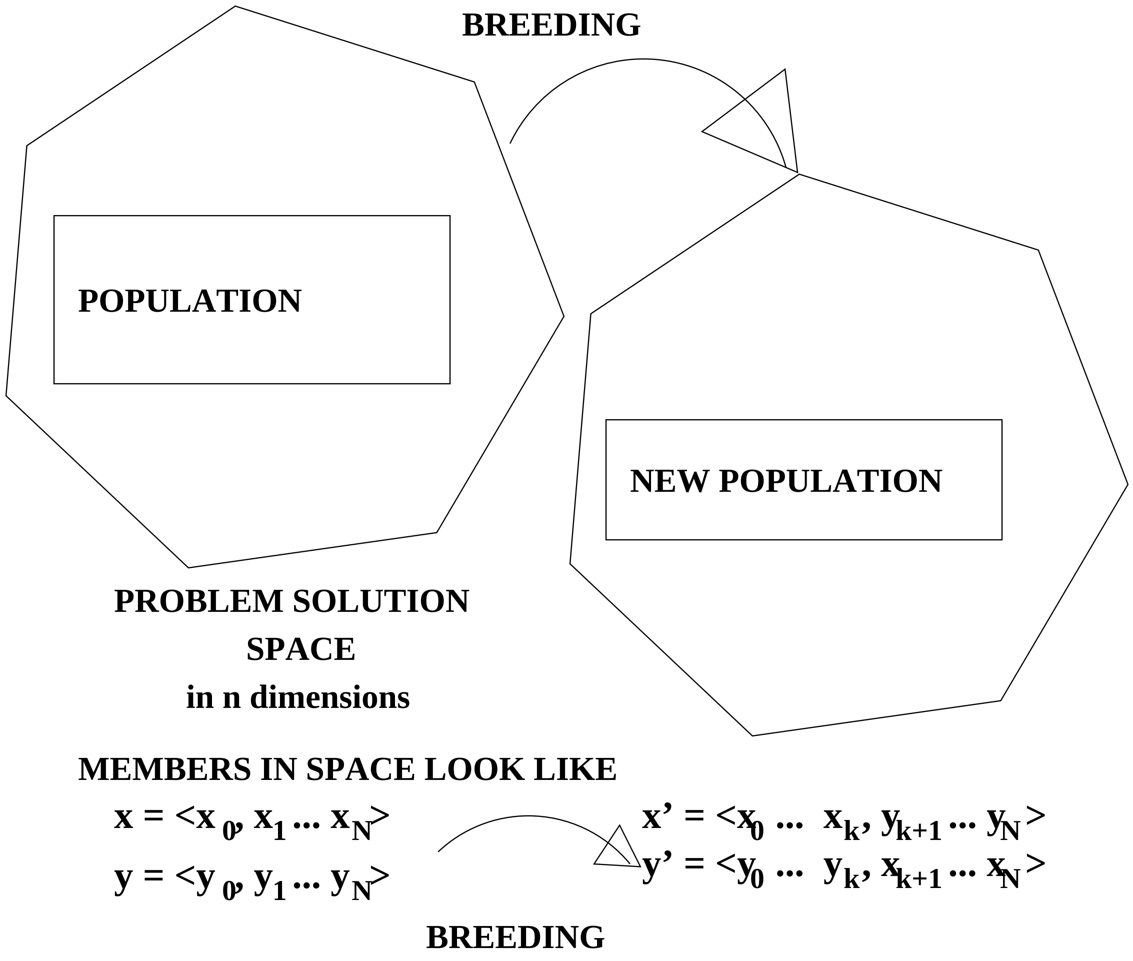
1.1 What are Genetic Algorithms?

1.1.1 The Archetypal Simple Genetic Algorithm (SGA)

An SGA addresses a Problem Solution Space P with n dimensions in arbitrary other spaces (Integer, Real, Complex, Alphabet, Colors, *et cetera*) by generating an initial population P_0 such that $P_0 \subset P$. The SGA has a fitness function f such that $f : \mathbf{x} \longrightarrow \Re$, where $\mathbf{x} \in P_0$. The next generation P_1 is created by assigning each $\mathbf{x_i} \in P_0$ a probability of reproduction p_i such that

$$p_i = \frac{f(\mathbf{x_i})}{\sum_j f(\mathbf{x_j})}$$

The reproducing members are then chosen randomly according to this probability, and the new members of P are created by randomly pairing the reproducing members and exchanging portions of their values.


BREEDING
POPULATION
NEW POPULATION
PROBLEM SOLUTION SPACE in n dimensions
MEMBERS IN SPACE LOOK LIKE
x = <x 0, x1 ... xN>
y = <y 0, y1 ... yN>
x' = <x0 ... xk, yk+1 ... yN>
y' = <y0 ... yk, xk+1 ... xN>
BREEDING

1.1.2 General Definition of GAs

The general definition of a GA is: the ordered application of a set of genetic operators (breeding, mutation, and various extensions of them) in accordance with a fitness function and in an iterative fashion upon an initial population subset of a problem solution space, where genetic operators take n-members to m-members (breeding with n parents and m offspring) in the problem space or take a member to another member in the space (mutation). Alternately, one could think of a GA a function operating on a fitness function, a problem space, and a number of iterations which evaluates to the best member (according to the fitness function) in the problem space reached in the number of iterations given. Essentially, a GA is an algorithm that given a problem solution space, generates a initial population randomly, then successively produces new populations from samples of the previous one based on the fitness of those being sampled. Like the SGA, fitness plays a strong role in determining each successive generation, but not necessarily by flooding the population with the strong.

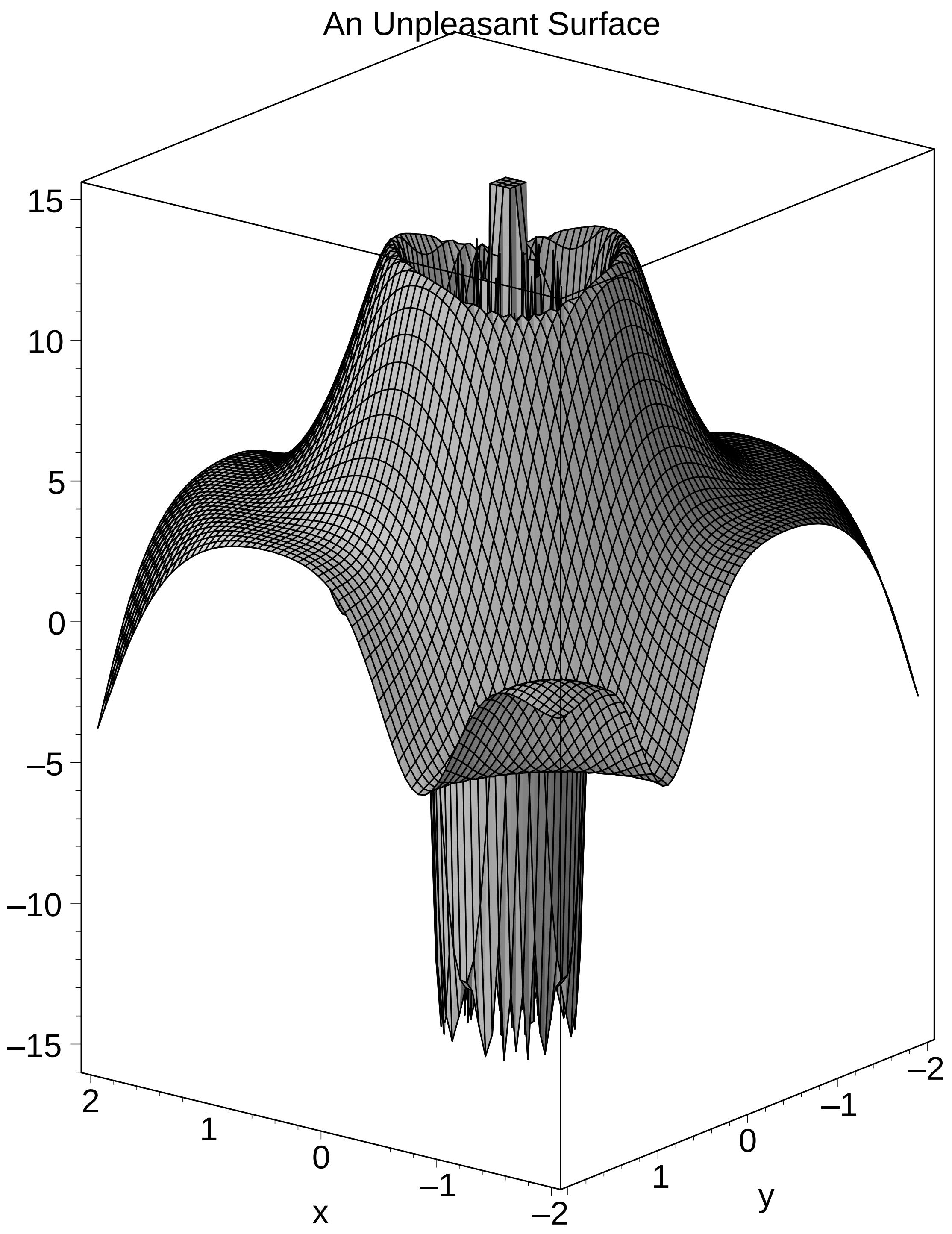
1.2 What are some problems GA's solve well?

1.2.1 General Problem Types

Generally speaking, GAs solve most problems equally well, so it is more informative to ask what problems GAs solve better than other algorithms. Essentially, GAs maintain the ability to sift problem spaces no matter what degree of complexity is associated with those spaces, provided they have the adequate memory resources available to represent the population. Furthermore, when convergence occurs, i.e. mutation becomes the dominant factor in improving the solution, a GA has become a Simulated Annealing (SA) process on the members of the population. Given that SA always converges to the optimal solution given "enough" (though potentially infinite) time, we can say the same for GAs with a mutation component. Obviously, GAs are going to be inferior to specific solving algorithms, since those specific algorithms take into account information about the problem. Problems that have no algorithmic solution (aside from enumerative searches) are the fare of GAs, especially NP-complete problems; specific heuristics will typically outperform GAs in smaller n cases, but once outside of their upper bounds the GAs continue to perform while the heuristics become lost.

1.2.2 Specific Examples

A few classical examples of NP-complete problems that GAs move through significantly faster than simple (or even heuristic) searches include the N-queens problem, the Traveling Salesman Problem, Maximizing returns on Prisoners' Dilemma over a large number of games, and circuit design problems. Additionally, construction of neural networks is usually done initially with a GA. Metastatistics on these problems indicate that the solutions obtained are among a small top percentage of possible solutions, where how small "small" is depends on the population size and generations, combined with what sort of breeding and mutation algorithms are used. GAs are also capable at solving non-NP problems, though since an optimum solution is not guaranteed for a given number of generations in any particular run, GAs are used only in cases where an absolute optimum is not required and timeliness of return is preferred.



2 WORK UNDER PRUV FELLOWSHIP

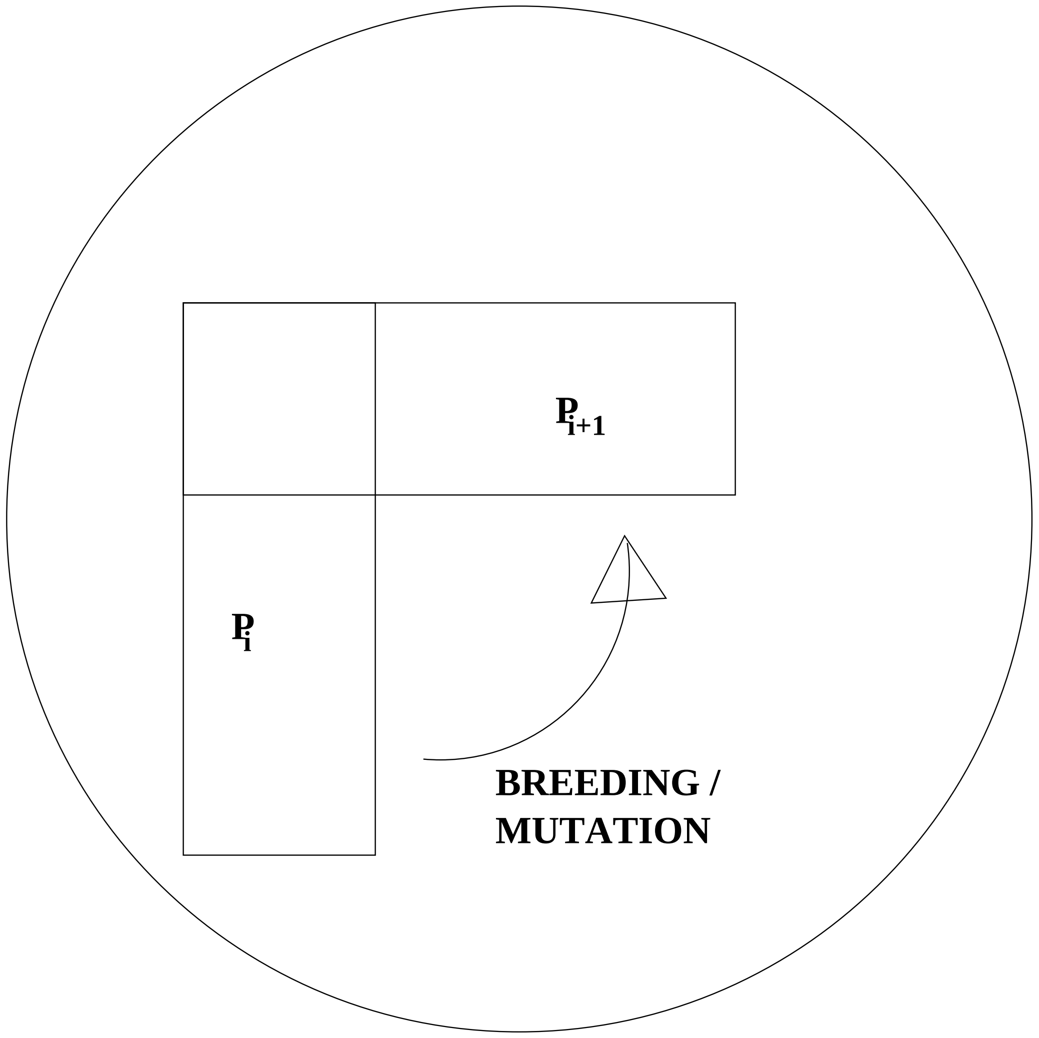
2.1 Problems with SGA

The primary problems with SGA are ones of convergence of the population; either the algorithm is to slow to converge to a good solution or it converges too quickly before it can find an optimal solution. These are the sort of problems found in real biological breeding program; good traits are present in the fit as well as the weak, and a program that too ferverently seeks to eliminate the weak will also lessen the chances that those good traits drift "up" into the rest of the population. However, the limitation of the weak is key in allowing the strong to maximally utilize resources (the GA analog being computational resources). The primary problems with SGA are the strong bias against the weak members of the population (the new population is choosen overwhelmingly from the strong via probability) and the action of the search when the population becomes essentially uniform (works little better than an extremely hampered SA with low energy corresponding with the low mutation rate). What modifications could be made to the SGA (both in the abstract and in the implementation) such that these problems can be addressed?

2.2 Possible Solutions - The Generalized GA (GGA) and Modularity

2.2.1 The GGA as a Problem-Independent Solver

What is needed is a program that adequately fulfills the general definition of genetic algorithms, but is also capable of very specific action, so as to take advantage of knowledge about a given problem. I will refer to such a program as a GGA; its primary means of operation, for the time being, is achieved by programming it in layers. A perl script is given a information corresponding the disposition of the problem space in the form of a file; from that, the script writes the appropriate h and c files for the member creation/copying/etc functions. The script then compiles the necessary files and runs the program; optionally, the script may take information about what sort of breeding/mutation/etc modules to link in as well. This implementation allows for a user to enter nothing more than the bare boundaries of the problem Given this basic premise and implementation for the GGA, let us consider some interesting possible modules for coding and statistical analyzation.



PROBLEM SOLUTION SPACE

$$x\,,\,y \;\rightarrow\; z = \langle (x_0 \text{ or } y_0),\ (x_1 \text{ or } y_1) \ldots (x_N \text{ or } y_N) \rangle$$

MEMBER BREEDING

2.2.2 Generation Modules

- Breeding Pools

  Consider divided the population up into breeding pools; either by strictly sectionally dividing them according to fitness of placing them probabilistically into the pools (i.e. the fit members are most likely to fall into the "fittest" breeding category, but also possess some chance to be placed in a lower pool, and the weak are affected in a parallel fashion. These pools may then also have optional breeding sanctions; the strongest pool perhaps has minimized reproductions, since the offspring there will likely survive to repoduce without fail, while the weakest pool perhaps has high offspring and mutation rate but those offspring are more aggressively elimated by the algorithm.

- Introducing notions of Age and Reproductive Vigor

  While this could certainly fall to the higher-order traits section to come, it is discussed equally appropriately here. If our GA maintains members of the population from generation to generation, much like the biological world, then it would also be desirable to elimanate them like biological systems do. The natural way to do so is to give members an age attribute and have a probabilistic killing function that is incresingly effective on members as they age, in addition to the normal probabilistic elimination of the weak. Additionally, one might want to implement a function that limited a member's ability to reproduce as that member aged, or might allow them to have a limited number of offspring much human females are only capable of having a limited number of children.

2.2.3 Mutation Modules

- Differential Mutation

  Differential mutation is designed to deal with the problems associated with GA search once the population diversity becomes limited (ie convergence has occured); it does this by making the mutation algorithm dependent on the standard deviation of the population member or some other measure of sameness. Essentially, we initially desire mutation to be capable of large jumps, but to occur with very small probability. As the population becomes more uniform, namely when a desired solution is being honed in on, we desire mutation to become more likely, but not to jump very far away from the solution area. Differential mutation thus replicates the process of SA in GA endgame searching. On an appropriately implemented GA, we duplicate SA in a parallel fashion, which is desirable when we do not have a known superior algorithm when close to optimal solutions (such as conjugate gradient search).

- Radiation Blast

  As an alternative to Differential Mutation we may desire a stronger form of resampling without taking an actual resampling of the space. To this end, we simply affect a widespread mutation on a large portion of the population (between % 50 and % 75). An adequate portion of the populace remains in place to return to the same maximum fitness value, but likewise "good" portions of a solution are paired with, hopefully, representation from totally new areas of the space. If stronger areas exist away from the one the population had converged to, then it will be possible (though perhaps not likely) that these areas will be reached. In the cases where better spaces exist over relatively small leaps, it is likely that these will be reached and the development of the population will continue.

2.2.4 Higher-Order Individual & Population Traits

- Sexual Differentiation - N-sexes

  The differentiation of the sexes allows for an introduction of endless possibilities to the algorithm; with a changing fitness function (co-adaptive with the population or otherwise evolving), the notions of dominance and recession and pheno- vs genotypes allow for the population to have enduring success when measured against the fitness function. Additionally, we may implementation different conditions for the sexes. Females may be a stable portion of the population with a low mutation rate, while the males can have high mutation rates and be produced abundantly only to be subject to very harsh fitness standards. With N sexual reproduction, each sex may have something special about it; the 3rd sex could have chromosome determining traits for example.

- Tribes

  We may desire to subdivided the problem space into smaller ranges and assign these as individual tribes; these tribes could easily be run on parallel processors with somekind of “cultural” (war, trade, et cetera - whatever metaphor suits ones fancy) interface every so many generations.

2.2.5 Self-Adaption

One distant goal of the GGA is to bring it to the point where it may perform self-tuning in order to not only speed the process of finding optimal solutions, but also to establish certain settings for various problem types which can be used effectively over again as initial parameters.

2.3 Problem Suite

2.3.1 N-dimensional Random Number Guess

This is in and of itself an obvious toy problem; an algorithmic approach can solve this problem in short spans of time by simply guessing above and below the number closer and closer until reaches it. However, if we choose to link some of the values in even a very simple fashion (say one pair want to get as far apart as possible) then the simple search will fail where the GA still succeeds. Certainly we can reimplement a search which will look for whatever it is we have programmed in, but the obvious trail we are heading down leads toward a problem where we do not know what kind of connections against which we are searching.

2.3.2 Linear System Solver

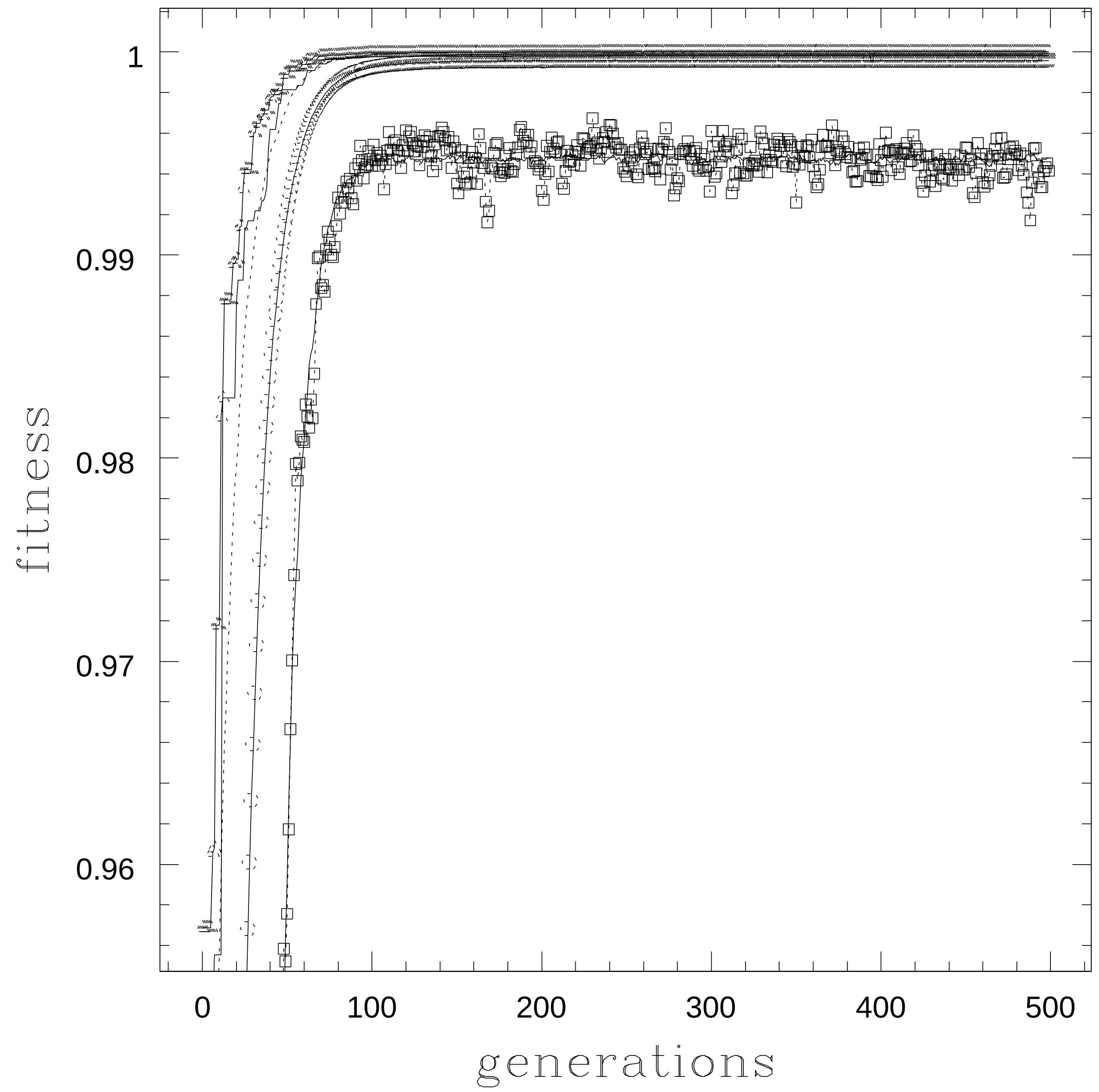
The Linear System Solver is a simple extension into an application of the random number guesser; it takes as input an n by n matrix and n length vector solution and then generates n length x-vector solutions. Solving $Ax = b$ via the simple classic LU-decomposition is $\mathrm{O}(n^3)$ operations. LU-decomposition, however, is very open to round-off error, and we prefer to use either the Method of Partial Pivoting or PLU-solve which is also $\mathrm{O}(n^3)$. Using PLU-solve will prevent accumulation of error in all but the most sensitive linear systems; in such cases, an algorithm called Complete or Maximal Pivoting is used. Complete Pivoting also completes in $\mathrm{O}(n^3)$ operations, however, its other terms are significant and this methodology takes significantly longer for smaller n values. Since there are minimized divisions in Complete Pivoting, it works exceptionally well in terms of handling round-off error. Additionally, there are iterative methods for determining x that do so in $\mathrm{O}(n^2)$, but which are based on removing error each iteration rather than strictly solving for x.

The members of the population in this application of the GGA are possible x solutions; to evaluate the fitness of a member, it is multiplied with A and then normed with b. So evaluating the fitness of each member is $\mathrm{O}(n^2)$ operations. However, the algorithm does this for each member of the population and for each generation, so if the algorithm reaches an acceptable solution in g generations, the total operations are $\mathrm{O}(p * g * n^2)$. Obviously, for small n, the GA does not compare favorably except against Complete Pivoting with its additional powers of n. For the GGA to outperform, strictly in terms of operations, the other methods at any point, the following must be true:

p, g functions of n such that $p(n) * g(n) < n, k < n$ for some k, where the population p has its best member within ϵ in g generations. This implies that p & g are powers of n less than 1 or converge to some constant values.

From statistics of natural systems, we are inclined to believe that this is the case for p; namely, a very small portion of the total population can represent the entire population via combination and crossover. Likewise, given n dimensions each with k possible values, we require only k members minimally to represent n^k possible members. Though the number of individuals required to represent via random generation all the traits with a strong certainty will no doubt be larger than k, it will also be less than n^k as the natural systems indicate.

3 CONCLUSIONS



3.1 Engine Performance on Problem Suite

On the two problems addressed, we witness vastly different performance in the engines. The random number guesser rapidly obtains a solution near the correct one and then spends a relatively long time ironing out the exact value. Comparing this toy case application to other search methods is not particularly relevant, since solid algorithms to solve this sort of problem exist, but we can see the power of such a GGA versus other random search algorithms and that simple extensions of the random number guess are not solvable by a general algorithm.

The performance of the Linear System Solver version of the GGA was rather abysmal; the population very rapidly converges to solutions that have good partial performance (about half of the b values are close), but these values are typically wildly different than those needed to obtain a solution where all b values are very close. The population can then not move towards these correct solutions because the "genes" that make up those solutions have been eliminated.