
On Genetic Algorithms*

(Extended Abstract)

Eric B. Baum Dan Boneh Charles Garrett
eric@research.nj.nec.com dabo@cs.princeton.edu garrett@research.nj.nec.com

NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

We analyze the performance of a Genetic Type Algorithm we call Culling and a variety of other algorithms on a problem we refer to as ASP. Culling is near optimal for this problem, highly noise tolerant, and the best known approach in some regimes. We show that the problem of learning the Ising perceptron is reducible to noisy ASP. These results provide an example of a rigorous analysis of GA's and give insight into when and how GA's can beat competing methods.

To analyze the genetic algorithm, we view it as a special type of submartingale. We prove some new large deviation bounds on this submartingale which enable us to determine the running time of the algorithm.

1 Introduction

Genetic Algorithms (GA's) have been widely studied ([7, 12]) and are frequently applied by engineers in the field to a great variety of problems. This interest is due to several intriguing, intuitive motivations. Nonetheless there is little understanding in the literature of when, or whether, GA's are better than competing methods. This paper will review the motivations for GA's and then study their performance relative to competing algorithms on some optimization and search problems. We believe these problems are of some intrinsic interest, having for example direct applications in learning theory, and that our analysis sheds light on when and why Genetic approaches are desirable.

*The full version of this paper is available via WWW at <http://www.neci.nj.nec.com/homepages/eric.html>

Permission to make digital/hard copies of all or part of this material without fee is granted provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.
COLT'95 Santa Cruz, CA USA© 1995 ACM 0-89723-5/95/0007..\$3.50

Consider the following problem, which we call ASP¹ for 'Additive Search Problem'. Let $X \equiv \{1, 2, \dots, L\}^N$. We have an oracle which returns the number of components common to a query $x \in X$ and a target vector $t \in X$. The objective is to find t with as few queries as possible. For simplicity, we always assume $N > L$.

Our genetic approach is to initiate a population P of strings drawn randomly from X . At each generation we "breed" random strings in P to produce a set of new strings. The top scoring ones are then selected to form a new population which, on average, is "fitter" than the last generation's. Iterating this appropriately can be shown to result in a perfect string after about $O(N \log L \log^2 N)$ queries. On ASP, the information theoretic lower bound is $\Omega(N \log L / \log N)$ queries, and a local search algorithm requires $\Omega(NL)$ queries. This thus represents the first problem we are aware of on which a Genetic Algorithm can be analytically shown more effective than known competitors such as hill climbing². In addition, a genetic approach is well suited for a parallel machine. Our method solves ASP in time $O(\sqrt{N})$ on a parallel machine.

Standard genetic algorithms were first rigorously analyzed in [15, 19]. They view GA's as quadratic dynamical systems. The GA's we analyze use multiway breeding and hence can't be modeled as a QDS.

Our genetic algorithm is very robust to noise. For example, say the oracle returns a value which is corrupted by Gaussian noise with $\sigma = \sqrt{N/L}$. The performance of our GA is not affected. The performance of several alternative algorithms degrades radically.

The problem of learning a half space with integer weights from examples drawn from a uniform distribution has been studied in [22, 23, 8, 4]. We show that this problem can be directly mapped into a noisy ASP problem, yielding some solution techniques.

¹ASP is related to the "Royal Road Function" of [14]. ASP is identical to the game of "Mastermind" except that in Mastermind the oracle also returns a second number, the maximum number of components common to t and any permutation of the components of x .

²C.f. [14] for a survey of the literature.

Historically the main motivation for interest in GA's was Holland's "Schema Theorem", which we review in section 2. The point is that GA's may benefit by use of memory. Instead of storing only one candidate like local search, GA's store a population of candidates and hope to improve rapidly by combining 'building blocks' from different candidates. The Schema Theorem indicates that GA's can search for useful building blocks, called 'Schema' (plural schemata), with huge 'implicit parallelism'. Unfortunately no explicit connection between the Schema Theorem and optimization effectiveness has ever been made. Our results clarify this issue.

Standard genetic algorithms differ from our proposal in that they select 'parents' with probability proportional to their fitness rather than culling offspring less fit than a threshold. They perform badly on ASP, requiring $\Omega(N^2 \log N)$ queries. As we shall see, when applying the standard genetic algorithm to solve ASP, the expected gain per generation is of order 1. This means that it takes $\Omega(N)$ generations to solve the problem. By contrast our proposal to cull the offspring gives an increase of a standard deviation, of order $\sqrt{N}$, per generation, and thus solves the problem in about $\sqrt{N}$ generations. In addition, the slower progress of standard GA's means there is more time for random deviations to act. This necessitates a larger population size.

'Intrinsic parallelism' might be thought more important in case the useful schemata consist of many components, say k , rather than 1 as in ASP, since there will then be a vast number, $\binom{N}{k} L^k$, of potential schemata to search through. We address this question in section 6 with regard to a generalization of ASP called k -ASP. Our conclusion is that the GA's do not succeed in realizing gains due to 'intrinsic parallelism', but we give a binary search based algorithm which does.

We also study averaging algorithms we call Mean Field Approximations (since they are related to a calculational technique of this name in the physics literature). These perform similarly to our GA's on ASP and k -ASP, but worse in some less symmetric situations.

Algorithms similar to Culling³ called "Evolution Strategies" have been proposed and studied by I. Rechenberg [16],[17] and H. P. Schwefel[21]. For a recent review of this work see [2]. Rechenberg [17] calculated convergence rates on some simple problems for a Culling-like algorithm with a population size of one, and proposed heuristics based on these results. Schwefel[21] gave an expression for expected progress rate of related algorithms, but without recombination (crossover), and studied as well proposals for dynamically changing mutation rates.

One practical reason why GA's are used widely is that they are easy to apply to any novel area. Thus it is important to compare GA's to other algorithmic tech-

niques which share this property, such as local search, simulated annealing, and tabu search.

Practical problems are often 'blind'. In a blind problem, one only has access to the fitness function by calling an oracle. One must learn much of the structure of the problem. In many blind applications it is well motivated to think of the number of queries as the primary cost of an algorithm (as opposed to, say, its computing time). So for example consider the problem of designing an intelligent system. After you set up the system, you may have to allow it to go through an extensive and costly learning procedure before you are able to evaluate its fitness. Holland has proposed[11] using GA's to train "Classifier Systems", which interact in complex environments and learn.

In complex environments one may have little understanding of the fitness function. For instance, consider a model where each component of a string contributes an additive amount to the fitness, and there is an additional, unknown contribution due to interaction between the components. If the interaction terms can be reasonably treated as random and smaller than the additive components (though by no means negligible) then our analysis of the Noisy ASP problem generalizes to show that our GA will be an effective approach.

In summary we have studied the behavior of a variety of algorithms on some natural search problems- ASP and its generalizations- and given an application to a learning theory problem. We have proposed a new Genetic Algorithm and analyzed its performance on these problems. Our analysis uses submartingale techniques which may have some intrinsic, mathematical interest. We have analyzed also the performance of the standard Genetic algorithm. These analyses cast considerable light on the motivations for GA's. GA's seem strong on certain problems, especially in the presence of noise or unknown interactions which can be modeled as noise. Finally we have proposed a Parallel Search Algorithm which can achieve the intrinsic parallelism motivating GA's in some cases (e.g. k -ASP) where GA's can not.

§2 reviews Holland's Schema Theorem. §3 proposes a genetic type approach we call 'Culling'. §4 discusses the ASP, and analyzes the performance of several algorithms on it. §5 discusses the noisy ASP problem. §6 discusses two generalizations of ASP. §7 discusses an alternative method for solving ASP which we refer to as the Mean Field Approximation (MFA) Algorithm. §8 reduces the problem of learning the Ising Perceptron problem to an ASP problem.

2 Holland's Schema Theorem

In this section we present the standard genetic algorithm approach and recall the original motivation for studying it⁴. Let $A = \{a_1, a_2, \dots, a_L\}$ be an alphabet of

³Culling was rediscovered by one of us[3] in ignorance of this literature. We thank an anonymous COLT referee for calling our attention to [2] and references therein.

⁴The results of this section are due to [12]. A simple, clear presentation may be found in [7], chapter 2.

size L . The characters in A are referred to as “alleles”. Let $P \subseteq A^N$ be a set of M strings of length N over the alphabet A . The set P will be referred to as the population. A fitness function is a function $F : A^N \rightarrow \mathbb{R}$. Suppose we are given an oracle for evaluating the fitness of any string $s \in A^N$. We wish to find a string of high fitness. The following method is the standard genetic algorithm approach for solving such problems:

1. Initialize P as M random strings of length N .
2. Evaluate the fitness of the strings in P . If one is optimal, quit.
3. Draw from P , with replacement, with probability proportional to fitness, a new set Q of $2M$ strings
4. Breed these strings to form a set P' of M strings. Let $P = P'$ and go to 2.

There are many methods for breeding two strings. One popular method (one point crossover) is to choose j uniformly between 1 and N . Then the child of strings s_1^1 and s_2^1 has components s_i^1 for $i \leq j$ and components s_i^2 for $i > j$.

We now discuss the original motivation for this approach. A ‘schema’ (plural ‘schemata’) is a geometric axis-parallel hyperplane, that is an assignment of some letters to some sites. For example, $c_2 = a_1, c_4 = a_3, c_7 = a_1$ is a schema. Any string satisfying these component constraints is said to be a member of this schema. In a pattern recognition context, schemata might be called ‘features’. The notion is that certain schemata are building blocks which combine to form good strings. Holland defines the “fitness” of a schema to be the average fitness of all its members in P .

Now draw a set Q of strings from our population as in step 3 above. Then the expected frequency of every schema in Q is proportional to its ‘fitness’. So we select for ‘strong’ schemata.

Take the strings in Q and recombine them in order to make new strings as in step 4. One might also ‘mutate’ by randomly changing the allele at each site with some low probability p_m . How many schemata are left undisturbed by these operations? Let H denote a particular schema, $f(H)$ denote its ‘fitness’, μ denote the average fitness of strings in the population, and let $\delta(H)$ denote the length of H , i.e. the distance between its first and last defining site, so that $\delta(H)/(N-1)$ is the probability the crossover point lies within the schema H . Let o_H denote the order of the schema H , i.e. the number of sites on which it is defined, so that the probability it will not be corrupted by mutation is $(1-p_m)^{o_H} \sim 1 - o_H p_m$. Let $m(H)$ denote the number of examples of H in P and $m'(H)$ denote the expected number of examples of H in P' . Then one can readily see:

$$m'(H) \geq m(H) \frac{f(H)}{\mu} \left[1 - \frac{\delta(H)}{l-1} - o_H p_m \right]$$

This is Holland’s schema theorem. Note the RHS is an underestimate of m' as we have neglected the possibility

of crossovers and mutations producing *new* examples of H in P' . This equation says that any schema whose ‘fitness’ is greater than average by an amount greater than their breakup probability (bounded by the quantity in square brackets) exponentially increases its prevalence in the population.

One may attempt to estimate the number N_S of schemata so processed. Neglect mutation and consider (as a lower bound) only schemata with breakup probability less than ϵ , a constant, i.e. schemata with length $\delta(H) < l_0 \equiv \epsilon(N-1) + 1$. Now we may readily count

$$N_S \geq M(N - l_0 + 1)2^{l_0-2}$$

Holland’s oft repeated estimate that when we reproduce a population of M strings we implicitly consider $O(M^3)$ schemata comes from plugging in $M = 2^{l_0/2}$ (chosen so that we expect in P to find one or fewer representatives of each schema of order $> l_0/2$) ([7, p41]). This specific $O(M^3)$ scaling relationship does not seem well motivated to us. For any fixed ϵ , we achieve an *exponential* relationship for any population size M polynomial in N .

The upshot is that this procedure increments or decrements the frequency of a vast number of schemata according to their relative fitness, while using relatively little computation. Holland describes this as a search with “intrinsic parallelism”. The lure of utilizing this efficiency has been the main intuition driving the field of Genetic Algorithms.

3 Is breeding proportional to fitness optimal?

Let $f(i)$ denote the fitness of the i th string in the population, and $\mu = \frac{1}{M} \sum_{i \in P} f(i)$ be the average fitness. Let μ'_H be the average fitness after sampling proportional to $f(i)$ as suggested by Holland: $\mu'_H = \frac{1}{M\mu} \sum_{i \in P} f^2(i)$. Now assume the distribution of fitness in the population follows a Gaussian with deviation σ . This might reasonably be expected by the Law of Large Numbers for $M, N \gg 1$. Then we find

$$\mu'_H \sim \frac{1}{\mu\sqrt{2\pi}\sigma} \int_0^\infty x^2 e^{-(x-\mu)^2/2\sigma^2} dx = \mu + \sigma^2/2\mu$$

If the fitness is largely additive (linear), we will expect $\mu \sim O(N)$, and $\sigma \sim O(\sqrt{N})$. Then the gain in fitness per generation, $\mu'_H - \mu$, is of order 1.

Now consider an alternate sampling method which we refer to as culling. Let P be the current generation of strings. Let μ, σ be the expectation and deviation in the fitness of a string bred from P . The culling breeding method is as follows:

1. Choose parents randomly and uniformly from P . Breed.
2. If child’s fitness $> \mu + \sigma$, accept. Otherwise, reject the child.

Since we accept any child better by σ , we reject only a constant fraction of the children. But we achieve a gain per generation of $\Omega(\sigma) = O(\sqrt{N})$. More generally, we could reject children of fitness less than $\mu + \lambda\sigma$ and optimize the parameter λ .

An alternative implementation of the culling strategy is: Let M be the population size and let $\alpha > 1$ be some fixed constant. Generate αM strings by randomly breeding members of the current population. Set the new population to be the top M of these αM strings.

4 Additive Search Problem

We now turn to the ASP, defined in the introduction. Note that a solution to ASP is an N bit string over L alleles. It evidently requires $N \log L$ bits of information to specify the answer. Each time it is queried, the Oracle returns a number from 0 to N , the fitness. Thus the oracle returns at most $\log(N)$ bits of information. Hence, any algorithm solving this problem must make at least $N \log L / \log N$ queries. Chvatal [5] shows that with high probability a set of $O(N \log L / \log N)$ random queries will completely constrain the solution. This shows that the lower bound is tight, however it does not provide a polynomial time algorithm to find the target.

The following is the most efficient algorithm we know of for solving ASP. We have a current string, incorporating any known distinguished alleles. Randomly change L sites for which the distinguished allele is not yet known and query. A finite fraction of the time the fitness will change. If so, one or more changed sites either were the distinguished allele, or became it (depending on the sign of the fitness change). Find these sites by a binary search, which will involve at most $\log L$ queries. Since on average we take $O(\log L)$ queries to find each distinguished allele, this algorithm takes $O(N \log L)$ queries to find the optimum string. We have shown:

Theorem 1 *ASP can be solved in polynomial time using $O(N \log L)$ queries. Furthermore, any algorithm which solves it must use at least $\Omega(N \log L / \log N)$ queries in the worst case.*

4.1 Hill Climbing and Genetic Approaches

We now discuss the performance of a simple hill climbing algorithm and of a traditional genetic algorithm on this problem. Consider the Random Mutation Hill Climbing (RMHC) procedure [14]:

1. Query on a random string, called "CFS" (Current Favorite String).
2. Change a random site in CFS and query.
3. If mutated string is as good or better than CFS, then set CFS to this string.
4. If not yet optimal string, go to (2).

This can easily be seen ([14]) to solve the problem in time $O(LN \log(N))$.

Next, we briefly analyze the performance of the standard GA described in section 2 when applied to ASP. Since (by the argument sketched in §3) the expected gain in average fitness per generation is of order 1, the algorithm requires $T = \Omega(N)$ generations to solve ASP. Let M be the size of the population used by the algorithm and let Np_t be the mean fitness in the population at time t . In t steps, there will be a random drift of about $\sqrt{tMp_t}$ in the number of correct alleles per component. To make sure that none of the correct alleles disappear from the population in any of the components we must require that $Mp_t > \log(N)\sqrt{tMp_t}$. Thus the size of the population must be at least $M > \log^2(N)t/p_t$. Since the mean fitness grows by 1 per generation, p_t increases by $1/N$ per generation. Hence, for large t we have that $t/p_t = \Omega(N)$. This means that the size of the population must satisfy $M > \Omega(N \log^2 N)$. In N generations, the number of oracle calls made by the algorithm is $\Omega(N^2 \log^2 N)$. A full analysis of the performance of this algorithm on a problem similar to ASP can be found in [15].

We have been discussing a GA with mutation rate set equal 0. finite mutation will not only impose a drift towards weaker strings, it will also increase the variance. It will remain true that we must choose the population size to be $\Omega(N \log N)$. Hence, mutation will not improve the scaling of the GA.

4.2 GA Approach Using Culling

Let $f(x)$ denote the ASP fitness value of a string x . The culling approach of §3 leads to the following algorithm for ASP which we designate as GA2:

1. Set $t = 0$, $p = 1/L$, $M = L \log N$.
2. Choose M random strings with fitness at least pN . Call this set P .
3. Let $\sigma = \sqrt{p(1-p)/N}$.
4. Use M -way breeding to breed a new string x . If $f(x) < (p + \sigma)N$ discard x . Repeat this until M strings have been generated. Set P to be this new set of strings. If a string of fitness N has been generated, stop.
5. Set $t := t + 1$; $p := p + \sigma$; $M = \left(\frac{t}{p}\right) \log^2 N$ and go to step 3.

The breeding strategy⁵ we refer to as M -way breeding is the following: For all k , the k 'th component of the child string is chosen by picking a string uniformly at random from P and using it's k 'th component. Thus, all strings in the population have a chance of contributing to the child. The rest of this section is devoted to sketching a proof of the following theorem.

⁵ M -way breeding avoids correlations which would otherwise obstruct our proof. Simulations show that culling also works well with standard pairwise uniform crossover, but we are unable to analyze it.

Theorem 2 *Algorithm GA2 will solve the ASP problem with constant probability in $O(\sqrt{N})$ generations and $O(N \log^2 N \log L)$ queries.*

Notice that the size of the population M is changing as the algorithm progresses. In each generation the average fitness of the population increases by at least $\sigma = \sqrt{Np(1-p)}$, which is known in advance. In addition there is a random drift. The size of the random drift, however, is determined by the population size M . What we must do now is to (a) estimate how long it will take till $p = 1$, and (b) Show that M is large enough so that the random drift doesn't kill us, e.g. because fluctuations remove all copies of a correct allele from the population at some location.

The value of pN always serves as a lower bound on the fitness of strings in the population. The number of generations till $p = 1$ upper bounds the number of generations the algorithm requires, assuming that in every generation it is possible to generate strings of fitness $(p + \sigma)N$.

Lemma 3 *The number of generations till $p = 1$ is no more than $O(\sqrt{N})$.*

Proof We ask first how many generations until $p = 1/2$. Disregard the $(1-p)$ term for simplicity as it will change the result by at most a factor of 2. Start with $p = 1/L$. At each generation, to begin with, p increases by more than $\sqrt{N/L}$. So after $\sqrt{N/L}$ generations, p has more than doubled, i.e. $p > 2/L$. Now at each generation p increases by more than $\sqrt{2N/L}$. After $\sqrt{2N/L}$ generations, p has at least doubled again, to $4/L$. Iterating this procedure $\log L$ times we find it takes a total of less than

$$\sqrt{N/L}(1 + \sqrt{2} + \sqrt{4} + \sqrt{8} + \dots + \sqrt{L/2}) = O(\sqrt{N})$$

generations until $p = 1/2$ (as the sum in the parenthesis is bounded by $O(\sqrt{L})$ since it is a geometric series). Now we must continue iterating from $p = 1/2$ to $p \sim (N-1)/N$. (The next generation at this point will result in a perfect string, since $q = N$.) In this second half of the iteration we may disregard the p term in the square root, and retain only the $1-p$ term. The convergence is now symmetrical, and takes fewer than $O(\sqrt{N})$ generations. $\square$

Naively, since at each generation we sample 'random strings' and ask to gain only one sigma in fitness, we should have to examine only a finite number of children per each one we accept. We analyze the number of queries the algorithm makes under this assumption.

Lemma 4 *Assuming that only a constant fraction of children are rejected in step 4 of the algorithm, the total number of queries performed by the algorithm is*

$$O(N \log^2 N \log L)$$

Proof Let p_t be the value of p at time t . First observe that once $p_t > \frac{1}{2}$ we have that $M < O(\sqrt{N} \log^2 N)$ implying that from that point on the algorithm will make at most $O(N \log^2 N)$ queries. We show that this is also the number of queries in the duration where $p_t < \frac{1}{2}$. The proof of the previous lemma shows that p_t goes through $\log L$ doubling periods until it reaches $p_t = \frac{1}{2}$. The number of queries within each doubling period is upper bounded by $tM = \frac{t^2}{p_t} \log^2 N$. It is not difficult to see that throughout the algorithm, $\frac{t^2}{p_t} < O(N)$. Hence, the number of queries in each doubling period is at most $O(N \log^2 N)$. Since there are $\log L$ doubling periods, the lemma follows. $\square$

The difficult part of the proof of correctness is showing that it is possible to gain one σ in fitness per generation. To prove this point we must analyze the behavior of the number of correct alleles per component in every generation. We first give some intuition and then state the formal theorem. Let $X(t, i)$ be a random variable counting the number of strings in the population containing the correct allele in component i at generation t . Let p_t be the value of p at time t and let M_t be the size of the population at time t . Naively, after t steps, the expectation of $X(t, i)$ is bigger than $M_t p_t$ and the random drift is bounded by $\sqrt{t M_t p_t}$. Therefore, to make sure that a correct allele doesn't vanish from the population in any of the N components we must require that $M_t p_t > \log(N) \sqrt{t M_t p_t}$. This implies that the population size must satisfy $M_t > \log^2(N) t / p_t$, which is precisely the value used by the algorithm.

We now briefly sketch the formal proof for the correctness of the algorithm. It is not difficult to see that the variables $X(t, i)$ are distributed as follows:

Lemma 5 *For any $t \geq 0$, given $X(t, i)$ the variable $X(t+1, i)$ is binomially distributed over M_{t+1} experiments with success probability*

$$\frac{X(t, i)}{M_t} + g(X(t, 1), \dots, X(t, N))$$

for some positive function $g : \mathbb{R}^N \rightarrow [0, 1]$. The function g will be referred to as the gain.

Since the gain is never negative,

$$X(0, i), X(1, i), \dots, X(t, i)$$

forms a submartingale. The main lemma we prove is the following:

Lemma 6 *Suppose at time t the population size was defined to be $M_t = kt/p_t$ for some fixed k . Then for all $i, t, \alpha > 0$ we have*

$$\Pr \left[\left(1 - \frac{\alpha}{\sqrt{k}}\right) p_t M_t \leq X(t, i) \leq \left(1 + \frac{\alpha}{\sqrt{k}}\right) p_t M_t \right] \geq 1 - e^{-\alpha}$$

It should be clear that using Lemma 6 the heuristic argument given above can now be made rigorous. To see this take $k = O(\log^2 N)$ and $\alpha = O(\log N)$.

To prove Lemma 6 we must first analyze the behavior of the gain function, g . The problem can be stated as follows: we are given N independent Bernoulli random variables, $Z_1, \dots, Z_N$ corresponding to the N components of the strings. Let p_i be the success probability of the i 'th Bernoulli random variable, $i = 1, \dots, N$. Each Z_i corresponds to the event that a randomly bred string from generation t will contain the correct allele at component i . Thus p_i is simply $X(t, i)/M_t$. The gain in component i is

$$g(p_1, \dots, p_N) = \Pr[Z_i = 1 \mid \sum_{j=1}^N Z_j > T] - p_i$$

where T is the threshold used in step 4 of the algorithm. The following lemma, which we state without proof, gives a lower bound on the gain.

Lemma 7 Let $P = \sum_{i=1}^N p_i = E[\sum Z_i]$ and $p = \frac{P-p_1}{N-1}$. For $T = P + \sqrt{P}$ we have

$$\Pr \left[Z_1 = 1 \mid \sum_{j=1}^N Z_j \geq T \right] \geq p_1 + \frac{p_1(1-p_1)}{\sqrt{Np(1-p)}}$$

Notice that when all the p_i 's are equal, we have that $p = p_1$ in which case the gain is exactly $\sqrt{p_1(1-p_1)/N}$, i.e. the gain is equal to one sigma.

Recall that our objective is to bound the large deviations of the submartingale

$$X(0, i), X(1, i), \dots, X(t, i).$$

This can be achieved using Azuma's inequality [1, p. 85]. However, since the inequality holds for any martingale, the resulting bound is too weak. Thus, we need to prove an Azuma type inequality for our special submartingale. We first consider a simpler problem in which the gain at step t is fixed, i.e. it does not depend on $X(t, i)$.

Lemma 8 Let $g : N \rightarrow [0, 1]$ an arbitrary function. Consider the following submartingale:
 $X_0 = g(0)$ and

$$X_t \mid X_{t-1} \approx \text{Binomial} \left[M, \min \left(\frac{X_{t-1}}{M} + g(t), 1 \right) \right]$$

Then, for any t and $\lambda > 0$:

$$\Pr[X_t - G(t) < -\lambda] < e^{-\frac{1}{2}\lambda^2/V(t)}$$

where $G(t) = M \sum_{j=0}^t g(j)$ and $V(t) = \sum_{j=0}^t G(j)$.

The function $G(t)$ is the expectation of X_t since by induction one can show that

$$\begin{aligned} E[X_t] &= E[E[X_t \mid X_{t-1}]] = E[X_{t-1} + Mg(t)] \\ &= G(t-1) + Mg(t) = G(t) \end{aligned}$$

The function $V(t)$ serves as an upper bound on the variance of X_t .

To see that the bound provided by Lemma 8 is better than the Azuma bound consider the case where $g(0) = p$ and $g(t) = 0$ for $t > 0$. In this case $G(t) = Mp$ and $V(t) = tMp$. Since the gain is 0, the variables $X_0, \dots, X_t$ form a martingale. Azuma's inequality shows that $\Pr[X_t - Mp < -\lambda] < e^{-\lambda^2/2tMp^2}$, while Lemma 8 shows that $\Pr[X_t - Mp < -\lambda] < e^{-\lambda^2/2tMp}$ which is better since the random drift can now be bounded by $\lambda = \sqrt{tMp}$.

Lemma 8 gives a good large deviation bound on a submartingale in which the gain is constant at every generation. However, Lemma 7 shows that the gain is not constant and can in fact be arbitrarily close to 0. To prove Lemma 6 we use Lemma 8 and argue that even though the gain is not constant, it can be lower bounded by some constant with high probability. We leave the details for the full version of the paper.

5 Learning in the Presence of Noise

In this section we compare the performance of several algorithms on noisy ASP, i.e. with a noisy fitness oracle⁶. The noise model we use will be of the form $\tilde{f}(x) = f(x) + e(x)$ where $f(x)$ is the true fitness of the string and $e(x)$ is a fixed noise function. For every x , the value of $e(x)$ is sampled independently from a given distribution. Thus, $e(x), e(y)$ are independent when $x \neq y$. However, querying the fitness of a string x twice will produce the same result both times. For convenience we assume that $E[e] = 0$. We denote by σ_e the standard deviation of the noise.

The binary search algorithm described in Section 4 can be made to work in the presence of noise using a result of Renyi [18]. Renyi shows how to perform binary search using an unreliable oracle. However, the performance of the algorithm degrades proportionally to the noise deviation.

The hill climbing algorithm considered in Section 4.1 performs poorly in the presence of noise. In fact, for some noise distributions such as the exponential distribution, it does not converge at all.

We now turn our attention to analyzing how GA2 performs in the presence of noise. We show that for 'small' noise the algorithm is unaffected. We first calculate the expected gain in true fitness from one generation to the next. Let x be a string generated by breeding random strings from the current generation. Let A be the event

⁶There are two roughly equivalent versions of this problem. One is where the oracle returns a noisy version of the true fitness, and the goal is to find the string of highest true fitness. Alternatively, one might discuss the situation where one simply desires to find the string x maximizing $\tilde{f}(x)$. In the first case, no procedure can guarantee with arbitrary confidence to return the optimal string, and in the second case this can only be done by complete exhaustive search. The methods we discuss solve either problem efficiently with very high (but not arbitrary) confidence.

that x is admitted into the next generation. Then

$$\Pr[A] = \sum_{k=0}^N \Pr[f(x) = k] \Pr[e > T - k]$$

where T is the threshold on the noisy fitness which x has to achieve. The expected true fitness in the next generation will be

$$E[f(x) | A] = \frac{1}{\Pr[A]} \sum_{k=0}^N k \Pr[f(x) = k] \Pr[e > T - k]$$

Let p_t be the value of p at time t . Let $\sigma = \sqrt{Np_t(1-p_t)}$. Then $T = Np_t + \sigma$. When the standard deviation of the noise is much bigger than σ the probability $\Pr[e > T - k]$ will hardly change as k ranges from $p_t - \sigma$ to $p_t + \sigma$. Hence, $E[f(x) | A]$ will change by very little from generation to generation. This means that there will be very little gain from one generation to the next.

An example where the standard deviation of the noise is larger than σ is the uniform noise model with e uniformly chosen from $1, \dots, N$. Here the standard deviation is $O(N)$ which is much larger than $\sqrt{N/L}$ which is the value of σ at the first few generations of the algorithm. For this noise model we can show that

$$E[f(x) | A] = Np_t + O(p_t).$$

Hence, the gain in every generation is $O(p_t)$. Initially $p_t = 1/L$ which is much smaller than $\sqrt{N/L}$ which is the initial gain with no noise. We see that with such large noise values, the expected gain per generation is small implying a considerable slowdown.

We now consider models where the standard deviation of the noise is much smaller than σ . Say $\sigma_e < \sigma/10$. We rewrite the expected fitness of the next generation as:

$$\begin{aligned} E[f(x) | A] &= \frac{1}{\Pr[A]} \sum_{k=-(N-T)}^T (T - k) \Pr[f(x) = T - k] \Pr[e > k] = \\ T - \frac{1}{\Pr[A]} \sum_{k=-(N-T)}^T k \Pr[f(x) = T - k] \Pr[e > k] &\geq \\ T - \frac{1}{\sigma \Pr[A]} \sum_{k=0}^T k \Pr[e > k] \end{aligned}$$

Using the fact the $E[e] = 0$ it is not difficult to see that $\sum_{k=0}^T k \Pr[e > k] \leq \frac{1}{2} \sigma_e^2$. Furthermore,

$$\Pr[A] \geq \Pr[f(x) > T] \Pr[e > 0] \geq \frac{1}{4} \times \frac{1}{2} = \frac{1}{8}$$

Hence,

$$E[f(x) | A] \geq T - 4\sigma_e^2/\sigma = Mp_t + \sigma - \frac{4\sigma_e^2}{\sigma}$$

Thus, when $\sigma_e < \sigma/10$ we see that the gain per generation is hardly effected by the noise. Hence, for such noise

models the GA will suffer a minor slowdown. This result should be contrasted with other search algorithms whose performance degrades considerably under these noise models. In this noise regime, the robust binary search algorithm is slower than GA2 by a factor of $\tilde{\Omega}(\sqrt{N})$, and the RMHC is slower still or doesn't converge at all, depending on the noise distribution.

6 Generalizations of ASP

Holland's 'intrinsic parallelism' might be thought more important in case the useful schemata consist of many components, say k , rather than 1 as in ASP, since there will then be a vast number of potential schemata to search through. Unfortunately, crossover breeding, used in genetic algorithms, tends to break up such schemata. Holland's Theorem attempts to estimate gains from 'intrinsic parallelism' when this breakup is taken into account, but there is a big gap between the conclusion of Holland's Theorem and the goal of showing the GA effective, especially compared to competing approaches.

To study this gap we propose a more complex problem, k -ASP. Here there is a target division (unknown to us) of the set $\{1, 2, \dots, N\}$ of components into N/k groups each of size k . A group of components is 'correct' in a query vector $x \in X$ just in case x agrees with the target vector t on all of the components in the group. The oracle returns the number of correct groups times k , and again we ask to find t . For example, take $k = 2$ and $t = 0110$ where the first and second component form one group and the third and fourth component form another. Then the string 0100 would receive a score of 2 even though there are 3 correct alleles. Note that when $k = 1$ this problem is identical to ASP. As before, we will assume that $N \gg L^k$.

The following algorithm is an adaptation of the one presented in Section 4.2 to the k -ASP problem.

1. Set $t = 0$, $p = 1/L$, $q = k/L^k$, $M = \sqrt{NL^k/k}$.
2. Choose M random strings with fitness at least qN . Call this set P .
3. Let $q = kp^k$ and $\sigma = k\sqrt{p^k(1-p^k)/N}$.
4. Use M -way breeding to breed a new string x . If $f(x) < (q + \sigma)N$ discard x . Repeat this until M strings have been generated. Set P to be this new set of strings. If a string of fitness N has been generated, stop.
5. Set $q := q + \sigma$; $p := q + (1 - q)p$ and go to step 3.

Here q is the fraction of correct groups in the population and p is the fraction of correct alleles in the population. The deviation in q is given by the expression for σ computed in step 3, since the probability of generating a correct group is p^k .

The reason we set $p := q + (1 - q)p$ in step 5 is the following: After step 4, the fraction of correct groups in

the population is at least $q + \sigma$, which is why we increment q by σ . Now, in the Nq sites which correspond to correct groups all alleles are correct. In the remaining $N(1 - q)$ sites the expected fraction of correct alleles hasn't changed⁷. Therefore there are $N(1 - q)p$ correct alleles in those sites. Hence, the total number of correct alleles in the population is now $Nq + N(1 - q)p$. Consequently, we obtain that the expected fraction of correct alleles in the population is given by $q + (1 - q)p$.

Note that unlike the algorithm in Section 4.2, the size of the population M is fixed throughout the algorithm. This is done for the sake of simplicity.

Theorem 9 *The above algorithm will solve the k -ASP problem using $\tilde{O}(NL^k/\sqrt{k})$ queries with high probability.*

Proof Sketch: Following a similar argument to the one given in Lemma 3 shows that assuming the algorithm doesn't get 'stuck' it will find the target string within $O(\sqrt{NL^k})$ generations. The argument used in Section 4.2 to bound the random drift shows that taking $M = O(\sqrt{NL^k/k})$ ensures that the algorithm doesn't get stuck with high probability. Hence, the GA makes a total of $O(NL^k/\sqrt{k})$ queries proving the theorem. $\square$

We note that the the Binary Search Algorithm from §4 can be adapted to solve k -ASP. At the random query step, instead of randomly changing L sites as for ASP, randomly change L^k sites, and then perform the binary search. This can be seen to allow efficient solution of k -ASP with only $O(Nk \log L)$ queries.

7 Mean Field Approximation

This section presents an alternative algorithm for noisy ASP. The algorithm can easily be adapted to solve the k -ASP problem as well (with L^k slowdown). We refer to this algorithm as the Mean Field Approximation or MFA since it is reminiscent of this approximation in physics[13, Ch. 7]. As before, the noisy fitness value will be denoted by $\tilde{f}(x) = f(x) + e(x)$, where $f(x)$ is the true fitness, and $e(x)$ the noise. For simplicity we assume the noise has expectation 0.

Algorithm(MFA):

1. Pick M random words in $\{0, \dots, L\}^N$ and evaluate their fitness value. The value of M will be determined later.

⁷To be precise, the expected fraction of correct alleles in components which are not part of a correct group has gone down by a negligible amount. We need to subtract the probability of a correct group forming and renormalize. The expected fraction of correct alleles in those sites is now $\frac{p-p^k}{1-p^k}$ which for large k is very close to p . Using this refinement p should be updated by $p := q + (1 - q)\frac{p-p^k}{1-p^k}$

2. Iterate for $j = 1, \dots, N$ and perform the following operations:
 - (a) Partition the set of words into L sets $A_1, \dots, A_L$ according to the value of the j 'th component. The set A_k will contain the words containing k in their j 'th component. The size of each A_k is roughly M/L .
 - (b) Define $S(A_k) = \sum_{x \in A_k} \tilde{f}(x)$.
 - (c) Set x_j to k where $S(A_k)$ is the maximum of $S(A_1), \dots, S(A_L)$.
3. Define the word $x = x_1 x_2 \dots x_N$ and output x as the target string.

The algorithm makes M oracle calls in total. Intuitively the partition set implied by the correct allele has the highest mean fitness value in every component. For large enough M fluctuations are overwhelmed and the algorithm is likely to find the distinguished allele in every component.

Consider the first component. Let $A_1, \dots, A_L$ be the partition of the random words according to the value of the first component. Let t_1 be the distinguished allele in the first component. It is not difficult to show that the expectation of $S(A_r)$ is

$$E[S(A_r)] = \begin{cases} \left\lfloor \frac{(N-1)}{L} \right\rfloor |A_r| & \text{when } t_1 \neq r \\ \left[1 + \left(\frac{(N-1)}{L} \right) \right] |A_r| & \text{when } t_1 = r \end{cases}$$

Recall that the expected size of $|A_r|$ is $\frac{M}{L}$. Hence, the bias towards the distinguished allele is the difference between the two values which comes out to be $b = \Omega(M/L)$. The value of M should be chosen so that the random drift in the values of $S(A_r)$ should be less than the bias. Therefore, we require that for $r \neq t_1$, the events

$$S(A_{t_1}) < E[S(A_{t_1})] - b/2 \quad \text{and} \quad S(A_r) > E[S(A_r)] + b/2$$

will occur with small probability. This should be true for all L partitions in all N components. We achieve this by choosing M large enough so that the probability of each of these events is less than $O(\epsilon/NL)$ for some $\epsilon > 0$. The algorithm will then find the correct solution with some constant probability.

The value of $S(A_r)$ can be written as $S(A_r) = f(A_r) + N(A_r)$ where $f(A_r) = \sum_{x \in A_r} f(x)$ is the contribution of the true fitness value and $N(A_r) = \sum_{x \in A_r} e(x)$ is the noise. First note that $E[N(A_r)] = 0$ and hence $E[S(A_r)] = E[f(A_r)]$. We guarantee that the above conditions will be satisfied by requiring that all the following events have probability less than $O(\epsilon/NL)$:

1. $f(A_{t_1}) < E[f(A_{t_1})] - b/4$
2. $f(A_r) > E[f(A_r)] + b/4$ for $r \neq t_1$.
3. $|N(A_r)| > b/4$ for all r .

We begin with the first two events. Observe that the random variable $f(A_{t_1})$ is a binomial random variable over $(N-1)M/L$ experiments with success probability $1/L$. The Chernoff bound [1, Th. A.13] tells us that

$$\Pr[f(A_{t_1}) < E[f(A_{t_1})] - \alpha\sqrt{(N-1)M/L}] < e^{-\alpha^2/2}$$

Hence, taking $M = 16N \log(NL/\epsilon)$ and $\alpha = \sqrt{\log(NL/\epsilon)}$ will guarantee that

$$\Pr[f(A_{t_1}) < E[f(A_{t_1})] - b/4] < \epsilon/NL$$

The same M will satisfy the second condition as well.

We now analyze event (3). For simplicity assume that the noise is a Gaussian random variable with deviation σ . Since a sum of Gaussian variables is Gaussian, the random variable $N(A_r)$ is a Gaussian random variable with deviation $O(\sqrt{M/L}\sigma)$. Taking $M = 16L\sigma^2 \log(NL/\epsilon)$ will ensure that

$$\Pr[|N(A_r)| > b/4] < \epsilon/NL$$

We see that taking $M = O((N + L\sigma^2) \log(NL/\epsilon))$ will guarantee that the algorithm outputs the correct results with probability ϵ . When $\sigma < \sqrt{N/L}$ the first term dominates and the algorithm is not effected by the noise. When $\sigma > \sqrt{N/L}$ the algorithm will suffer a slowdown proportional to σ^2 . In the previous section we observed a similar behavior for the GA. However the MFA algorithm, because it is oblivious, can be defeated by problems with less symmetry which the GA can solve.

8 Learning Half Spaces with integer weights

In this section we consider the “Ising Perceptron Problem”, aka the problem of learning half spaces of integer weight [22, 23, 8, 4]. Let $t \in \{1, -1\}^N$ be an N -dimensional vector and let H_t be the hyperplane $(x, t) \geq 0$ passing through the origin. We are given a number of samples $x_1, \dots, x_s$ in the N -dimensional unit ball classified according to whether they are in H_t or not in H_t . The samples were chosen uniformly and independently in the unit ball. A result of [8] shows that $1.44N$ random samples suffice to uniquely determine t . Although t is uniquely determined, it is of course NP-hard in general to find a Boolean hyperplane consistent with a set of examples. [23] gave an algorithm which efficiently determined t using $O(N \log N)$ random examples. We show that the Ising Perceptron problem can be reduced to a noisy ASP problem, and give some evidence that our GA can solve it using only $O(N)$ random examples.

Let $u \in \{1, -1\}^N$ be some N -dimensional vector. Define the fitness $h(u)$ of u to be

$$h(u) = \frac{\text{volume}(H_t \triangle H_u)}{\text{volume}(D_N)}$$

where $H_u \triangle H_t$ is the symmetric difference of H_u and H_t in the N -dimensional unit ball and D_N is the N dimensional unit ball. It should be clear that $h(u)$ depends only on the angle between u and t . Let $f(u)$ be the number of components in which u and t match. Then

$$\begin{aligned} h(u) &= \frac{1}{\pi} \arccos\left(\frac{(u, t)}{N}\right) \\ &= \frac{1}{\pi} \arccos\left(\frac{f(u) - (N - f(u))}{N}\right) \\ &= \frac{1}{\pi} \arccos\left(\frac{2f(u)}{N} - 1\right) \end{aligned}$$

Solving for $f(u)$ we get

$$f(u) = \frac{N}{2} [\cos(\pi h(u)) + 1]$$

Note that $f(u)$ is exactly the ASP fitness function. We can approximate the value of $h(u)$ by picking s random samples $x_1, \dots, x_s$ in the N -dimensional unit ball and then compute $\hat{h}(u)$ defined by:

$$\hat{h}(u) = \frac{1}{s} |\{x_i \in H_t \triangle H_u\}|$$

With $\hat{h}(u)$ we may compute a noisy version of $f(u)$.

Thus, we see that the Ising Perceptron Problem reduces to a noisy ASP problem. To use our results from the previous section we must make sure that the error values of different strings are independent. This can be achieved by selecting $O(N)$ random examples each time our GA needs to estimate the fitness of a string. This method requires $O(N^2)$ examples to learn the Ising Perceptron.

We conjecture that one could use our GA to learn the Ising Perceptron using only $O(N)$ examples by recycling examples to compute the fitness of many strings. This can be done by collecting a single set of $O(N)$ random examples and using them repeatedly to estimate the fitness of each string. The correlations which will exist do not seem critical intuitively, but defeat our current proof techniques.

Consequently, we performed numerical experiments to determine whether $O(N)$ examples suffice for small N . In our experiment we used $1 - \hat{h}(u)$ as the fitness function rather than $f(u)$. We used the genetic algorithm described in §4.2, with the modification that instead of breeding until we have M strings of fitness at least $(p + \sigma)N$, we bred $6M$ strings and kept the M strings with the largest fitness. In cases where the GA is effective, both methods produce a new population in $O(M)$ steps with fitness increased by $O(\sigma N)$ and the second method may be used with more general fitness functions, such as $1 - \hat{h}(u)$, since it does not require the calculation of a fitness threshold.

We ran experiments with a population size of $N \log N$ using kN sample points with k between 2 and 5. For values of N up to 800, $k = 3$ was sufficient to achieve at least 90% accuracy in finding the correct hyperplane. The results are given below as number of successes/total number of trials:

	N				
	50	100	200	400	800
2	6/10	4/10	1/10	0/10	
k 3	10/10	10/10	9/10	10/10	10/10
4	10/10	10/10	10/10	10/10	
5	10/10	10/10	10/10		

These results are consistent with the conjecture that the GA can solve the "Perceptron Problem" with $O(N)$ examples.

9 Conclusion

We have set out to explore when, whether, and why GA's may provide more effective optimization algorithms than alternative approaches. We reviewed the Schema Theorem, which is the main intuition driving this hope. We exhibited an example, which we call ASP, where the Schema Theorem is fully satisfied, and yet the GA based on it is inefficient. We suggested a different breeding approach called Culling which on general grounds should frequently be more effective than standard GA's. We showed that Culling is near optimal on ASP, far better than standard Hill Climbing approaches or GA's, and is highly noise tolerant. Culling is the best algorithm known in some noise regimes. These are the first results we are aware of demonstrating that GA's can beat competing approaches. We have studied a number of competing approaches, and found one, parallel binary search, which can achieve Holland's 'intrinsic parallelism' on some problems where GA's can not, e.g. k -ASP. An interesting open problem is to extend these analyses to other, more complex fitness functions.

Acknowledgments

The authors would like to thank L. Gurvits, M. Mitchell, S. Rao, J. Reif, and W. D. Smith for helpful conversations.

References

- [1] N. Alon, J. H. Spencer: The probabilistic Method. John Wiley and Sons, New York, (1992).
- [2] T. Back, H-P Schwefel, "An overview of evolutionary algorithms for parameter optimization", *Evolutionary Computation* 1(1)pp 1-23, (1993).
- [3] E. B. Baum. "On genetic algorithms", NEC Internal Report June 1994.
- [4] E. B. Baum, Y. Lyuu, "The Transition to Perfect Generalization in Perceptrons", *Neural computation* 3, pp. 386-401.
- [5] V. Chvatal, "MasterMind", *Combinatorica* 3 (1983), pp. 325-329.
- [6] S. Forrest and M. Mitchell: What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. *Machine Learning*, to appear (1994).
- [7] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading MA, 1989.
- [8] D. Haussler, M. Kearns, S. Seung, N Tishby, "Rigorous Learning Curve Bounds from Statistical Mechanics", *Proceedings of COLT 1994*, pp. 76-85.
- [9] D. Hillis: "Co-evolving parasites improve simulated evolution as an optimization procedure." *Physica D* 42 (1990) 228-234.
- [10] W. Hoeffding: Probability inequalities for sums of bounded variables. *J. American Statistical Association*, 58:13-30, 1963.
- [11] J. H. Holland: "Properties of the bucket brigade algorithm". In *Proceedings of the first international conference on genetic algorithms and their applications*, pp 1-7, Lawrence Erlbaum Associates (1985).
- [12] J. H. Holland: *Adaptation in natural and artificial systems*, MIT Press, Cambridge MA, (1975).
- [13] S-K Ma: *Modern Theory of Critical Phenomena*, W. A Benjamin, Reading MA 1976.
- [14] M. Mitchell, J. H. Holland, S. Forrest : When will a genetic algorithm outperform hillclimbing, in *Advances in Neural Information Processing Systems 6*, eds. J. D. Cowan, G. Tesauro, and J. Alspector, Morgan Kaufmann, San Mateo CA, (1994) pp 51-58.
- [15] Y. Rabinovich: Quadratic dynamical systems and theoretical aspects of genetic algorithm, Ph.D. Thesis, Hebrew university of Jerusalem, 1993.
- [16] I. Rechenberg: "Cybernetic solution path of an experimental problem", ROy. Aircr. Establ., libr. transl. 1122. Hants, UK.: Farnborough (1965)
- [17] I. Rechenberg: "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution", Stuttgart: Frommann-Holzboog (1973).
- [18] A. Renyi: Collected works, pp. 633-638.
- [19] Y. Rabinovich, A. Sinclair, A. Wigderson: Quadratic dynamical systems, *Proceedings of the 33rd IEEE FOCS* (1992), pp. 304-313.
- [20] C. E. Shannon (1951) Prediction and entropy of printed English, *Bell System Technical J.* pp50-64, January.
- [21] H.-P. Schwefel. *Numerical optimization of computer models*. Chichester: Wiley.
- [22] S. Venkatesh: On learning binary weights for majority functions, *proceedings of COLT 1991*.
- [23] S. Venkatesh: On the average tractability of binary integer programming and the curious transition to perfect generalization in learning majority functions, *Proceedings of COLT 1993*, pp. 310-316.