Where Genetic Algorithms Excel*

Eric B. Baum
eric@research.nj.nec.com

Dan Boneh
dabo@cs.princeton.edu

Charles Garrett
garrett@research.nj.nec.com

NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

We analyze the performance of a Genetic Algorithm (GA) we call Culling and a variety of other algorithms on a problem we refer to as Additive Search Problem (ASP). ASP is closely related to several previously well studied problems, such as the game of Mastermind and additive fitness functions. We show that the problem of learning the Ising perceptron is reducible to a noisy version of ASP. Culling is efficient on ASP, highly noise tolerant, and the best known approach in some regimes. Noisy ASP is the first problem we are aware of where a Genetic Type Algorithm bests all known competitors. Standard GA's, by contrast, perform much more poorly on ASP than hillclimbing and other approaches even though the Schema theorem holds for ASP. We generalize ASP to k-ASP to study whether GA's will achieve 'implicit parallelism' in a problem with many more schemata. GA's fail to achieve this implicit parallelism, but we describe an algorithm we call Explicitly Parallel Search that succeeds. We also compute the optimal culling point for selective breeding, which turns out to be independent of the fitness function or the population distribution. We also analyze a Mean Field Theoretic algorithm performing similarly to Culling on many problems. These results provide insight into when and how GA's can beat competing methods.

1 Introduction

Genetic Algorithms (GA's) have been widely studied ([8, 13]) 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.

Consider the following problem, which we call ASP for 'Additive Search Problem'. Let $X \equiv \{1, 2, ..., L\}^N$. We have an oracle that 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 $N > L$. ASP is related to the "Royal Road Function" of [17] (c.f. our Appendix A).

*This paper is an expanded version of "On Genetic Algorithms"[4] that appeared in COLT'95 copyright 1995 by ACM Inc. We have now calculated the optimal culling point and rewritten the analysis from this point of view. We also include detailed proofs, and appendices omitted from the conference paper, and expand discussion of the k-ASP problem and its relevance to the Schema Theorem.

ASP is also identical to the well known 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.

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 that, on average, is "fitter" than the last generation's. Iterating this appropriately can be shown to result in a perfect string after about $O(NL\log^2 L \log N)$ queries. Moreover a heuristic analysis indicates that a GA will produce the 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. In addition, a genetic approach is well suited for a parallel machine. Our method solves ASP in time $O(\sqrt{N}\log L)$ on a parallel machine.

Our genetic algorithm is very robust to noise. For example, say the oracle returns a value that 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. For example, we show that the performance of a simple zero temperature annealing algorithm degrades so substantially, that for some noise distributions it does not converge at all. This represents the first problem we are aware of on which a Genetic Algorithm has been shown more effective than all known competitors[1].

The robustness of our genetic algorithm allows us to extend our results to in various directions. The problem of learning a half space with integer weights from examples drawn from a uniform distribution has been studied in [28, 29, 9, 5]. We show that this problem can be directly mapped into a noisy ASP problem, yielding some solution techniques.

Alternatively consider the problem a "Classifier System" [12] faces in learning from interaction with a complex environment. In complex environments one may have little *a priori* 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.

Historically the main motivation[2] 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

[1] C.f. [17] for a survey of the literature.

[2] Another naive appeal of GA's is the analogy to biological evolution. We discount this. Firstly, it is far from an isomorphism, and it is plausible that any of a number of missing factors could be crucial to the putative 'efficiency' of biological evolution. Secondly, there is little hard evidence that biological evolution *is* efficient. Evolution has produced our minds, but using immense computational resources, and has produced many 'suboptimal' developments like, e.g. peacock's tails, in the process. Certainly the history of life provides little reason to conclude that crossover based evolution utilizing a large population should be more computationally efficient than stochastic hillclimbing utilizing only one candidate hypothesis.

on ASP, requiring $\Omega(N^2 \log N)$ queries, illustrating that the schema theorem can fail to translate into algorithmic efficiency even on a problem where useful schemata are never broken by crossover. As we shall see, when applying the standard genetic algorithm to solve ASP, the expected fitness 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.

The fact that standard GA's can take work $\Omega(N^2)$ on a simple problem where the schema theorem would seem applicable but random hill climbing takes work $O(N)$ should serve as a warning against uncritical application of GA's.

'Implicit 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 7 with regard to a generalization of ASP called k-ASP. In k-ASP the components are divided into N/k groups of k components. The fitness of a string is k times the number of groups which agree with a target string. A group which is partially correct adds nothing to the fitness. Thus k-ASP is a highly non-linear, non-additive fitness function. We are not told which components belong to which groups.

Our conclusion is that the GA's do not succeed in realizing gains due to 'implicit parallelism' on this problem. To be more specific, if a GA were able to search effectively in parallel over the relevant schema, we argue that it should regard the k-ASP problem like an ASP problem with L replaced by L^k. But since our genetic approach takes a number of generations (and conjecturally a number of queries as well) depending only logarithmically on L, this means it should suffer a slowdown only linear in k. In fact, the GA we have analyzed, and others we have experimented with, seem to suffer a slowdown exponential in k. We propose an explanation for this, namely that the destruction of an exponential *fraction* of the potentially relevant schemata in the crossover operation is cancelling out the fact that exponentially many schemata are searched in parallel. We also describe a binary search based algorithm we call "Explicitly Parallel Search" that *is* able to exhibit slowdown linear in k. In contrast to GA's that search 'implicitly' in parallel, this algorithm maintains explicit knowledge of the successful schemata so far discovered. Thus it is able to achieve a highly parallel search without destroying previously discovered schemata.

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 conjectured performance on ASP and k-ASP, and thus represent an interesting alternative. They do, however, perform worse in some less symmetric situations.

Algorithms closely related to Culling[3] called "Evolution Strategies" have been proposed and studied by I. Rechenberg [20],[21] and H. P. Schwefel[26]. Evolution strategies use truncation selection. In truncation selection, one breeds only certain fraction of the best members of the population. In culling, one breeds only members better than a threshold. So, e.g., one might perform truncation selection where one keeps members better than the median, and culling where one keeps members better than the mean. These can lead to rather different outcomes. It is not obvious which would be better in practice, but culling can sometimes be easier to analyze, and seems to be easier to optimize, which may give it an edge.

[3] Culling was discovered 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.

In §3.1 we calculate the optimal culling threshold, that is at which point one should cull to achieve the maximum gain in fitness per oracle query, i.e. 'birth'. The result is that one should in fact cull all descendants of less than mean fitness. This seems natural, but was not obvious to us[4]. Remarkably this result holds independent of the fitness function or the distribution of fitnesses of offspring. Aside from application to 'Genetic Algorithms' on computers, this result may interest those engaged in animal husbandry and horticulture. This result distinguishes culling from other Evolution Strategies, or truncation selection, where the optimal procedure is distribution dependent.

For a recent review of the Evolution Strategies literature see [2]. Rechenberg [21] 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[26] gave an expression for expected progress rate of related algorithms, but without recombination (crossover), and studied as well proposals for dynamically changing mutation rates. Kimura and Crow [14, 15] also studied truncation selection. In [15] they compare *truncation selection* to more gradual methods of selection. Their work is largely orthogonal to ours. They study an individual loci in a diploid model with no consideration of population size.

Muhlenbein and Schlierkamp-Voosen[18] have previously showed results similar to those we describe in § 3. They showed that for the ONEMAX fitness function (equivalent to ASP with $L \equiv 2$) truncation selection leads to an expected gain per generation of $O(\sqrt{N})$, and thus leads to faster convergence than standard genetic algorithms.

Standard genetic algorithms were first rigorously analyzed in [19, 23]. These authors view GA's as quadratic dynamical systems. The GA's we analyze use multiway breeding and hence can't be modeled as a QDS.

In summary we have studied the behavior of a variety of algorithms on some natural search problems. These include not only the ASP problem, but also generalizations including noise and k-ASP which are non-linear and non-additive. We have proposed a new Genetic Algorithm and analyzed its performance on these problems. Our analysis uses submartingale techniques that may have some intrinsic, mathematical interest. We have compared our Culling algorithm to Evolutionary Strategies and calculated the optimal culling point, which in contrast to evolutionary strategies is, in a sense, independent of the fitness function or distribution of fitnesses in the population. 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 that can be modeled as noise. We have in fact exhibited a simple problem where a GA is the strongest known approach. On the other hand, standard genetic algorithms were seen to be quadratically slow on a simple problem and exponentially slow on a complex problem, where standard schema theorem based intuition might have suggested they would be effective. Finally we have proposed the Explicitly Parallel Search (EPS) algorithm that can achieve the implicit parallelism motivating GA's in some cases (e.g. k-ASP) where GA's can not. EPS is the strongest known algorithm on ASP without noise, and with small amounts of noise, and on k-ASP, and may have other applications.

§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 analyzes the performance of Culling. §6 discusses the noisy ASP problem. §7 discusses two generalizations of ASP. §8 discusses an alternative method for solving ASP that we refer to as the Mean Field Approximation

[4]Indeed our original intuition (reflected in reference [4]) was that one should cull all descendants less fit than a standard deviation above the mean.

(MFA) Algorithm. §9 reduces the problem of learning the Ising Perceptron problem to an ASP problem. §10 is a conclusion.

Appendix A comments briefly on the relationship of ASP to the Royal Road function of Mitchell et al. [17] and comments on their results. Appendix B is independent of the rest of the paper. It comments on the "Coevolving Parasites" mechanism of Hillis [10].

2 Holland's Schema Theorem

In this section we present the standard genetic algorithm approach and recall the original motivation for studying it[5]. Let $A = \{a_1, a_2, ..., a_L\}$ be an alphabet of 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 $F : A^N \to \Re$. 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 and s^2 has components s^1_i for $i \leq j$ and components s^2_i for $i > j$.

We now discuss the original motivation for this approach. A 'schema' 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 that 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

[5]The results of this section are due to [13]. A simple, clear presentation may be found in [8], chapter 2.

expected number of examples of H in P'. Then one can readily see:

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

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([8, p41])

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

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 $M = 2^{l_0/2}$. $M = 2^{l_0/2}$ is chosen here to simplify the count since for $M >> 2^{l_0/2}$ there will be duplication of many of the small schema ([8, p41]). We do not understand, however, the motivation for calling this an $O(M^3)$ scaling relationship. Typically M will grow at most polynomially with N. In this case, for any fixed ϵ, since l_0 grows linearly with N one finds the number of schemata implicitly considered grows exponentially with 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 "implicit parallelism". The lure of utilizing this efficiency has been the main intuition driving the field of Genetic Algorithms.

3 Selective breeding strategies

In this section we describe the breeding mechanism that we refer to as *culling*, explain why culling often converges faster than breeding proportional to fitness. For a similar result (described above in the introduction) regarding truncation selection, see also [18].

Let P be the current population and let $f(i)$ denote the fitness of the ith string in the population. We denote by $\Pr[f(i) = x]$ the fraction of strings in P with fitness x. Set $\mu = \frac{1}{M}\sum_{i \in P} f(i)$ to be the average fitness and σ to be the standard deviation of the fitness of elements in P. Let μ'_H be the average fitness after sampling proportional to $f(i)$ as suggested by Holland. Then

$$\mu'_H = \frac{1}{M\mu}\sum_{i \in P} f^2(i) = \frac{1}{M\mu}\sum_x x^2 M \Pr[f(i) = x] = \frac{1}{\mu}(\sigma^2 + \mu^2) = \mu + \sigma^2/\mu$$

The second equality follows from the fact that $M\Pr[f(i) = x]$ is the number of elements in the current population with fitness x. In the ASP problem studied below, one can show that $\mu \sim O(N)$, and $\sigma \sim O(\sqrt{N})$. Roughly speaking, this kind of behavior may occur generically if we have a largely additive fitness function, that is if fitness is reasonably approximated as the sum of N largely uncorrelated terms. Then the gain in fitness per generation, $\mu'_H - \mu$, is of order 1.

Now consider an alternate sampling method that 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[6] from P. Let λ be a numeric parameter. For the moment, let $\alpha = 1$; we will shortly discuss how to optimize the choice of α. The culling breeding method is as follows:

1. Choose parents randomly and uniformly from P. Breed.
2. If child's fitness $\geq \mu + \alpha\sigma$, 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 $\Omega(\sigma) = \Omega(\sqrt{N})$. This is much better than the $O(1)$ gain per generation produced by breeding proportional to fitness.

3.1 Optimal culling threshold

Let G be the expected gain per generation. and S be the expected number of strings tested at every generation. The ratio G/S measures the expected gain per bred string. What choice of λ will optimize G/S?

Denote by $\Pr[f(x) = r]$ the probability that a bred string will have fitness r. Suppose we use a culling threshold of $\mu + \lambda\sigma$. Then the expected number of total strings tested is simply:

$$S(\lambda) = \frac{M}{\Pr[f(x) \geq \mu + \lambda\sigma]}$$

This follows since an expected $1/\Pr[f(x) > \mu + \lambda\sigma]$ strings have to be tested until a string with fitness larger than the culling threshold is found. The expected gain in fitness per generation is

$$G(\lambda) = \frac{1}{\Pr[f(x) \geq \mu + \lambda\sigma]} \int_{\lambda}^{\infty} r\sigma \Pr[f(x) = \mu + r\sigma] dr$$

Thus the ratio G/S is

$$g(\lambda) = G(\lambda)/S(\lambda) = \frac{1}{M} \int_{\lambda}^{\infty} r\sigma \Pr[f(x) = \mu + r\sigma] dr$$

Since the integrand is non-negative, the function $g(\lambda)$ is maximized at $\lambda = 0$. Thus, the optimum culling threshold is simply μ. We therefore modify the culling breeding method above as follows:

Optimal culling algorithm: Let μ be the expected fitness of a string bred from the current population. Do:

1. Choose parents randomly and uniformly from P. Breed.
2. If child's fitness $\geq \mu$, accept. Otherwise, reject the child.

[6]Note we are defining these in terms of the distribution of *children*. We will shortly see that σ is unimportant, because the optimal choice of α is 0. Furthermore, for additive fitness functions, the mean of the child population is the same as of the parent population- and the scaling argument offered is only strictly valid for additive fitness functions anyway. To apply this algorithm in more complex situations one would have to estimate the parameters, for example by producing children until M self-consistently satisfied the cutoff.

We call this the "Optimal culling algorithm" because it achieves the greatest expected gain in fitness per evaluation.

Note that one can contrive problems where making partial progress makes it harder to ultimately solve the problem. For example, one might construct a problem where one starts with a population of strings each having fitness 0. The child of two strings with fitness 0 might be a string of fitness 2 with probability .001, a string of fitness 1 with probability .499, or a string of fitness 0 with probability .5. The child of two parents might have fitness 1 whenever either of its parents has fitness 1. In this case, it would be a mistake to accept any child with fitness 1. One's best bet would be to sit and wait for only fitness 2 children, and solve the problem in a "single generation". The problems considered in this paper do not have this characteristic. In these problems (and generically in many problems one wishes to address) one's expectation is that one is better off with a fitter string. If one is e.g. trying to breed bigger livestock, one typically wishes to maximize the expected gain in size per birth.

Note that this analysis is not so clear for the $\mu - \lambda$ or truncation strategies, which suffer from the disadvantage that the optimal truncation point T depends on the specific distribution $\Pr[f(x) = r]$. If $\Pr[f(x) = r]$ is symmetric, then using $T = .5$ (i.e. breeding the top half the population and discarding the lower half) will essentially be the same as the culling proposed above where we keep the first M strings with fitness greater than the mean fitness, because the median fitness of the M strings will essentially be the mean fitness. Thus a value of T near .5 will be optimal. However, it is clear by reduction to the previous argument, that if $\Pr[f(x) = r]$ is very unsymmetric, so the mean string is not close to the median, that some other T would be preferable.

4 Additive Search Problem

We now turn to the ASP, defined in the introduction. Note that a solution to ASP is an N character 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 [6] 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 algorithm, which we call Explicitly Parallel Search (EPS), 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. The contribution of the fitness on those L sites will then be i with probability $P_i = \binom{L}{i}(1/L)^i((L-1)/L)^{L-i}$. One may readily see $.5 \geq P_i$. Whatever the contribution to the fitness of these sites was before this change, it will change at least half the time. 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, that will involve at most $\log L$ queries. This procedure takes on average $O(\log L)$ queries to find each distinguished allele, until there are L left. When there are $i < L$ alleles left, we must make L/i changes of all the remaining alleles before we discover a change. Thus the last L alleles may require $\sum_{i=1}^{L}(L/i)\log L$ queries, or about $L\log^2 L$ queries. Thus this algorithm takes $O(N \log L + L \log^2 L)$ queries to find the optimum string. We have shown:

Theorem 1 *ASP can be solved in polynomial time using $O(N \log L + L \log^2 L)$ queries. Furthermore, any algorithm that 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 [17]:

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 ([17]) to solve the problem in time $O(LN\log(N))$.

Note that EPS gains a factor of $L/\log L$ in efficiency over RMHC because EPS has appropriate parallelism in its search, i.e. changes an optimized number of components when it queries, and gains an additional factor of $\log N$ because its search is explicit in the sense that it knows which components of its candidate hypothesis it has decided are correct, and so does not experiment with these further. We will discuss a generalization of EPS in §7.1.

Next, we briefly and intuitively analyze the performance of the standard GA described in section 2 when applied to ASP. The basic idea, which we will use repeatedly in this paper, is this. Let $p_{t,i}$ be the fraction of alleles in the population at component i and time t which are correct, and let p_t be the mean of the $p_{t,i}$, hence the fraction of components overall which are correct at time t. Hence Np_t is the mean fitness at time t and $Mp_{t,i}$ is the number of correct alleles at component i in the population at time t. The $p_{t,i}$ each undergo a random walk, but with positive expectation due to selection. By symmetry, the expected value of $p_{t,i}$ is independent of i, and equal the expected value of p_t. The variance from component to component arises because at each generation there are random fluctuations in the gain (or loss) of correct alleles at component i. However, the larger the population is, the smaller in fractional terms is the likely deviation in number of correct alleles at component i and time step t. Indeed, if the population size is large enough so that the total expected deviation in gain over the t generations is much smaller than the expected gain itself, then we will be safely able to approximate that $p_{t,i} = p_t$. Conversely if the population size is so small that the expected deviation is comparable to the expected gain, then some of the $p_{t,i}$ will likely fluctuate to 0. If any of the $p_{t,i}$ should fluctuate to zero, however, we would not be able to solve the problem, and more generally if any $p_{t,i}$ fluctuate very low, they will slow up convergence.

So we will first estimate the expected gain per generation under the assumption that M, the population size, is large enough so that fluctuations from component to component are unimportant. Knowing the expected gain per generation, we can readily estimate the number of generations necessary to solve the problem. Next, we will lower bound the population size necessary to prevent fluctuations of some components to zero. In computing this lower bound, we will assume for simplicity that the population size is large enough that we can treat the $p_{t,i}$ as equal. Rigorous arguments require complex analysis of fluctuations. We will engage in these in analyzing the culling approach, but will argue intuitively here regarding the standard GA. A full analysis (using different techniques) of the performance of this algorithm on a problem similar to ASP can be found in [19].

Since (by the argument sketched in §3) the expected gain in average fitness, Np_t, per generation is of order 1, the standard GA requires $T = \Omega(N)$ generations to solve ASP. In t steps, selection causes a random drift of about[7] $\sqrt{tMp_t}$ in the number of correct alleles per component. If we wish to achieve a finite probability that the correct allele has not vanished from the population at any component, we must demand that the probability of all correct alleles vanishing from a given component be less than $1/N$. The number of correct alleles in a given component is distributed binomially. To achieve a probability of order $1/N$ of a fluctuation the size of the mean, we must demand that the mean be bigger than the deviation by a factor of about $\log(N)$, i.e. we must require $Mp_t > \log(N)\sqrt{tMp_t}$. Thus the size of the population must be at least $M > \log^2(N)t/p_t$. Since the mean fitness grows by O(1) per generation, p_t increases by $O(1/N)$ 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 $\Omega(N^2 \log^2 N)$.

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

The upshot is, that for fixed L and large N, the standard GA will require a number of oracle calls that is *quadratic* in the size used by hillclimbing. This occurs simply because the GA achieves low gain per generation, and thus requires a large population size to avoid deadly fluctuations. This quadratic penalty warns that one should be careful about one's problem before uncritically selecting a genetic approach.

5 GA Approach Using Culling

In this section we describe and analyze an algorithm for solving ASP based on the culling approach of Section 3.1. Our analysis suggests that the culling algorithm performs better than either the standard GA or RMHC on ASP.

Let $f(x)$ denote the ASP fitness value of a string x. As before, we assume $N \gg L$. Specifically, $N > L^2$. The culling approach of Section 3.1 leads to the following algorithm for ASP that we designate as GA2:

1. Set $t = 0$, $M = \sqrt{N} \log^2 N$.
2. Choose M random strings with fitness at least N/L. Call this set P and let μ be the average fitness of strings in P.
3. Use M-way breeding to breed a new string x. If $f(x) < \mu$ 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.
4. Check if the majority vote word [8] has fitness N. If so, stop.

[7] More precisely, the drift deviation is $\sqrt{\sum_{j=1,t} Mp_{i,j}(1-p_{i,j})}$. We replace $p_{i,j}$ by p_j as discussed above and then one can approximate the deviation by $\sqrt{tMp_t(1-p_t)}$. The important case here is where p_t is small, or more precisely $(1-p_t)$ is bounded away from zero. Including the $(1-p_t)$ factors would change the calculation by at most some small constant factor. We will drop these factors throughout where appropriate to enhance clarity.

[8] The majority vote word contains in the i'th component the allele which appears most often in the i'th component of strings in the current populations. Ties are broken arbitrarily.

5. Set $t := t+1$ and let μ be the new average fitness of strings in P. Set $M = \left(\max\{4N\frac{t}{\mu}, \sqrt{N}\}\right) \log^2 N$ and go to step 3.

The breeding strategy we refer to as M-way breeding[24, 27] 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[9]. Thus, all strings in the population have a chance of contributing to the child.

Notice that the size of the population M is changing as the algorithm progresses. We will show that the size of the population goes down with time. This is useful in reducing the total number of fitness evaluations performed by the algorithm.

5.1 Heuristic running time analysis

We now analyze the performance of algorithm GA2 on ASP. We conjecture that the algorithm solves ASP using $\tilde{O}(N)$ queries and give a heuristic argument to support the conjecture. In the next section we prove a weaker theorem showing that the algorithm solves ASP using $\tilde{O}(NL)$ queries. The proof involves bounding the large deviations of a certain sub-martingale and may be of independent interest.

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

Assuming Conjecture 2 algorithm GA2 solves ASP using less queries than both the hill climbing and the standard genetic approach. Furthermore, algorithm GA2 achieves the lower bound of Theorem 1 up to some log factors. This result shows that the culling approach is very useful for solving ASP and more generally for solving problems where the fitness function is mostly additive.

The rest of this section is devoted to a heuristic argument supporting the conjecture. First we must introduce some notation.

1. The set of strings in generation t will be denoted by P_t.
2. Set $M_t = |P_t|$, the size of the population.
3. Denote by μ_t the mean fitness of strings in P_t.
4. We let $X_{t,i}$ be a random variable counting the number of strings in P_t that contain the correct allele in the i'th component.
5. Set $p_{t,i} = X_{t,i}/M_t$. $p_{t,i}$ is the fraction of strings with correct i-th component in generation t. Note that $\mu_t = \sum_i p_{t,i}$.
6. Set $p_t = \mu_t/N$. Thus p_t is the mean fraction of correct alleles per component $p_t = \frac{1}{N}\sum_i p_{t,i}$.

We can prove under some assumption that the gain in the mean fitness of strings in the population is approximately $\sqrt{Np_t(1-p_t)}$ per generation. This is stated in Corollary 4. The corollary captures the intuition discussed in §3, namely that culling allows one to gain about one standard deviation in fitness per generation. Lemma 5 will then show that with such a gain in fitness per generation, the target string will be found in $O(\sqrt{N})$ generations. That is, since one gains about $\sqrt{N}$ in fitness

[9] M-way breeding avoids correlations that would otherwise obstruct our proof. Culling would probably also work well with standard pairwise uniform crossover, but we are unable to analyze it.

per generation, and since the optimal string has fitness N, it takes about $\sqrt{N}$ generations to find the optimal string. Using this bound on the number of generations and a bound on the population size derive the bound on the total number of queries the given in Conjecture 2.

The question that remains is how large should the population size be to ensure a gain of $\sqrt{Np_t(1-p_t)}$ in fitness per generation. We give an intuitive argument to show that the population size used by algorithm GA2 is sufficient.

Observe that a problem will occur if due to random drift the correct allele in a certain component is wiped out from the population. This will cause the algorithm to fail in finding the target string. When $p_{t,i} < 1/2$ the standard deviation of $X_{t,i}$ is bounded by $\sqrt{tM_tp_t}$. To see this observe that $X_{t,i}$'s deviation is roughly $\sqrt{tM_tp_t(1-p_t)}$ because $X_{t,i}$ represents a deviation over t steps each of variance $M_tp_t(1-p_t)$. By definition, the mean of all the $X_{t,i}$ in generation t is M_tp_t. Note that the larger the population size M_t, the bigger the ratio of the mean to the standard deviation. The risk of a large fluctuation decreases exponentially in this ratio. We choose M_t large enough to make the probability of a fluctuation as large as the mean, less than $1/N^2$. Then it will be unlikely that in any generation t and component i the variable $X_{t,i}$ will fluctuate to zero. Thus we require that

$$M_tp_t > \log(N^2)\sqrt{tM_tp_t}$$

This implies that $M_t > 4\log^2(N)t/p_t$. Indeed, this is the value used by the algorithm since $t/p_t = Nt/\mu$.

We now show that if indeed throughout the algorithm all components $p_{t,i}$ are roughly equal then algorithm GA2 will have the running time stated in Conjecture 2. As was stated above, the larger the population size M is, the more likely this condition is to hold. The heuristic argument above suggests that the population size used in algorithm GA2 is indeed sufficient to ensure that all $p_{t,i}$ are a fixed constant factor away from their mean throughout the algorithm.

The population at generation t can be characterized by the vector $(p_{t,1}, p_{t,2}, \ldots, p_{t,N})$. This vector is useful in describing the fitness of strings generated using M-way breeding from the population P_t. The fitness of the bred strings is distributed as a sum of N independent Bernoulli random variables where the i'th variable has success probability $p_{t,i}$. Thus, without the application of culling the expected fitness of bred strings is μ_t. With the application of culling the expected fitness is higher. The next lemma bounds the gain in expected fitness due to culling. We note that it is likely that the constant factors in our bounds can be sharpened.

Lemma 3 *Let* $\sigma^2 = \sum_{i=1}^N p_{t,i}(1-p_{t,i})$. *Assume* $\sigma^2 \geq \frac{1}{4}Np_t(1-p_t)$. *Then* $\Pr[\mu_{t+1} < \mu_t + \sigma/40] < \frac{1}{N}$.

Proof Let Y be a random variable that is the fitness of a string generated using M-way breeding from the population P_t. The expected fitness of a string that survives the culling is

$$\begin{aligned} E[Y|Y \geq \mu_t] &= \sum_{k=\mu_t}^{N} k \Pr[Y = k \mid Y \geq \mu_t] = \\ & \sum_{k=\mu_t}^{N} \mu_t \Pr[Y = k \mid Y \geq \mu_t] + \sum_{k=\mu_t+1}^{N} (k-\mu_t) \Pr[Y = k \mid Y \geq \mu_t] = \\ & \mu_t + \sum_{k=\mu_t+1}^{N} \Pr[Y \geq k \mid Y \geq \mu_t] \end{aligned}$$

When $\mu_t \le k \le \mu_t + \frac{1}{2}\sigma$ we have that[10]

$$\Pr[Y \ge k | Y \ge \mu_t] = \Pr[Y \ge k] / \Pr[Y \ge \mu_t] \ge \frac{1}{10}$$

Thus, $E[Y|Y \ge \mu_t] \ge \mu_t + \frac{\sigma}{2} \cdot \frac{1}{10} = \mu_t + \sigma/20$. This means that the expected fitness of a string that survives the culling process is at least $\mu_t + \sigma/20$. Since the elements of P_{t+1} are drawn independently at random, the law of large numbers implies that the mean fitness of strings in P_{t+1} should also be at least $\mu_t + \sigma/20$. Formally, we use Hoeffding's inequality to show that it is unlikely that μ_{t+1} is less than $\mu_t + \sigma/40$. Hoeffding's inequality [11] states that if $Z_1, \ldots, Z_r$ are independent and identically distributed random variables with $B_1 \le Z_i \le B_2$ then for any $\rho > 0$

$$\Pr\left[\frac{1}{r}\sum_{i=1}^{r} Z_i < E[Z_1] - \rho\right] \le \exp\left(-2\rho^2 r/(B_2 - B_1)^2\right)$$

In our case let $Z_1, \ldots, Z_{M_{t+1}}$ be independent random variables that measure the fitness of strings which survive the culling threshold. Observe that μ_{t+1} is distributed like $\sum_i Z_i / M_{t+1}$. The above paragraph shows that $E[Z_i] \ge \mu_t + \sigma/20$. Define A to be the event that all Z_i satisfy $\mu_t \le Z_i \le \mu_t + \sigma \log N$. Then we have:

$$\Pr\left[\frac{Z_1 + \ldots + Z_{M_{t+1}}}{M_{t+1}} < (\mu_t + \sigma/20) - \sigma/40\right] \le \Pr\left[\frac{Z_1 + \ldots + Z_{M_{t+1}}}{M_{t+1}} < \mu_t + \sigma/40 \middle| A\right] + \Pr\left[\bar{A}\right] \le$$

$$\exp\left(-2M_{t+1}(\sigma/40)^2/(\sigma \log N)^2\right) + \Pr\left[\bar{A}\right] < \frac{1}{2N} + \Pr\left[\bar{A}\right]$$

The first inequality follows since any two events A, B satisfy $\Pr[B] \le \Pr[B|A] + \Pr[\bar{A}]$. The second inequality is an application of the Hoeffding bound. The last inequality follows from the fact that $M_{t+1} > \sqrt{N}\log^2 N$. We can bound $\Pr\left[\bar{A}\right]$ by applying a Hoeffding type bound to the Z_i. We obtain

$$\Pr[\bar{A}] \le N\Big(1 - \Pr[\mu_t \le Z_1 \le \mu_t + \sigma \log N]\Big) = N \Pr[Y \ge \mu_t + \sigma \log N \mid Y \ge \mu_t] \le \frac{1}{2N}$$

The last inequality follows using the Hoeffding type bound stated in Theorem A.11 of [1, p. 237], the assumption that $\sigma^2 \ge \frac{1}{4}Np_t(1-p_t)$ and the fact that $\Pr[Y \ge \mu_t] > 1/4$ which follows from the same result used in footnote 10. Plugging this bound into the above inequality leads to

$$\Pr\left[\frac{\sum_i Z_i}{M_{t+1}} < \mu_t + \sigma/40\right] < \frac{1}{N}$$

This completes the proof of the lemma. □

[10] The $\frac{1}{10}$ in the last inequality follows from general results on the asymptotic behavior of sums of Bernoulli random variables. Theorem 1.1 of [7] shows that since $N^{-3/4} < p_{t,i} < 1 - N^{-3/4}$ (the population size is less than $N^{3/4}$) it follows that for large enough N

$$\Pr[Y \ge \mu_t] \le 1.1\phi(0) \quad \text{and} \quad \Pr[Y \ge \mu_t + \frac{1}{2}\sigma] \ge 0.9\phi(-1)$$

where $\phi(x)$ is the distribution function of a standard normal $N(0,1)$ random variable. Therefore,

$$\frac{\Pr[Y \ge \mu_t + \frac{1}{2}\sigma]}{\Pr[Y \ge \mu_t]} \ge \frac{0.9\phi(-1)}{1.1\phi(0)} \ge .2596 \ge \frac{1}{10}$$

Corollary 4 *Suppose that $p_{t,i}(1-p_{t,i}) > \frac{1}{4}p_t(1-p_t)$ for all i. Then*

$$\Pr\left[p_{t+1} - p_t < \epsilon\sqrt{p_t(1-p_t)/N}\right] < \frac{1}{N}$$

for some constant $\epsilon > 0$.

Proof The assumption that $p_{t,i}(1-p_{t,i}) > \frac{1}{4}p_t(1-p_t)$ for all i implies that the σ defined in Lemma 3 satisfies $\sigma > \sqrt{\frac{1}{4}Np_t(1-p_t)}$. Hence,

$$\Pr\left[p_{t+1} - p_t < \frac{1}{80}\sqrt{p_t(1-p_t)/N}\right] < \Pr\left[\mu_{t+1} - \mu_t < \sigma/40\right] < \frac{1}{N}$$

where the last inequality follows from Lemma 3. Setting $\epsilon = 1/80$ proves the corollary. $\square$

The assumption of the corollary clearly holds if all $p_{t,i}$ are sufficiently close to their mean p_t, e.g. $(1-\epsilon)p_t \leq p_{t,i} \leq (1+\epsilon)p_t$ for some fixed $\epsilon > 0$. Thus, under this condition, the corollary shows that with high probability in every generation the gain is at least

$$p_{t+1} - p_t \geq \epsilon\sqrt{p_t(1-p_t)/N}$$

Using this bound we can estimate the number of generations until the algorithm terminates.

Lemma 5 *Let $\epsilon > 0$ be some fixed constant. Define the sequence $q_1 = \frac{1}{L}$ and $q_{t+1} = q_t + \epsilon\sqrt{q_t(1-q_t)/N}$. Then the number of generations until $q_t = 1$ is no more than $O(\sqrt{N})$.*

Proof We ask first how many generations until $q_t = 1/2$. Disregard the $(1-q_t)$ term for simplicity as it will change the result by at most a factor of 2. Similarly we may disregard the ϵ factor on the gain since it will effect the result by at most a constant factor of $\frac{1}{\epsilon}$.

Start with $q_1 = 1/L$. At each generation, to begin with, q_t increases by more than $\sqrt{1/LN}$. So after $\sqrt{N/L}$ generations, q_t has more than doubled, i.e. $q_t > 2/L$. Now at each generation q_t increases by more than $\sqrt{2/LN}$. After $\sqrt{2N/L}$ generations, q_t has at least doubled again, to $4/L$. Iterating this procedure $\log L$ times we find it takes a total of less than

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

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

Under the assumption of Corollary 4 we have that throughout the algorithm $p_t > q_t$ with high probability. The algorithm will terminate when $p_t = 1$, i.e. when $\mu_t = N$. Thus, combining Corollary 4 and Lemma 5 shows that the algorithm will terminate after $O(\sqrt{N})$ generations with high probability. This explains the first part of Conjecture 2. Next we analyze the number of queries the algorithm will perform.

Our culling criteria guarantees that at most a constant fraction of the strings generated will be rejected at every generation. Hence, the total number of queries the algorithm makes is $O(\sum_t M_t)$. We now bound this sum.

Lemma 6 *Let $\epsilon > 0$ be some fixed constant. Define the sequence $q_1 = \frac{1}{L}$ and $q_{t+1} = q_t + \epsilon\sqrt{q_t(1-q_t)/N}$. Let $\hat{M}_t = \max\{(t/q_t), \sqrt{N}\}\log^2 N$. Then*

$$\sum_t \hat{M}_t = O(N \log^2 N \log L)$$

Proof By Lemma 5 the sum can be truncated when $t = O(\sqrt{N})$. Thus, once $q_t > \frac{1}{2}$ we have that $\hat{M}_t < O(\sqrt{N}\log^2 N)$ implying that from that point on the algorithm will make at most $O(N\log^2 N \log L)$ queries. We show that this is also the number of queries in the duration where $q_t < \frac{1}{2}$.

The proof of Lemma 5 shows that q_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 $t\hat{M}_t = t^2/q_t \log^2 N$. Now, the doubling period argument from before can be used to show that throughout the process $t^2/q_t = O(N)$. 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. □

Under the assumption of Corollary 4 we know that $p_t > q_t$ with high probability throughout the algorithm. This means that $M_t < \hat{M}_t$. Hence, with high probability the total number of queries done by the algorithm is bounded by $\sum_t \hat{M}_t$. Hence, Lemma 6 explains the second half of Conjecture 2.

5.2 Rigorous running time analysis

The previous section presented a heuristic estimate for the running time of the algorithm. To make the argument precise one needs to bound the distance from the $p_{t,i}$'s to their mean p_t. In this section we prove such a bound at the cost of increasing the population size M. As a result we obtain a weaker running time estimate. We denote by GA2' the algorithm GA2 in which the population size M is fixed to $M = 128L\sqrt{N}\log N \log L$ throughout the algorithm.

Theorem 7 *Algorithm GA2' will solve the ASP problem with constant probability in $O(\sqrt{N}\log L)$ generations and $O(NL\log N\log^2 L) = \tilde{O}(NL)$ queries.*

Notice that due to step 4 in algorithm GA2 (majority vote step) it suffices to show that in $O(\sqrt{N})$ generations all $p_{t,i}$ satisfy $p_{t,i} > \frac{1}{2}$ with constant probability. We do this by proving a large deviation bound on the $X_{t,i}$. The following lemma analyzes the gain in $X_{t,i}$ that is caused by both M-way breeding and culling.

Lemma 8 *Given $X_{t,i}$, the value of $X_{t+1,i}$ is binomially distributed over M_{t+1} experiments and success probability q where*

$$q \geq \min\left(1, p_{t,i} + \frac{p_{t,i}(1-p_{t,i})}{4\sqrt{Np_t(1-p_t)}}\right)$$

Proof First, since the components of the strings in the population P_{t+1} are generated independently at random it is clear that $X_{t,i}$ is binomially distributed. The question is what is the success probability q. The culling mechanism will ensure that $q > p_{t,i}$. We will give a lower bound on q.

The problem can be stated as follows: we are given N independent Bernoulli random variables, $Z_1, \ldots, Z_N$ corresponding to the N components of the strings. Let $p_{t,i}$ be the success probability of the i'th Bernoulli random variable, $i = 1, \ldots, N$. Each Z_i corresponds to the event that a randomly bred string from generation t will contain the correct allele at component i. Then since the culling threshold is μ_t, we have:

$$q = \Pr[Z_i = 1 \mid \sum_{j=1}^{N} Z_j > \mu_t]$$

We provide a lower bound for q as follows:

$$\begin{aligned}
q &= \Pr[Z_1 = 1 | \sum_{i=1}^{N} Z_i \geq \mu_t] = p_{t,1} \frac{\Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t - 1\right]}{\Pr\left[\sum_{i=1}^{N} Z_i \geq \mu_t\right]} = \\
& p_{t,1} \frac{\Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t - 1\right]}{p_{t,1} \Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t - 1\right] + (1 - p_{t,1}) \Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t\right]} = \\
& \frac{p_{t,1}}{p_{t,1} + (1 - p_{t,1}) \frac{\Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t\right]}{\Pr\left[\sum_{i=2}^{N} Z_i \geq \mu_t - 1\right]}} = \\
& \frac{p_{t,1}}{p_{t,1} + (1 - p_{t,1}) \left[1 - \Pr[\sum_{i=2}^{N} Z_i < \mu_t \mid \sum_{i=2}^{N} Z_i \geq \mu_t - 1]\right]} = \\
& \frac{p_{t,1}}{1 - (1 - p_{t,1}) \Pr\left[\sum_{i=2}^{N} Z_i < \mu_t \mid \sum_{i=2}^{N} Z_i \geq \mu_t - 1\right]}
\end{aligned}$$

Hence, it suffices to lower bound

$$\Pr\left[\sum_{i=2}^{N} Z_i < \mu_t \;\middle|\; \sum_{i=2}^{N} Z_i \geq \mu_t - 1\right] = \Pr\left[\sum_{i=2}^{N} Z_i = \lceil \mu_t \rceil - 1 \;\middle|\; \sum_{i=2}^{N} Z_i \geq \mu_t - 1\right]$$

Let R be the random variable $R = \sum_{i=2}^{N} Z_i$. The standard deviation of R is maximal when all Z_i come from the same distribution, i.e. all the $p_{t,i}$ are equal. Thus we see that $\sigma(R) < \sqrt{Np_t(1-p_t)}$. Let $\mu = \lceil \mu_t \rceil - 1$. Then

$$\Pr[R = \mu \mid R \geq \mu_t - 1] = \frac{\Pr[R = \mu]}{\Pr[R \geq \mu_t - 1]} \geq \frac{1}{4\sigma(R)} > \frac{1}{4\sqrt{Np_t(1-p_t)}}$$

The second inequality follows from the fact that for sufficiently large N when $\sigma(R) > N^{1/4}$ we have $\Pr[R = \mu] \geq \frac{1}{4\sigma(R)}$. This can be seen using the asymptotic behavior of sums of Bernoulli random variables(see e.g. [7]). For large enough N the $\frac{1}{4}$ can be replaced by any constant less than $\frac{1}{\sqrt{2\pi}}$.

Plugging this bound into the above expression for q leads to

$$\begin{aligned}
q &= \Pr\left[Z_1 = 1 \mid \sum_{i=1}^{N} Z_i \geq \mu_t\right] \geq \frac{p_{t,1}}{1 - \frac{1-p_{t,1}}{4\sqrt{Np_t(1-p_t)}}} = \\
& p_{t,1} + \frac{p_{t,1}(1-p_{t,1})}{4\sqrt{Np_t(1-p_t)} - (1-p_{t,1})} \geq p_{t,1} + \frac{p_{t,1}(1-p_{t,1})}{4\sqrt{Np_t(1-p_t)}}
\end{aligned}$$

□

Notice that when all the $p_{t,i}$'s are equal, we have that $p_t = p_{t,i}$ in which case the gain is $\frac{1}{4}\sqrt{p_t(1-p_t)/N}$. The above lemma shows that $X_{1,i}, X_{2,i}, \ldots, X_{t,i}$ forms a submartingale and provides a bound on the gain between the different generations. We wish to prove a large deviation bound for this submartingale. This can be achieved using Azuma's inequality [1, p. 85]. However, since Azmua's bound holds for any martingale, the resulting bound is too weak. Thus, we need to prove an Azuma type inequality for our special submartingale.

Unfortunately, Lemma 8 shows that the gain in $X_{t,i}$ depends on all of $X_{t-1,1}, \ldots, X_{t-1,N}$. At the present we can not derive a large deviation bound for a submartingale with such dependence. We therefore simplify the bound given in Lemma 8 by weakening it. The simplification is the reason why Theorem 7 is weaker than Conjecture 2. Observe that $p_t(1-p_t)$ is upper bounded by $\frac{1}{4}$. It follows that when $p_{t,i} < 3/4$ the bound of Lemma 8 can be simplified as follows:

$$q \geq \min\left(1, p_{t,i} + \frac{p_{t,i}(1-p_{t,i})}{4\sqrt{Np_t(1-p_t)}}\right) \geq \min\left(1, p_{t,i} + \frac{p_{t,i}}{8\sqrt{N}}\right) \geq \min\left(1, p_{t,i}(1 + \frac{1}{8\sqrt{N}})\right)$$

Hence, it makes sense to define a new submartingale $Y_0, Y_1, \ldots$. The random variable Y_t given Y_{t-1} is distributed as a binomial random variable over M experiments with success probability $g(Y_{t-1})$ where $g(Y)$ is the function

$$g(Y) = \min\left[c\frac{Y}{M}, 1\right] \quad \text{where} \quad c = 1 + \frac{1}{8\sqrt{N}}$$

and $Y_0 = M/L$. By the simplified version of Lemma 8 it follows that for all i and t we have $X_{t,i} \geq Y_t$, i.e. $\Pr[X_{t,i} > k] > \Pr[Y_t > k]$ for any constant k. Therefore, it suffices to derive a large deviation bound for the submartingale Y_t. This is done in the following lemma.

Lemma 9 *Set $p_0 = M/L$. For any t and $\lambda > 0$*

$$\Pr[Y_t < p_0 c^t - \lambda] < e^{-\frac{1}{2}\lambda^2/p_0 c^{2t} t}$$

Proof We bound the generating function $E[e^{-\alpha Y_t}]$.

$$\begin{aligned} E[e^{-\alpha Y_t}] &= E\left[E[e^{-\alpha Y_t}|Y_{t-1}]\right] = E\left\{\left[1 - g(Y_{t-1}) + e^{-\alpha}g(Y_{t-1})\right]^M\right\} = \\ &= E\left[\left(1 - \frac{g(Y_{t-1})(1-e^{-\alpha})M}{M}\right)^M\right] \leq E\left[e^{-(1-e^{-\alpha})g(Y_{t-1})M}\right] \leq E\left[e^{-\beta Y_{t-1}}\right] \end{aligned}$$

Where $\beta = c(1-e^{-\alpha})$. We intend to apply the exact same procedure to the variable Y_{t-1} and bound $E\left[e^{-\beta Y_{t-1}}\right]$ in terms of $E\left[e^{-\gamma Y_{t-2}}\right]$ where $\gamma = c(1 - e^{-c(1-e^{-\alpha})})$. We iterate this derivation until we reach Y_0. To state the result we must first define an auxiliary function:

$$h_0(\alpha) = \alpha \quad \text{and} \quad h_k(\alpha) = c\left(1 - e^{-h_{k-1}(\alpha)}\right)$$

Claim 10 *for all $\alpha > 0$ and positive integers k we have*

$$h_k(\alpha) \geq c^k\alpha - \frac{k}{2}c^{2k}\alpha^2$$

We prove the claim by induction on k using the fact that all $\alpha > 0$ satisfy

$$1 - e^{-\alpha} \leq \alpha \quad \text{and} \quad 1 - e^{-\alpha} \geq \alpha - \frac{1}{2}\alpha^2$$

The first fact implies that $h_k(\alpha) \leq c h_{k-1}(\alpha) \leq \ldots \leq c^k \alpha$. Using the second fact and the inductive hypothesis we obtain

$$\begin{aligned} h_k(\alpha) &= c(1 - e^{-h_{k-1}(\alpha)}) \geq c h_{k-1}(\alpha) - \frac{1}{2} c h_{k-1}(\alpha)^2 \geq \\ &\geq c\left[c^{k-1}\alpha - \frac{1}{2}(k-1)c^{2k-2}\alpha^2\right] - \frac{1}{2}c(c^{k-1}\alpha)^2 = \\ &= c^k\alpha - \left[\frac{1}{2}(k-1)c^{2k-1} + \frac{1}{2}c^{2k-1}\right]\alpha^2 \geq c^k\alpha - \frac{k}{2}c^{2k}\alpha^2 \end{aligned}$$

Hence the claim follows. Armed with the bound of Claim 10 we obtain the following bound on the generating function for Y_t:

$$E[e^{-\alpha Y_t}] \quad \leq \quad E\left[e^{-h_t(\alpha) Y_0}\right] = e^{-h_t(\alpha) p_0} \leq e^{-c^t p_0 \alpha - \frac{1}{2} c^{2t} t \alpha^2 p_0}$$

To prove the lemma we use Markov's inequality and obtain

$$\Pr[Y_t - p_0 c^t < -\lambda] \quad = \quad \Pr[e^{-\alpha(Y_t - p_0 c^t)} > e^{\alpha\lambda}] < E\left[e^{-\alpha(Y_t - p_0 c^t)}\right] e^{-\alpha\lambda} \leq e^{\frac{1}{2}\alpha^2 p_0 c^{2t} t - \alpha\lambda}$$

Plugging in $\alpha = \lambda / p_0 c^{2t} t$ leads to

$$\Pr[Y_t - p_0 c^t < -\lambda] < e^{-\frac{1}{2}\lambda^2 / p_0 c^{2t} t}$$

which is the required result. $\square$

The proof of Theorem 7 is now immediate. Since $c = 1 + \frac{1}{8\sqrt{N}}$ and $p_0 = M/L$ after $t = 8\sqrt{N}\log L$ generations we obtain for all i

$$\Pr\left[X_{t,i} < \frac{M}{2}\right] \leq \Pr\left[Y_t < \frac{M}{2}\right] \leq \Pr\left[Y_t < c^t p_0 - \sqrt{4 p_0 c^{2t} t \log N}\right] \leq \frac{1}{2N} \tag{1}$$

The last inequality follows from Lemma 9. The second inequality follows from the fact that the size of the population is $M = 128 L\sqrt{N}\log L \log N$:

$$c^t p_0 - \sqrt{4 p_0 c^{2t} t \log N} = L\frac{M}{L} - \sqrt{32\frac{M}{L}L^2\sqrt{N}\log L \log N} = M - \sqrt{\frac{M^2}{4}} = \frac{M}{2}$$

In the first equality above we assume for simplicity that c^t is exactly equal to L.

To complete the proof of Theorem 7 we note that inequality (1) proves that after $t = 8\sqrt{N}\log L$ generations all $p_{t,i}$ will be greater than $\frac{1}{2}$ with probability at least $\frac{1}{2}$. Hence, after t generations the majority vote (step 4 of algorithm GA2) will produce the target string with probability at least $\frac{1}{2}$. The theorem now follows since the total number of queries is $tM = O(NL\log^2 L \log N)$. $\square$

6 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 oracle[11]. 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. We denote by σ_e the standard deviation of the noise. We will assume symmetry of the noise as necessary for convenience of the analysis, discussing it when we assume it.

The EPS algorithm described in Section 4 can be made to work in the presence of noise using a result of Renyi [22]. Renyi shows how to perform binary search using an unreliable oracle. However, the performance of the algorithm degrades proportionally to the noise deviation.

6.1 RMHC and Noise

In this subsection we we will see that the RMHC 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. Intuitively the problem is that RMHC often accepts a change to a string x with a high noise component $e(x)$ but mediocre true fitness $f(x)$. Once it has worked itself onto a string with very high noise component, it stalls as few if any neighbors will have higher total noise plus fitness $\tilde{f}$. The RMHC has another intuitive drawback relative to the GA. Since it only changes one component at each step, it is comparing a potential fitness improvement of at most 1 to the noise deviation. By contrast, each new string the GA generates differs on average by about $\sigma >> 1$, thus the GA operates at a far higher level of signal to noise.

Let c be the current favorite string and let $\tilde{f}(c)$ be c's noisy fitness value. Then $\tilde{f}(c)$ is made up of two components: the true fitness, $f(c)$, and the noise, e_c.

When a new string s is examined, its noisy fitness value is again made up of two components: the true fitness, $f(s)$ and the noise e_s. The string s will be chosen as the new CFS if $f(c) + e_c < f(s) + e_s$. Let $\Delta_f = f(s) - f(c)$, and $\Delta_e = e_s - e_c$. The method by which s is chosen guarantees that $|\Delta_f| \leq 1$. Let A be the event that the random string s is chosen as the new CFS, then

$$\begin{aligned}
\Pr[A] &= \Pr[\Delta_f = 1]\Pr[\Delta_e > -1] + \Pr[\Delta_f = 0]\Pr[\Delta_e > 0] + \Pr[\Delta_f = -1]\Pr[\Delta_e > 1] \\
&= \frac{N - f(x)}{NL}\Pr[\Delta_e > -1] + \frac{N - f(x)}{N}\frac{L-1}{L}\Pr[\Delta_e > 0] + \frac{f(x)}{N}\Pr[\Delta_e > 1]
\end{aligned}$$

The expected gain in true fitness of the new current favorite string is given by

$$\begin{aligned}
E[\Delta_f \mid A] &= \frac{1}{\Pr[A]}[\Pr[\Delta_f = 1]\Pr[e_s > e_c - 1] - \Pr[\Delta_f = -1]\Pr[e_s > e_c + 1]] \\
&= \frac{1}{\Pr[A]}[\frac{N - f(x)}{NL}\Pr[e_s > e_c - 1] + \frac{f(x)}{N}\Pr[e_s > e_c + 1]]
\end{aligned}$$

[11] There 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.

The point where $E[\Delta_f \mid A] = 0$ is where RMHC will get stuck, i.e. the true fitness will not increase. The value of the true fitness at this point comes out to be

$$f(x) = N\frac{\Pr[e_s > e_c - 1]}{\Pr[e_s > e_c - 1] + L\Pr[e_s > e_c + 1]}$$

Hence, if RMHC is ever to find the optimal solution, the value of $L\Pr[e_s > e_c + 1]$ must become negligible in comparison to $\Pr[e_s > e_c - 1]$, so that $f(x)$ can come close to N. If the probability distribution of the noise is such that this never happens, then RMHC will not find the optimal solution.

Consider the case of uniform noise, i.e. e is uniformly distributed between $-E$ and E for some $E > 0$. When the current noise satisfies $e_c = E$, then clearly $L\Pr[e_s > e_c + 1] = 0$. in addition $e_c = E$ is the smallest point where $L\Pr[e_s > e_c + 1]$ is negligible in comparison to $\Pr[e_s > e_c - 1]$. This means that for RMHC to find the solution the noise value should remain at around E. This forces a slowdown of a factor of E since a string with a noise value of E will be sampled with probability $1/E$.

Next, consider the case of the Gaussian noise with deviation σ.

$$\begin{aligned}
\Pr[e_s > e_c + 1] &\approx \frac{1}{\sqrt{2\pi}\sigma}e^{-(e_c+1)^2/2\sigma^2} \\
&= \frac{1}{\sqrt{2\pi}\sigma}e^{-(e_c/2-1)/2\sigma^2}e^{-2e_c/\sigma^2} \approx \Pr[e_s > e_c - 1]e^{-2e_c/\sigma^2}
\end{aligned}$$

We see that for $L\Pr[e_s > e_c + 1] \ll \Pr[e_s > e_c - 1]$, we must have $L \ll e^{2e_c/\sigma^2}$. Hence, $e_c > \frac{1}{2}\sigma^2 \ln L$ which implies a slow down of at least

$$1/\Pr[e_s > e_c] \approx e^{2e_c/\sigma^2} > e^{\frac{1}{8}\sigma^2 \ln^2(L)}$$

Note that this value is only a lower bound on the slowdown, since it does not take into account the fluctuations in the value of the true fitness of the current favorite string.

An example where RMHC does not converge at all is the exponential distribution: $\Pr[X > k] = e^{-|k|}$. For this distribution we see that

$$\Pr[e_s > e_c + 1] = e^{-(e_c+1)} = e^{-(e_c-1)}e^{-2} = e^{-2}\Pr[e_s > e_c - 1]$$

implying that $L\Pr[e_s > e_c + 1]$ will never be negligible in comparison to $\Pr[e_s > e_c - 1]$, and indeed for this distribution RMHC does not converge to the optimal string.

Possibly some adaptation of RMHS using multiple strings and component by component voting or some such could be made to converge. We have not explored such possibilities, which don't seem particularly well motivated.

6.2 Culling and Noise

We now turn our attention to analyzing how GA2 performs in the presence of noise. The notation introduced in Section 5.1 will be used here as well. Let x be a string generated by breeding random strings from the current generation. Let A be the event that x is admitted into the next generation. Recall that a string will be accepted if its noisy fitness is larger than $\tilde{\mu}$, where $\tilde{\mu}$ is the mean $\tilde{f}$ of strings in the current population. Thus

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

The expected true fitness in the next generation will be

$$E[f(x) \mid A] = \frac{1}{\Pr[A]} \sum_{k=0}^{N} k \ \Pr[f(x) = k] \ \Pr[e \geq \tilde{\mu} - k]$$

Let σ be the standard deviation of the true fitness of a string bred from the current population and μ be the mean true fitness of strings in the current population. We can then rewrite the expected fitness of the next generation as:

$$E[f(x) \mid A] = \frac{1}{\Pr[A]} \sum_{k=-(N-\mu)}^{\mu} (\mu - k) \ \Pr[f(x) = \tilde{\mu} - \mu + k] \Pr[e \geq k] =$$

$$\mu + \frac{1}{\Pr[A]} \left[\sum_{k=1}^{(N-\mu)} k \ \Pr[f(x) = \mu + k] \ \Pr[e \geq \tilde{\mu} - \mu - k] - \sum_{k=0}^{\mu} k \ \Pr[f(x) = \mu - k] \ \Pr[e \geq \tilde{\mu} - \mu + k] \right]$$

If you remove the factor $\Pr[e \geq \tilde{\mu} - \mu - k]$ from the first sum, this sum is identically the expected gain in the absence of noise. The second sum accounts for the possibility that noise causes you to accept strings with lower than the mean true fitness.

Now consider first the domain where the standard deviation of the noise is much bigger than σ. Then the probability $\Pr[e \geq \tilde{\mu} - \mu + k]$ will not change much as k ranges a few σ. Thus the two sums will largely cancel. Strings are almost as likely to be accepted if they decrease the true fitness as if they increase it. Thus there will be very little gain in $E[f(x) \mid A]$ from generation to generation.

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

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

Hence, the gain in every generation is $O(p_t)$. Initially $p_t = 1/L$ that is much smaller than $\sqrt{N/L}$ that 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.

Conversely, consider the case where the standard deviation of the noise is much smaller than σ, e.g. $\sigma_e < \sigma/10$. In such a case almost all the variation in $\tilde{f}$ of new bred strings comes from the variation in their fitness, and the noise will have little impact on which strings are accepted. Hence our GA will suffer only a minor slowdown.

We assume for simplicity that the noise is symmetric around the origin so that $E[e] = 0$. (The reader is invited to explore generalizing this discussion.) Then because the distribution of noise and true fitness is symmetric, $P[A] = 1/2$. We can crudely lower bound the first sum by using $\Pr[e \geq \tilde{\mu} - \mu - k] \geq \Pr[e \geq \tilde{\mu} - \mu]$. $\tilde{\mu} - \mu$ is just the average noise of strings in the previous generation, so we are asking for the probability a new string has noise larger than the average in an accepted string. This can be lower bounded by 1/4 since, by symmetry, $Pr[e \geq 0] \geq 1/2$ and $Pr[e \geq \tilde{\mu} - \mu | e \geq 0] \geq 1/2$. Thus the first sum is crudely lower bounded by 1/4 of the gain in the absence of noise, and thus is greater than $\sigma/8$. The second sum can readily be upper bounded by $\Pr[f(x) = \mu] \sum_{k=0}^{\mu} k \ \Pr[e \geq k]$. But by using the asymptotic approximation for sums of Bernoulli random variables by the normal distribution (see e.g. [7]) as we did before in Theorem 8, one can show that $\Pr[f(x) = \mu] < 1/\sigma$. Using $E[e] = 0$ one can readily show that $\sum_{k=0}^{\mu} k \Pr[e \geq k] \leq \frac{1}{2}\sigma_e^2$. Hence the second sum is upper bounded by $\sigma_e^2/2\sigma$ which will be small compared to $\sigma/8$ so long as $\sigma_e << \sigma$, say $\sigma_e < \sigma/10$.

The initial σ is $\sqrt{N(1/L)(1-1/L)}$. If we have $\sigma_e < \sqrt{N(1/L)(1-1/L)}/10$, GA2 will converge almost as fast as if there were no noise until the strings are almost perfect. However, as $\mu \to N$, σ will decrease until it is comparable to σ_e. At that point the GA will slow down. We can modify the GA to avoid this problem. The simplest thing to do is to run the GA until $\mu \geq N/2$. At that point, we can truncate the genetic algorithm, and take a component by component vote of the members of the population. The correct component value will greatly exceed in frequency all the others.

The modified GA's robustness should be contrasted with other search algorithms whose performance degrades considerably under these noise models. In this noise regime, e.g. with $\sigma_e = \sqrt{N/L}/10$ the robust EPS 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.

7 Generalizations of ASP

Holland's 'implicit 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 'implicit 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 of the set $\{1, 2, ..., N\}$ of components into N/k groups each of size k. We are not told which components belong to which group. 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 5 to the k-ASP problem.

1. Set $M = \sqrt{N(L/2)^k/k}\ \log^2 N$.

2. Choose M random strings with fitness at least N/L^k. Call this set P.

3. Use M-way breeding to breed $2M$ new strings. If a string of fitness N has been generated, stop. Otherwise let P be the M most fit of these.

4. Go to step 3.

First observe that the expected fitness of a random string is N/L^k. Thus, step 2 of the algorithm will take an expected $O(M)$ time. The analysis of the above algorithm is similar to the one given in Section 5. We sketch here the intuition behind the result.

Let p_t be the fraction of correct components in the population at time t. We assume (for the purpose of this intuition sketch) that this is equal to the fraction of strings in the population having correct j-th component, that we assume constant independent of j, that will be essentially true assuming M is large enough that component to component fluctuations are small. The rigorous proof involves bounding deviations from component to component as in the ASP case. At generation $t+1$, we choose new strings from a population with each component correct with probability p_t. The mean number of groups such strings have correct is $\mu \equiv p_t^k N/k$ (i.e. their mean fitness is $p_t^k N$) and the deviation in number of correct groups is $\sqrt{(N/k)p_t^k(1-p_t^k)}$. By creating $2M$ strings and keeping only the top M, we will keep strings that can be seen on average to have more than $\sigma/4 \equiv (1/4)\sqrt{(N/k)p_t^k}$ more correct groups than the mean.

Now say we draw strings with each component right with probability p_t until we find one with exactly i correct groups. Let $\langle p|p_t, i\rangle$ be the expected fraction of correct components in such a string. Then

$$\langle p|p_t, i\rangle = \frac{ik}{N} + (1 - \frac{ik}{N})\frac{p_t - p_t^k}{1-p_t^k}$$

Here the first term on the right is the contribution from the i groups that are correct, and the second term is the fraction of components in incorrect groups times the probability that components in these incorrect groups are correct.

Now it is easy to see that

$$\langle p_{t+1}\rangle \geq \langle p|p_t, \mu + \sigma/4\rangle \equiv \langle p|p_t, p_t^k N/k + (1/4)\sqrt{N/kp_t^k}\rangle$$

Thus[12]

$$p_{t+1} \geq p_t^k + (1/4)\sqrt{kp_t^k/N} + (1 - p_t^k - (1/4)\sqrt{kp_t^k/N})\frac{p_t - p_t^k}{1-p_t^k}$$

Expanding the right hand side in a power series in p_t, one finds

$$p_{t+1} \geq p_t + (1/4)\sqrt{kp_t^k/N}(1-p_t)$$

Now the analysis is similar to the ASP case. If we have a gain of about $(1/4)\sqrt{kp_t^k/N}$ in p_t at each step, the time for p_t to double can be seen to be about

$$T \sim \sqrt{\frac{N}{k}(\frac{1}{2p_t})^k}$$

The total number of generations needed is of order the initial doubling time, i.e. $O(\sqrt{\frac{N}{k}(\frac{L}{2})^k})$. To achieve small divergence in the number of strings correct at differing components, we need to choose M large enough so the mean number of correct i-th components in the population Mp_T is much greater than the deviation $\sqrt{MTp_T}$ for which $M = \sqrt{N(L/2)^k/k}\ \log^2 N$ suffices. As in § 5.1 we state the running time of the algorithm implied by the above heuristic argument as a conjecture.

Conjecture 11 *Let $k \geq 3$. The above algorithm will solve the k-ASP problem with constant probability in $O(\sqrt{N(L/2)^k/k})$ generations using $\tilde{O}(N(L/2)^k/k)$ queries.*

To obtain a provable result one can weaken the conjecture using an approach similar to the one taken in § 5.2. Notice that unlike the algorithm in Section 5, the size of the population M was fixed throughout the algorithm. This was done for the sake of simplicity. The algorithm can be slightly improved by letting the population size shrink as the algorithm progresses.

[12] Here we drop the $\langle\rangle$ on p_{t+1} that is now understood to be the expected number of correct components at time $t+1$.

7.1 Explicitly and Implicitly Parallel Search

If you knew which components belonged to which groups, you could readily map the k-ASP problem into a simple ASP problem with $L' = L^k$. That is to say, you could regard the L^k different possible k-tuple value belonging to the components in a given group each as the value of a single component taking on of L' possible values.

If a GA were able to search over all schemata of size k in parallel, as the intuitive picture of the Schema Theorem hopes, then the GA would automatically treat each k-group as one single component taking $L' = L^k$ different values, effectively reducing the k-ASP problem to an ASP problem with L' instead of L. But since GA2 solves the ASP problem in a number of generations depending only logarithmically on L, (cf Theorem 7) it would then solve the k-ASP problem in a number of generations depending only linearly on k, not exponentially. Moreover we have conjectured (cf Conjecture 2) that the GA uses a number of queries depending only logarithmically on L, so that it should solve the k-ASP problem in a number of queries depending only linearly on k, not exponentially. In fact, however, GA's that we are able to analyze suffer an exponential slowdown in both generations and queries. There are various types of crossover operation one might propose in an effort to solve this problem. We have searched for an effective one. We have experimentally studied the main candidates we are unable to analyze and find similar results. Thus GA's do not appear to achieve the 'implicit parallelism' hoped for by Holland's schema theorem on the k-ASP problem.

This is perhaps not surprising. In any one-point crossover approximately a fraction $(2^k - 1)/2^k$ of the order k schemata would be destroyed, i.e. roughly speaking one could only generate schemata with local support, all in the first half of the string or all in the second half. Uniform crossover is no improvement. It destroys the same fraction of schemata on any given generation, but changes which schemata survive from generation to generation randomly. The GA literature typically emphasizes that a large number of schemata remain, which is true. But it is far from clear how the benefits compare to the drawbacks, especially relative to other methods such as local search. This situation is even murkier in more realistic problems with local minima.

However, perhaps surprisingly, we will see that all the gains one could hope for from 'Intrinsic Parallelism' are available on the k-ASP problem, by the approach of Explicitly Parallel Search mentioned in §4. To see this modify the EPS algorithm so that, 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. By changing L^k sites one expects to change the fitness, analogously to the case for plain ASP. Binary search will now uncover one component[13] that individually suffices to change the fitness value. Analogously to the ASP case, we need only an amortized $k \log L$ queries per correct component we discover (and mark known). Thus we can efficiently solve k-ASP with only $O(Nk \log L)$ queries.

Theorem 12 *EPS can solve k-ASP efficiently with $O(Nk \log L)$ queries.*

[13]This seems counterintuitive, so we elaborate. One has two strings A and B differing on some components and differing in fitness. Change half the differing components of B to agree with A and call this string C. Now C can not have the same fitness as both A and B. Say WLOG C differs from A in fitness. Iterating will eventually produce two strings, one with the fitness of A, that differ in one component and have different fitness. We can now isolate a correct component and mark it known.

8 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[16, 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.
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

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

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 that 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] shows that

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

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

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

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 (which will be a close approximation). Since a sum of independent 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

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

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 §6 we observed a similar behavior for the GA. However the MFA algorithm, because it is oblivious, can be defeated by problems with less symmetry that the GA can solve.

9 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 [28, 29, 9, 5]. Let $t \in \{1, -1\}^N$ be an N-dimensional vector and let H_t be the halfspace $(x, t) \geq 0$. We are given a number of samples $x_1, \ldots x_s$ 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 [9] 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. [29] gave an algorithm that 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

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

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 $x_1, \ldots, x_s$ 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 query its oracle for (a noisy estimate of) 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 that 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 culling variant described in §3 where instead of breeding until we had M strings of fitness at least μ, we bred $6M$ strings and kept the M strings with the largest fitness. In cases where the GA is effective, either method produces a new population in $O(M)$ steps with fitness increased by $O(\sigma N)$ but 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.

10 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, that is the main intuition driving this hope. We exhibited an example, that we call ASP, where the Schema Theorem is fully satisfied, and yet the GA based on it is very inefficient. We suggested a different breeding approach called Culling that on general grounds should frequently be more effective than standard GA's. We showed that Culling is effective on ASP, conjecturally near optimal, 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 also calculated the optimal culling point for selective breeding in a simple model. This culling point was universal in the sense that it is independent of the fitness function and the distribution of fitnesses of offspring.

We also proposed and studied a problem we call k-ASP with a more interesting schema structure. We argued that naive notions of "implicit parallelism" would lead one to hope for a slowdown linear in k on this problem. All the GA's we studied, however, suffered a slowdown exponential in k, perhaps because, although they explore exponentially large numbers of schemata in parallel, each crossover destroys a *fraction* of the potentially relevant schemata that is exponentially close to unity. Perhaps surprisingly, we were able to describe an algorithm, that we call Explicit Parallel Search, that did solve this problem efficiently, having slowdown linear in k.

We also studied an algorithm we call MFA that performs similarly to Culling on noisy ASP and k-ASP, but because of its oblivious nature would fare worse on some less symmetric problems. Finally we noted that the problem of learning the Ising perceptron maps into a noisy ASP problem, suggesting some solution techniques.

The running time of our genetic algorithm was stated in Conjecture 2. We then gave a proof of the weaker running time estimate given in Theorem 7. An interesting open problem is to prove Conjecture 2. This would require strengthening our large deviation bound given in Lemma 9 to handle the full sub-martingale which models the behavior of algorithm GA2. Such a result will be of great interest to us.

Another interesting open problem is to extend these analyses to other, more complex fitness functions. Are there practically important problems on which Culling or EPS approaches can beat competing algorithms?

Acknowledgments

The authors would like to thank W. Finoff, L. Gurvits, M. Mitchell, S. Rao, J. Reif, and W. D. Smith for helpful conversations, and Harold Stone and the anonymous referees for comments on a draft.

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, D. Boneh, C. Garrett, "On Genetic Algorithms", COLT '95: Proceedings of the Eighth Annual Conference on Computational Learning Theory, Association for Computing Machinery, New York, (1995) pp 230-239.

[5] E. B. Baum, Y. Lyuu, "The Transition to Perfect Generalization in Perceptrons", Neural computation 3, pp. 386-401.

[6] V. Chvatal, "MasterMind", Combinatorica 3 (1983), pp. 325-329.

[7] P. Deheuvels, M. Puri, S. Ralescu, "Asymptotic Expansions for Sums of Nonidentically Distributed Bernoulli Random Variables", J. of Multivariate Analysis, Vol. 28, pp. 282–303, 1989.

[8] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading MA, 1989.

[9] D. Haussler, M. Kearns, S. Seung, N Tishby, "Rigorous Learning Curve Bounds from Statistical Mechanics", Proceedings of COLT 1994, pp. 76-85.

[10] D. Hillis: "Co-evolving parasites improve simulated evolution as an optimization procedure." Physica D 42 (1990) 228-234.

[11] W. Hoeffding: Probability inequalities for sums of bounded variables. J. American Statistical Association, 58:13-30, 1963.

[12] 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).

[13] J. H. Holland: *Adaptation in natural and artificial systems*, MIT Press, Cambridge MA, (1975).

[14] M. Kimura, J. Crow, *Effect of overall phenotypic selection on genetic change at individual loci*, Proc. Natl. Acad. of Sci., USA, Vol. 75, No. 12, pp. 6168–6171, December 1978.

[15] M. Kimura, J. Crow, *Efficiency of truncation selection*, Proc. Natl. Acad. of Sci., USA, Vol. 76, No. 1, pp. 396–399, January, 1979.

[16] S-K Ma: *Modern Theory of Critical Phenomena*,W. A Benjamin, Reading MA 1976.

[17] 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.

[18] H. Muhlenbein and D. Schlierkamp-Voosen: Predictive models for the breeder genetic algorithm I Continuous parameter optimization, Evolutionary Computation 1, pp 25-50 1993.

[19] Y. Rabinovich: Quadratic dynamical systems and theoretical aspects of genetic algorithm, Ph.D. Thesis, Hebrew university of Jerusalem, 1993.

[20] I. Rechenberg: "Cybernetic solution path of an experimental problem", ROy. Aircr. Establ., libr. transl. 1122. Hants, UK.: Farnborough (1965)

[21] I. Rechenberg: "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution", Stuttgart: Frommann-Holzboog (1973).

[22] A. Renyi: Collected works, pp. 633-638.

[23] Y. Rabinovich, A. Sinclair, A. Wigderson: Quadratic dynamical systems, Proceedings of the 33rd IEEE FOCS (1992), pp. 304-313.

[24] J. P. Ros: Learning Boolean functions with genetic algorithms: a PAC analysis, Foundations of Genetic Algorithms 2, ed. L. Darrell Whitley, pp257-275 (1993).

[25] C. E. Shannon (1951) Prediction and entropy of printed English, Bell System Technical J. pp50-64, January.

[26] H.-P. Schwefel. *Numerical optimization of computer models.* Chichester: Wiley.

[27] G. Syswerda: Simulated crossover in Genetic Algorithms, Foundations of Genetic Algorithms 2, ed. L. Darrell Whitley, pp239-255 (1993).

[28] S. Venkatesh: On learning binary weights for majority functions, proceedings of COLT 1991.

[29] 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.

Appendix A: Mitchell, Holland, and Forrest's Royal Road Function

Our function F_1 is closely related to a fitness function studied by Mitchell, Holland, and Forest [17] and called by them the Royal Road function R_1. They used an alphabet of only 2 alleles, 0 and 1, and their fitness function was

$$R_1(x) = \sum_i \delta_i(x) \text{ where } \delta_i(x) = \left\{ \begin{smallmatrix} 1 & \text{if} x \in s_i \\ 0 & \text{otherwise} \end{smallmatrix} \right.$$

where s_i is the set $\{ik, ik+1, ik+2, ..., (i+1)k-1\}$ for k a chosen integer. (They used $k = 8$).

To translate into our F_1, if $2^k = L$ one may replace the $ik, ik+1, ..., (i+1)k-1$ bits of their strings with one component taking L values. Then it is immediate that a binary string of fitness f under R_1 translates into an L-ary string of fitness f under F_1 and vice versa.

[17] found experimentally (to their surprise) that a GA like GA* did not perform as well as RMHC and offered various explanations. We believe the most plausible explanation, however, is the variance analysis presented in section 4.1. Mitchell et al then discussed an 'Idealized Genetic Algorithm' (IGA) that cheated, but did outperform RMHC. The idea of this was to understand what features of this IGA allowed it to outperform Hill Climbing, and to try to find problems or improvements that would allow a GA to win without cheating. The lower bound analysis of section 4, however, shows that not much can be gleaned from the IGA in terms of insight applicable in practice. IGA actually outperforms the lower bound, so it is apparent that its gains are entirely due to cheating, and can not be realized in any practical problem or method.

Appendix B: Coevolving Parasites

A recent paper of D. Hillis [10] suggested a mechanism modelled on the coevolution of parasites and hosts for speeding evolution. The problem he was addressing was evolving a sorting network on 16 inputs with as few gates as possible. Since there are 2^{16} possible input strings, he didn't want to test on each one at every generation. So to speed up, he chose a subset of 1024 strings to test on. Based on this he would keep the fittest networks, do crossover, and iterate. This would learn a sorting network on that particular set of 1024 strings rapidly, so he would change the set of 1024 strings randomly. This method learned sorters using 65 gates.

Then he tried a different idea: 'co-evolving parasites'. The idea here was to simultaneously evolve the test set and the sorting networks. The sorting networks were 'fit' if they did well on the test set, and the test strings were 'fit' if they fooled sorting networks. Using this method he was able to produce a sorting network with 61 gates.

We wish to remark, however, that the 'co-evolution' here may be very problem specific. Hillis was giving his strings a head start by initializing the first 32 gates to be the Butterfly. Now, by the time you've sent all 2^{16} strings through the Butterfly, there are only 168 equivalence classes. So presumably the co-evolution was simply selecting (a somewhat redundant) set of examples from these 168 classes.

Warren D. Smith has used a "randomized greedy" (non genetic) approach to design sorting networks. His runs produce 60 gate sorters regularly.