On the Convergence of an Estimation of Distribution Algorithm Based on Linkage Discovery and Factorization

Alden H. Wright
Computer Science
University of Montana
Missoula, Montana 59812 USA
alden.wright@umontana.edu

S.V.P.M.Sandeep Pulavarty
Computer Science
University of Montana
Missoula, Montana 59812 USA
sandeep_pulavarty@yahoo.com

November 23, 2005

Abstract

Estimation of distribution algorithms construct an explicit model of the problem to be solved, and then use this model to guide the search for good solutions. For an important class of fitness functions, namely those with k-bounded epistasis, it is possible to construct a complete explicit representation of the fitness function by sampling the fitness function. A very natural model of the problem to be solved is the Boltzmann distribution of the fitness function, which is an exponential of the fitness normalized to a probability distribution. As the exponentiation factor (inverse temperature) of the Boltzmann distribution is increased, probability is increasingly concentrated on the set of optimal points. We show that for fitness functions of k-bounded epistasis that satisfy an additional property called the running intersection property, an explicit computable exact factorization of the Boltzmann distribution with an arbitrary exponentiation factor can be constructed. This factorization allows the Boltzmann distribution to be efficiently sampled, which leads to an algorithm which finds the optimum with high probability.

1 Introduction

An **additively decomposable fitness function** (ADF) over bit strings is one which can be written as a sum of simpler subfunctions each of which depends only on a small number of bits. The **support set** of a subfunction is the set of bits (or bit positions) that it depends on. An ADF is called k**-epistatic** if the maximum size of the support sets of the subfunctions is k. Heckendorn and Wright [6, 7] have described a randomized algorithm that can find the structure of a black box k-epistatic ADF with any given success probability. In other words, with only the prior knowledge that the fitness is k-epistatic, the algorithm can determine an ADF decomposition of the function by sampling the fitness function. This paper explores one way in which this linkage detection algorithm can be extended to an estimation of distribution algorithm.

The Boltzmann probability distribution can be described for any fitness function $f(x)$ [19]:

$$p(x) = \frac{e^{uf(x)}}{\sum_y e^{uf(y)}} \tag{1}$$

We will call the parameter u the **exponentiation factor**, but traditionally it has been interpreted as the inverse temperature. As u tends to infinity, the probability of the set of optimal points goes to 1. Thus, if one could sample from the Boltzmann distribution for sufficiently large u, one would have a high probability of finding the optimum point. However, this is not feasible in general because computing the denominator in (1) would require summing over all points in the search space.

Mühlenbein, Mahnig, and Rodriguez [19] prove a factorization theorem that gives a condition (called the running intersection property) on the collection of support sets of an additively decomposable fitness function that allow the Boltzmann distribution of an ADF to be factorized. Once the Boltzmann distribution is factorized, it can be efficiently sampled.

Our algorithm consists of three phases. The first phase is to apply the Linkage Detection Algorithm of Heckendorn and Wright [6, 7] to determine the ADF structure of the fitness function. The second phase is to compute an exact (up to roundoff error) factorization of the Boltzmann distribution of the fitness function. The third phase is to sample the Boltzmann distribution. If the exponentiation factor of the Boltzmann distribution is sufficiently high, then there will be a high probability of finding the optimum on each sample.

We describe our algorithm in more detail. We assume that the search space is the set of binary strings of length ℓ. If the fitness function is known to be k-**epistatic**, then the Linkage Detection algorithm of [7] can be applied. Given a success probability $\delta < 1$, it will find an ADF structure for the fitness function using $O(\ell^2 \log \ell)$ (assuming that k and δ are fixed) function evaluations with probability at least δ. If the ADF structure has the running intersection property, then the factorization theorem can be applied to produce a factorization of the Boltzmann distribution. The computation of the factorization is $O(2^k \ell)$. Then an exponentiation factor must be chosen. The exponentiation factor necessary to obtain a given probability of the optimum will depend on the fitness separation between the optimum and the suboptimum. In practice, the exponentiation factor can be increased until a high probability of the optimum is obtained by sampling. The sampling phase is $O(\ell)$.

Our algorithm has the advantage that the subfunctions of the ADF may overlap and may have different scales. Estimation of distribution algorithms based on sampling a population may have difficulties with ADF fitness functions where the subfunctions have varying scales.

Our algorithm naturally finds multiple optima. When the Boltzmann distribution is sampled, all optimal points have equal probabilities of being sampled. Further, the number of optima can be estimated from the probability of any single optimum. Thus, the algorithm may have applications in multiple objective optimization.

We describe an implementation of this algorithm. If the exponentiation factor u in the Boltzmann distribution is large, then there are potential problems with floating overflow and underflow. Our implementation deals with these problems and allows the exponentiation factor to be very large. We describe a test problem which is what we conjecture to be a worst-case for the algorithm. We give test results on this function and other fitness functions.

2 Background

Genetic algorithms have been found to not be very effective at solving optimization problems where there is a large amount of interaction between variables [26].

One approach to solving this problem has been based on manipulating the representation of solutions in the algorithm to make the interacting components of partial solutions less likely to be disrupted by recombination operators. For example, various reordering operators have been tried, but these operators were too slow to work in a genetic algorithms optimization framework where quick convergence is expected. There has been considerable work on algorithms that evolve the representation or ordering of a problem. This work includes the messy GA [3], the gene expression messy GA [11], and the linkage learning genetic algorithm (LLGA) [4].

This also led to research on methods to detect the nonlinear interactions between variables independent of how this information would be used. Munetomo and Goldberg [21] pioneered the use of perturbational methods for this purpose. These methods do perturbations of solutions designed to test for a specific interaction between variables. Munetomo and Goldberg developed perturbational methods for detecting pairwise interactions. Heckendorn and Wright [7] extended their results to deal with higher order interactions, and extended results of Kargupta [12] to give an algorithm that discovers the structure of an ADF. They show that this can be done for a constant success probability and a constant degree of epistasis using $O(\ell^2 \log \ell)$ function evaluations (assuming k and the success probability are constant). Streeter [25] showed that if the ADF subfunctions are non-overlapping, then pairwise interactions can be discovered using $O(\ell \log \ell)$ function evaluations.

Estimation of distribution algorithms [20] have developed into another way to deal with interactions between variables. The general principle is to use a set of solutions to estimate the fitness distribution, and then to generate new candidate solutions using this estimated distribution.

The first estimation of distribution algorithms did not try to estimate interactions between variables. Instead, they generated new solutions by only preserving the proportions of values of variables in the current solution. These algorithms included PBIL [1], UMDA [20], and the compact genetic algorithm (cGA) [5]. These algorithms work well on problems without significant interactions between variables. Some algorithms that were based on pairwise interactions include MIMIC of de Bonet et al [2] and BMDA of Pelikan and Mülenbein [24].

However, dealing with only pairwise interactions is insufficient to solve problems with higher order interactions efficiently.

An important EDA that deals with higher order interactions is the Bayesian optimization algorithm (BOA) of Pelikan et al [22, 23]. BOA uses Bayesian networks to represent the distribution and uses the Bayesian Dirichlet metric to measure the quality of solutions.

The factorized distribution algorithm (FDA) [15, 16, 19, 14] is another EDA that uses approximations of higher-order interactions. The FDA uses a factorization of the Boltzmann distribution for the generation of new solutions. The factorization theorem that is used in this paper is proved in [19]. The FDA requires prior information about the problem in the form of a problem decomposition (i. e., the support sets of an ADF decomposition of the fitness function).

Zhang [27] has developed infinite population models for the UMDA and the FDA and has shown that there are uniform populations which are stable fixed points for the UMDA which are not stable fixed points for the FDA. This suggests that by using higher-order information about the fitness function, the FDA is better able to escape being trapped in local optima.

Zhang and Mühlenbein [28] have shown that an infinite population model of the FDA with real variables will converge to the optimum for an additively decomposable fitness function with the running intersection property.

An algorithm that learns a factorization, the LFDA, is described but only minimally developed. A hybrid algorithm between the FDA and the LFDA is applied to graph bipartitioning problems in [17].

Mühlenbein and Mahnig [18, 17] point out how sampling according to the Boltzmann distribution fulfills a conjecture by Holland [8] on a good population-based search algorithm. In the following, $P(\xi, t)$ is the probability of schema ξ at time t.

Conjecture 1 *Each (schema) ξ represented in the current population $B(t)$ should increase (or decrease) in a rate proportional to its "observed" "usefulness" $\hat{\mu}_\xi(t) - \hat{\mu}(t)$ (average fitness of schema ξ minus average fitness of the population)*

$$\frac{dP(\xi,t)}{dt} = (\hat{\mu}_\xi(t) - \hat{\mu}(t))P(\xi,t) \tag{2}$$

This is Mühlenbein and Mahnig's [18] restatement of Holland's conjecture ([8], page 88). Mühlenbein and Mahnig point out that the simple genetic algorithm does not behave according to (2), but that sampling according to the Boltzmann distribution does satisfy (2).

3 The Linkage Detection Factorization Algorithm

Our Linkage Detection Factorization Algorithm (LDFA) is a combination of the linkage detection algorithm of [7] and an algorithm based on the factorization theorem. There are three phases:

Linkage Detection Factorization Algorithm:

1 Run the linkage detection algorithm to determine the ADF structure of the fitness
2 Use the factorization theorem to compute the factorization of the Boltzmann distribution of the fitness.
3 Sample the Boltzmann distribution r times.

3.1 The linkage detection algorithm

The linkage detection algorithm is a randomized algorithm that determines a subfunction decomposition of a k-epistatic fitness function over fixed length binary strings given as a black box. In other words, the only prior knowledge of the fitness function is that the maximum degree of interaction between bits is of degree k. Given a desired probability $0 < \delta < 1$, the algorithm determines an additive decomposition for the function with probability at least δ. The algorithm is based on perturbation: the algorithm does specific perturbations of a string to test for interaction between bits. To test for interaction between two bits, one starts with a background string c, and then flips each bit individually and both bits together. If the sum of the effects of flipping each bit individually, is different than the effect of flipping both bits, then interaction between the two bits has been discovered. In [6, 7] this kind of a test is called a **probe**. A probe of order j tests for interaction among a set of j bits and requires 2^j fitness evaluations. If $j < k$, multiple probes (with multiple background strings) must be done to reliably test for interaction among a given set of j bits.

The linkage detection algorithm not only detects which subsets of bits interact in the fitness function, it also constructs an ADF decomposition for the fitness. We give a brief overview of how this is done. A set of bits is called **epistatic** if the bits interact relative to the fitness function. If a set of bits is epistatic and no superset is epistatic, then the result of any probe on that set of bits is exactly the Walsh coefficient of the fitness function whose index is the mask for the set of bits. Once this Walsh coefficient has been determined, then Walsh coefficients for masks of subsets can be determined by using probes on these subsets using the same background string as was used to determine the Walsh coefficient of the maximal epistatic set. When the Walsh coefficients of an ADF subfunction have been determined, then the inverse Walsh transform can be used to obtain the standard basis description of that subfunction. The result is a complete description of an ADF decomposition of the fitness function.

One advantage of use of perturbation methods to determine the structure of the fitness function is that perturbation methods are insensitive to the relative scaling of the ADF subfunctions. Since the perturbation (or probe) is specifically designed to test for the interaction of a set of bits, noise from subfunctions not involved with those bits is not a problem. A second advantage is that there is a rigorous complexity analysis of the number of function evaluations needed [7]. If number of subfunctions grows linearly with the string length, and if a global success probability to find all subfunctions is wanted, then the number of function evaluations is $O(\ell^2 \log \ell)$ where ℓ is the string length. The number of fitness evaluations grows exponentially with the degree k of epistasis. The number of fitness evaluations can be reduced in practice by using a population and by caching fitness evaluations, however the correctness of this technique has not been proved.

If the support sets of the subfunctions of the ADF are nonoverlapping, then the algorithm of Streeter [25] can find the pairwise dependencies in $O(\ell \log \ell)$ time. It is clear that Streeter's algorithm can be combined with the Linkage Detection Algorithm of [7] to find the ADF structure of an ADF fitness function whose subfunctions do not overlap using $O(\ell \log \ell)$ function evaluations. We have not implemented this.

The worst case for the linkage detection algorithm is when the subfunctions are either needle-in-the-haystack functions or are linear except for one special value. As the subfunctions deviate from these cases, fewer probes are needed. For example, if the subfunctions were random, then mathematically only one probe would be needed to test for the interaction of a set of bits. (Roundoff error problems would mean that a small number of probes should be used.) Concepts such as deceptiveness are irrelevant. Also, due to roundoff error, a tolerance must be set in the determination of whether a probe result is nonzero, and so if the scaling of subfunctions was too extreme, there could problems due to

the finite precision of floating-point arithmetic.

3.2 Computing the factorization

The factorization part of our algorithm assumes that the ADF structure of the fitness function is known from the linkage detection phase of the algorithm. The proof of factorization theorem of [19] shows how to write the Boltzmann distribution of the fitness as a product of conditional probabilities and shows how to efficiently compute the required conditional probability tables.

We first introduce some notation. Let $\mathbf{X}$ be the set of bit strings of length ℓ. We use binary operators $\wedge$ and $\oplus$ where $\wedge$ denotes bitwise AND, and $\oplus$ denotes EXCLUSIVE-OR. Let $\mathcal{L} = \{0, 1, \ldots, \ell - 1\}$ be the set of bit positions. We will often denote a subset s of $\mathcal{L}$ by its mask: the mask is a binary string m such that $m_i = 1$ for $i \in s$ and $m_i = 0$ for $i \notin s$. Sometimes we will use the same symbol for a set and its mask. Let $\mathbf{X}_m = \{x \in \mathbf{X} \ : \ x \wedge m = x\}$. In other words $\mathbf{X}_m$ is the set of length ℓ bit strings that may have 1s only in the positions masked by m.

If $s \subset \mathcal{L}$, let π_s be the projection mapping that takes a bit string onto the substring corresponding to the bit positions in s. For example, if $\ell = 5$ and $s = \{1, 3\}$, $\pi_s x = \pi_s(x_0 x_1 x_2 x_3 x_4) = x_1 x_3$. Let $x_s = x \wedge s$ (where s is interpreted as a mask). Thus, $\pi_s x$ is a string of length $|s|$ while x_s is a string of length ℓ. Furthermore, $\pi_s x = \pi_s x_s$.

If $f = \sum_{i=1}^{n} f_i(\pi_{s_i} x)$ is an ADF, let

$$e_i(x) = e^{u f_i(\pi_{s_i} x)}$$

If $p(x)$ is the Boltzmann probability distribution defined by (1), then

$$p(x) = \frac{\prod_{i=1}^{n} e_i(x)}{\sum_y \prod_{i=1}^{n} e_i(y)} \tag{3}$$

In other words, the Boltzmann distribution of an ADF is multiplicatively decomposable.

Let $p(\pi_{s_i} x)$ denote marginal probability. In other words,

$$p(\pi_{s_i} x) = \sum_{y \in \mathbf{X}_{\mathcal{L} \setminus s_i}} p(x_{s_i} \oplus y) \tag{4}$$

where $x_{s_i} \in \mathbf{X}_{s_i}$ such that $\pi_{s_i} x = \pi_{s_i} x_{s_i}$. Under conditions given below, the Boltzmann distribution $p(x)$ of an ADF can be *factorized* so that $p(x)$ can be computed efficiently from sub-functions of the ADF [19]. Given a fitness function f that can be written as an ADF with sub-functions f_i's whose support masks are s_i's, then we compute new sets d_i,b_i and c_i as

$$d_i := \bigcup_{j=1}^{i} s_j \tag{5}$$

$$b_i := s_i \setminus d_{i-1} \tag{6}$$

$$c_i := s_i \cap d_{i-1} \tag{7}$$

We set $d_0 := \emptyset$. If $j < i$ and $c_i \subseteq s_j$, i is called the **successor** of s_j.

Theorem 2 (Factorization Theorem [19]) *Let*

$$f = \sum_{i=1}^{n} f_i(\pi_{s_i} x)$$

be an ADF and $p(x)$ be its Boltzmann distribution. If

$$b_i \neq 0 \quad \forall\, i = 1, \ldots, n; \quad d_n = \mathcal{L} \tag{8}$$

$$\forall\, i \geqslant 2 \;\; \exists j < i \text{ such that } c_i \subseteq s_j \tag{9}$$

Then

$$p(x) = \prod_{i=1}^{n} p(\pi_{b_i} x | \pi_{c_i} x) \tag{10}$$

where $p(\pi_{b_i} x | \pi_{c_i} x)$ is the probability of $\pi_{b_i} x$ given $\pi_{c_i} x$.

Conditions (8) and (9) defi ne the **running intersection property**.

Intuitively, the running intersection property says that the support sets have a tree-like structure determined by the successor function.

It follows from [19] that if f can be written as an ADF with running intersection property and $\mathcal{S}(i)$ is the successor set of s_i, then for any $x \in \mathbf{X}$

$$p(\pi_{b_i} x | \pi_{c_i} x) = \frac{e_i(x_{b_i} \oplus x_{c_i}) \prod_{j \in \mathcal{S}(i)} \sum_{z \in \mathbf{X}_{b_j}} e_j(z \oplus x_{b_i})}{\sum_{w \in \mathbf{X}_{b_i}} e_i(w \oplus x_{c_i}) \prod_{j \in \mathcal{S}(i)} \sum_{z \in \mathbf{X}_{b_j}} e_j(z \oplus w)} \tag{11}$$

Example:

Consider a fi tness function f which can be written in the form of an ADF as

$$f(x_0 x_1 x_2 x_3) = f_1(x_0 x_1) + f_2(x_1 x_2) + f_3(x_2 x_3)$$

$x_i x_j$	$f_1(x_0 x_1)$	$f_2(x_1 x_2)$	$f_3(x_2 x_3)$	e^{uf_1}	e^{uf_2}	e^{uf_3}
00	2.36	0.73	1.49	10.6	2.08	4.44
01	0.69	0.14	0.94	1.99	1.15	1.15
10	0.95	0.27	0.14	2.59	1.31	2.56
11	1.64	0.41	1.08	5.16	1.51	2.94

Table 1: Fitness Values.

Table 1 represents the fi tness values of each sub-function f_i (given the values of $\pi_{s_i} x$) and e^{uf_i} values with $u = 1$.

For example, the probability of selecting an individual $x = 0110$ can be computed using equation (3) as

x_2	$p(x_3 = 0\|x_2)$	$p(x_3 = 1\|x_2)$
0	0.79	0.21
1	0.47	0.53

Table 2: Distribution for $p(x_3|x_2)$.

x_1	$p(x_2 = 0\|x_1)$	$p(x_2 = 1\|x_1)$
0	0.65	0.35
1	0.47	0.53

Table 3: Distribution for $p(x_2|x_1)$.

$$p(0110) = \frac{e^{uf_1(01)} * e^{uf_2(11)} * e^{uf_3(10)}}{F_u} = \frac{1.99 * 1.51 * 2.56}{347.84} = 0.02 \quad (12)$$

The probability of an individual can also be computed using the factorization of the probability. The first step is to construct the required sets as shown below:

We set $d_0 = \emptyset$, and the other d_i's, b_i's and c_i's are computed using equations (5), (6) and (7) as

$$\begin{array}{lll} d_1 = \{0,1\} & b_1 = \{0,1\} & c_1 = \{\} \\ d_2 = \{0,1,2\} & b_2 = \{2\} & c_2 = \{1\} \\ d_3 = \{0,1,2,3\} & b_3 = \{3\} & c_3 = \{2\} \end{array}$$

As the sets b_1, b_2 and b_3 are not empty and $c_2 \subseteq s_1$, $c_3 \subseteq s_2$, running intersection property is not violated. So the factorization theorem holds. It is easy to see that the successor of s_1 is 2, and the successor of s_2 is 3, and there are no successors for s_3. The conditional probability tables can be computed using (11).

For example,

$$p(x_3 = 0|x_2 = 1) = \frac{e_3(0000 \oplus 0010)}{e_3(0000 \oplus 0010) + e_3(0001 \oplus 0010)} = 0.47$$

and

$$p(x_2 = 1|x_1 = 1) = \frac{e_2(0010 \oplus 0100) * \sum_{z \in \mathbf{X}_{b_3}} e_3(z \oplus 0010)}{\sum_{w \in \mathbf{X}_{b_2}} e_2(w \oplus 0100) * \sum_{z \in \mathbf{X}_{b_3}} e_3(z \oplus w)} = 0.53$$

x_0x_1	$p(x_0x_1)$
00	0.55
01	0.09
10	0.13
11	0.23

Table 4: Probability Distribution $p(x_0x_1)$ for set 1.

The probability distributions $p(x_3|x_2), p(x_2|x_1)$ and $p(x_0x_1)$ are tabulated in tables 2,3 and 4 respectively.

From equation (10), the probability of selecting an individual $x = x_0x_1x_2x_3$ is given by

$$p(x) = p(x_0x_1)p(x_2|x_1)p(x_3|x_2) \tag{13}$$

For Example,

$$\begin{aligned} &p(0110) \\ &= p(x_0 = 0, x_1 = 1)p(x_2 = 1|x_1 = 1)p(x_3 = 0|x_2 = 1) \\ &= 0.09 * 0.53 * 0.47 = 0.02 \end{aligned}$$

This value matches with the value that is directly computed in equation (12).

3.3 Choosing the exponentiation factor

The Boltzmann distribution depends on a parameter that we call the **exponentiation factor**. This parameter is also called the **inverse temperature**. As the exponentiation factor goes to infinity, the probability of the set of optimal points goes to 1. Thus, our algorithm consists of setting the exponentiation factor to be suitably large, and then sampling from the Boltzmann distribution. This raises two questions: (1) how large does the exponentiation factor need to be? (2) how does one avoid floating overflow and underflow problems in computing with high exponentiation factors?. We discuss each of these questions.

Our objective in setting the exponentiation factor is to make sure that the probability of the set of optimal points in the Boltzmann distribution is at least C times the probability the set of less than optimal points.

Suppose that the fitness of an optimal point is $1 + \epsilon$ times the fitness of the next best point and suppose that there are K times as many suboptimum points as optimum points. If we want the probability of the set of optimum points to be at least C times the probability of the set of less than optimal points, then we want to choose the exponentiation factor to satisfy the following inequality:

$$\begin{aligned} & e^{u(1+\epsilon)} \geq CKe^u \\ \iff & e^{u\epsilon} \geq CK \\ \iff & u \geq \frac{\ln C + \ln K}{\epsilon} \end{aligned}$$

Clearly, K is bounded above by $2^\ell - 1$. To summarize:

Lemma 3 *If the fitness of the optimum is $1+\epsilon$ times the fitness of the suboptimum, and if the power-of-two exponentiation factor u is chosen so that*

$$u \geq \frac{\ln C + \ell \ \ln 2}{\epsilon}$$

then the probability of the set of optimum points in (1) will be at least C times the probability of the set of suboptimum points.

The only way to have $K = 2^\ell - 1$ is to have a "needle-in-the-haystack" fitness landscape which is not an ADF with $k < \ell$. We conjecture that K is bounded above by $2^k \ell$. Under this conjecture, it is sufficient to choose u so that

$$u \geq \frac{\ln C + k \ \ln 2 + \ln \ell}{\epsilon} \tag{14}$$

In section 5, we give an example of a fitness function that comes close to achieving this conjectured upper bound.

In practice, there might be some a priori knowledge of ϵ. For example, if fitness was constrained to be an integer, and it was known that the maximum fitness was at most 100, then one could choose $\epsilon = 1/100$. Or there may be an ϵ so that any point whose fitness is within a factor of $1-\epsilon$ of the optimum is equivalent to being optimal from the point of view of the user.

A moderate value of C, say 2, can be used. Then repeated sampling from the Boltzmann distribution can be used to increase the probability of finding the optimum.

Clearly, the use of (14) may lead to exponentiation factors which would lead to overflow or underflow if equations (10) and (11) were to be implemented with standard floating point arithmetic. Thus, we store these quantities in logarithmic form.

In the computation of equation (11), the quantities $e_i(x_{s_i}) = e^{uf_i(\pi_{s_i}x)}$ are stored in logarithmic form as $uf_i(\pi_{s_i}x)$. The computation of equation (11) involves both product and summation of these quantities. To compute a product, we add the stored values. To compute a summation, we must convert the stored values to exponential form, perform the addition and then convert the final result to logarithmic form.

In the computation of a summation, additional precautions are taken to avoid overflows. For example, to compute

$$\sum_{j \in \mathcal{S}(i)} e^{uf_j(\pi_{s_j}x)}$$

the stored value could be computed as

$$\log \sum_{j \in \mathcal{S}(i)} e^{uf_j(\pi_{s_j}x)}$$

However, this computation might overflow. So we make the following modifications. Define F such that e^F is the maximum representable floating point number in the language to be implemented. Let $M = max_{j \in \mathcal{S}(i)} \ uf_j(\pi_{s_j}x)$, $E = F - 20$ and we rescale all the stored values to be added in such a way that the maximum value will be E. The computation of the stored value for the summation is done by

$$D + log\left(\sum_{j \in \mathcal{S}(i)} e^{uf_j(\pi_{s_j}x) - D}\right)$$

where $D = M - E$. In other words, first D is subtracted from each value to be added. The summation is then computed for the exponential of these values, which will not lead to an overflow as addition of

quantities $\leq e^E$ doesn't exceed e^F in general. We then take the logarithm of this result. The value D that we subtracted earlier has be added to this value in order not to change the final result of addition.

3.4 Sampling the Boltzmann distribution

The final phase of our algorithm is to sample the Boltzmann distribution using the factorization computed and stored as described above. The sampling applies formula (10) sequentially from $i = 1$ to $i = n$. Since $c_1 = \emptyset$, the first step is to choose $\pi_{b_1} x$ according to the probability distribution $p(\pi_{b_1} x)$. Note that $\pi_{b_1} x$ is required to determine $\pi_{c_2} x$, so the next step is to choose $\pi_{b_2} x$ according to the probability distribution $p(\pi_{b_2} x | \pi_{c_2} x)$. At step i, $\pi_{c_i} x$ has already been determined, and so $\pi_{b_i} x$ can be chosen according to the probability distribution $p(\pi_{b_i} x | \pi_{c_i} x)$. As x is chosen, we can compute $p(x)$ by accumulating the factors of the product of (10).

Clearly, the time necessary to compute one sample is $\Theta(\ell)$.

Since it is the exact Boltzmann distribution algorithm that is being sampled, the probability of sampling any optimal point is exactly the same as the probability of sampling any other optimal point. Thus, if α is the probability of the optimal set of points in the Boltzmann distribution, and if x is any optimal point, then the number of optimal points is $\alpha / p(x)$. If the exponentiation factor is sufficiently large that α is close to 1, then $1/p(x)$ is a good approximation for the number of optimal points. This is illustrated in section 5.

4 What if the running intersection property does not hold?

The running intersection property is a strong condition on the support sets of the subfunctions of the ADF decomposition of the fitness function.

There are two basic approaches if an ordering of the support sets with the running intersection property cannot be found. The first method expands the support sets or until the running intersection property is satisfied. The second method ignores some of the dependencies in the ADF.

One example of the first approach is the junction tree method which we briefly overview. The graph G_{ADF} of an ADF is defined as follows: The vertices are the variables of the ADF. Two vertices are connected by an arc if the corresponding variables are contained in a common subfunction. In other words, two variables are connected by an arc if there is epistatic interaction between the variables. (This assumes that the ADF cannot be simplified.) An algorithm for constructing a junction tree is briefly described in [14], and is more completely described in [13, 9, 10]. The vertices of the junction tree are clusters of vertices of G_{ADF}. The junction tree satisfies the junction property which is equivalent to the running intersection property. The difficulty is that the vertices of the junction tree may correspond to large subsets of variables which leads to sizes of conditional probability tables whose size is exponential in the string length.

Another example of the first approach is the subfunction merger heuristic algorithm given in [14]. This has the same limitation of possibly producing exponential size conditional probability tables.

In the second approach, either some points are dropped from the support sets of the ADF, or some subfunctions are dropped from the ADF, in order to achieve the running intersection property. This has the effect of ignoring some of the variable dependencies in the fitness function. We do not explore this approach in depth, but we point our that the Linkage Detection Algorithm used in this paper gives a lot of information which can be used to choose dependencies to ignore. For example, if we want to know

the strength of interaction between two variables, we can first look at the Walsh coefficient of the fitness whose index is the mask of the set consisting of the two variables. For example, if we want to know the magnitude of the direct interaction between x_1 and x_5 with $\ell = 5$, we would look at the Walsh coefficient whose index is the binary string 01001 (where bit positions are labeled starting at 0 from the left). We can also find information about higher order interactions involving these two variables by looking at the magnitudes of the Walsh coefficients whose indices are masks of sets containing the two variables.

Since the Linkage Detection Algorithm gives complete information about the fitness function, it is possible to try multiple approaches of the type described above without doing any more function evaluations. If the approach is to ignore interactions, then the cost of computing alternative approximate factorizations is much less than the cost of running the Linkage Detection Algorithm, so it would be possible to try many alternate ways of ignoring interactions.

5 Numerical results

The purpose of this section is both to validate the theory and to demonstrate that our algorithm can solve difficult problems.

5.1 An overlapping random needle function

The first test function is an ADF where the subfunctions are needle-in-the-haystack functions with a randomly placed needle. In other words, the value of the subfunction is 1 for all strings except one randomly chosen string. Recall that this is the most difficult kind of subfunction for the Linkage Detection Algorithm. For this string, the value is randomly chosen to be either $1 + \epsilon$ (positive needle) or $1 - \epsilon$ (negative needle) where $\epsilon = 0.1$ for our experiments. The degree of epistasis is $k = 4$, and there are $\ell - k + 1$ subfunctions. The support set of the ith function f_i is $\{i - 1, i, \ldots, i + k - 2\}$, so there is an overlap of $k - 1 = 3$ bits between f_i and f_{i+1}. It turns out that this test function can have many optima. In general, not all of the positive needle subfunctions can simultaneously have the needle set. Thus, the optimal value is somewhere between $\ell - k + 1$ and $(\ell - k + 1)(1 + \epsilon)$, and it is almost always the case that the suboptimum is ϵ less than the optimum.

It is interesting to note that if the exponentiation factor is set to be sufficiently large (say by inequality (14) for a reasonably large value of C), then the approximate number of optimal points can be determined from the probability of any optimal point. The probability of all optimal points are the same, and when the factorization is sampled, it is easy to accumulate the probability of the point that results from the sample. As a typical example, on one run with $\ell = 20$ and exponentiation factor 100 there were 1424 optimal points (as determined by enumerating all points in the search space), and the probability of an optimum as determined by factorization was 7.01528×10^{-4}. Note that $1/1424 = 7.02247191011 \times 10^{-4}$. With exponentiation factor 1000, the probability of an optimum was $7.02247191009 \times 10^{-4}$.

Figure 1 shows the number of function evaluations used in running this test function for various string lengths. The number N of order-j probes used was determined by the formula formula given in [7]:

$$N \geq \begin{cases} \frac{\ln(1-\delta^{1/J})}{\ln(1-2^{j-k})} & \text{if } j < k \\ 1 & \text{if } j = k \end{cases} \tag{15}$$

Here J is the number of order-j epistatic sets which is bounded by $\ell\binom{k}{j}$. The success probability δ was chosen to be 0.9999. For example, if $\ell = 64$, $k = 4$, and $j = 2$, then $N = 52.7$.

Also plotted is a constant times $\ell^2 \log \ell$, the function evaluation complexity as predicted in [7]. The graph empirically verifies that the function-evaluation complexity of the linkage detection phase of the algorithm is $O(\ell^2 \log \ell)$.

5.2 A function with many suboptimal

The second test function is a function that comes close to realizing our conjectured upper bound on the ratio of the number of suboptimal points to the number of optimal points. It is as difficult a test function as we know for our algorithm. We assume that ℓ is a multiple of k. The support sets for this function are the same as for the previous function. This function has $\ell - k + 1$ subfunctions of two types.

$$f_i(x) = \left\{ \begin{array}{ll} 1+\epsilon & \text{if } bc(x) = k \\ 0 & \text{if } bc(x) = 0 \\ 1 & \text{otherwise} \end{array} \right\} \quad \text{for } (i-1) \mod k = 0$$

$$f_i(x) = \left\{ \begin{array}{ll} 0 & \text{if } bc(x) = 0 \\ 1 & \text{otherwise} \end{array} \right\} \quad \text{for } (i-1) \mod k \neq 0$$

Here, $bc(x)$ is the "bit count" of x, i. e. the number of 1s in x. It is not hard to see that the unique optimum is the all-ones string with fitness $\ell - k + 1 + k\epsilon$. A suboptimum point can be obtained by setting the bits of one of support sets s_i with $(i-1) \mod k = 0$ to be any string except the all-ones string or the all-zeros string. For each such i, there are $2^k - 2$ such points, and there are ℓ/k such i. Thus, there are $(2^k - 2)\ell/k$ suboptimal points.

Figure 2 verifies the formula

$$u \geq \frac{\ln C + k \ln 2 + \ln \ell - \ln k}{\epsilon}$$

which is analogous to formula (14) for this case. The string length used was 128, $k = 4$, and the number of samples of the Boltzmann distribution was 1000. The values of ϵ were chosen as shown in the graph. Then trial and error was used to determine the exponentiation factor that would give the probability of an optimal point as 0.99 accurate to 0.001.

We also did a number of runs of this test function with string length 256 and verified that it consistently found the optimum with high probability.

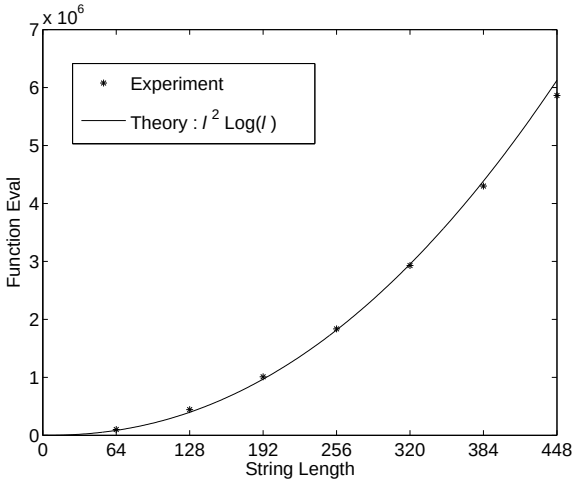

Figure 1: Number of function evals for overlapping random needle

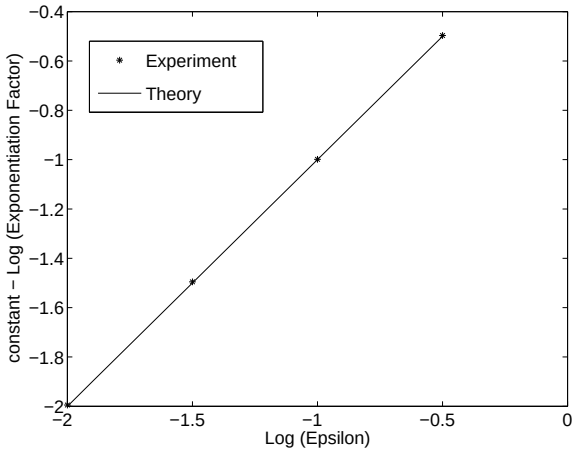

Figure 2: Exponentiation factor as a function of epsilon for a function with many suboptimal points

5.3 A scaled concatenation of trap functions

The third function is a concatenated scaled trap function (with non-overlapping subfunctions). The string length was 128 and k was 4, so there were 32 subfunctions.

To defi ne the subfunction before scaling, we start with a counting-ones function. Then the fi tness of the all-zeros string was made 1 larger than the fi tness of the all-ones string. A scaling factor of $0.7498^{(i-1)}$ was applied to subfunction f_i, and the order of the f_i was randomized. This scaling factor means that the subfunction with the smallest scaling has a scale of 10^{-4} times the subfunction with the largest scaling.

The optimum could be found with high reliability. For example, for an exponentiation factor of 20000, the algorithm gave a probability of 0.877 for the optimum.

6 Conclusion

The Boltzmann distribution of fitness function is the function exponentiated and then normalized into a probability distribution. The parameter of the Boltzmann distribution is the exponentiation factor or inverse temperature. As the exponentiation factor goes to infinity, the probability becomes increasing concentrated around the optimal points. It has been argued [18, 14] that sampling from the Boltzmann distribution would be the ideal genetic algorithm. (See the end of section 2.) Diversity is automatic since the Boltzmann distribution is over the entire search space. However, in general, sampling from the Boltzmann distribution is not feasible.

This paper shows that sampling from the exact Boltzmann distribution is possible for a large class of fitness functions over fixed length binary strings. To be in the class, a fitness function has to satisfy 2 properties. First, it must be k-epistatic. This means that interaction between variables (or bits) must be limited to at most k variables. Fitness functions that satisfy this condition include MAX k-SAT, NK-landscapes, and order-k concatenated trap and deceptive functions. A k-epistatic fitness function is additively decomposable into subfunctions each of which depends on at most k bits. Second, the support sets of the subfunctions of the additive decomposition must satisfy a technical condition called the running intersection property. The running intersection property requires that these support sets have a tree-like intersection property.

If the fitness function is k-epistatic, then the first phase of our algorithm (which is taken from [6, 7]) determines the k-epistatic structure by sampling the fitness function (which is assumed to be given as a black box). This randomized algorithm succeeds with any predefined success probability. Additionally, the algorithm computes an additive decomposition of the fitness. If this additive decomposition satisfies the running intersection property, then next phase of our algorithm computes an exact factorization of the Boltzmann distribution of the fitness function for any given exponentiation factor. The final phase of our algorithm uses this factorization to sample from the exact Boltzmann distribution. If the exponentiation factor is chosen to be sufficiently large, the algorithm will find optimum points with high probability. Furthermore, it will provide an accurate estimate of the number of optimal points, and all optimal points are equally likely to be sampled.

Our algorithm has a rigorous complexity analysis if k is considered to be constant. (The algorithm is exponential in k.) The first phase, the Linkage Detection Phase, requires $O(\ell^2 \log \ell)$ function evaluations. (If the subfunctions of the ADF decomposition of the fitness are non-overlapping, this can be reduced to $O(\ell \log \ell)$ by using the result of [25].) The construction of the factorization can be done in $O(\ell)$ time. Each sampling of the Boltzmann distribution requires $O(\ell)$ time.

Our implementation overcomes potential floating overflow problems with large exponentiation factors. We give formulas to guide the choice of the exponentiation factor as a function of the fitness separation between the optimum and suboptimum points and the ratio of the number of optimum and suboptimum points. We give numerical examples which demonstrate that the algorithm works in practice.

Unlike almost all other algorithms developed in an evolutionary computation framework, we have given conditions under which the algorithm is guaranteed to find the optimum with any predefined success probability in time which is polynomial in the string length.

References

[1] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann Publishers.

[2] J. S. de Bonet, C. L. Isbell, Jr., and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. C. M. et. al., editor, *Advances in Neural Information Processing Systems*, volume 9, page 424. MIT Press, 1997.

[3] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult optimization problems using fast messy genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, California, 1993. Morgan Kaufman.

[4] G. R. Harik and D. E. Goldberg. Learning linkage. In R. K. Belew and M. D. Vose, editors, *Foundations of Genetic Algorithms 4*, pages 247–262. Morgan Kaufmann, San Francisco, CA, 1997.

[5] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE-EC*, 3(4):287, November 1999.

[6] R. E. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. In Erick Cantú Paz et al., editor, *Genetic and Evolutionary Computation – GECCO 2003*, Lecture Notes in Computer Science LNCS 2724, pages 1003–10014. Springer Verlag, 2003.

[7] R. E. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. *Evolutionary Computation*, 12(4):517–545, 2004.

[8] J. Holland. *Adapdation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[9] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[10] F. V. Jensen and F. Jensen. Optimal junction trees. In *Proceedings of the 10th conference on uncertainty in artificial intelligence*, pages 360–366, Seattle, 1994.

[11] H. Kargupta. The gene expression messy genetic algorithm. In *International Conference on Evolutionary Computation*, pages 814–819, 1996.

[12] H. Kargupta and B. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69, 2001.

[13] S. L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.

[14] H. Mühlenbein and R. Höns. The estimation of distributions and the maximum relative entropy principle. *Evolutionary Computation*, 13(1):1–28, 2005.

[15] H. Mühlenbein and T. Mahnig. Convergence theory and application of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7(1):19–32, 1999.

[16] H. Mühlenbein and T. Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.

[17] H. Mühlenbein and T. Mahnig. Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal of Approximate Reasoning*, 31:157–192, 2002.

[18] H. Mühlenbein and T. Mahnig. Evolutionary algorithms and the Boltzmann distribution. In *Foundations of genetic algorithms (FOGA-7)*, pages 133–150, San Mateo, 2003. Morgan Kaufmann.

[19] H. Mühlenbein, T. Mahnig, and A. O. Rodriguez. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

[20] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I, binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature PPSN IV, 1996*, pages 178–187. Springer Verlag, 1996.

[21] M. Munetomo and D. E. Goldberg. Linkage identifi cation by non-monotonicity detection for overla pping functions. *Evolutionary Computation*, 7(4):377–398, 1999.

[22] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, 13-17 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[23] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation*, 8(3):311–340, 2000.

[24] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[25] M. J. Streeter. Upper bounds on the time and space complexity of optimizing additively separable functions. In Kalyanmoy Deb et al., editor, *Genetic and Evolutionary Computation – GECCO 2003, procedings part II*, Lecture Notes in Computer Science LNCS 3103, pages 186–197. Springer Verlag, 2004.

[26] D. Thierens. *Analysis and design of genetic algorithms*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[27] Q. Zhang. On stability of fi xed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. *IEEE Transations on Evolutionary Computation*, 8(1):80–93, 2004.

[28] Q. Zhang and H. Mühlenbein. On the convergence of a class of estimation of distribution algoirthms. *IEEE Transations on Evolutionary Computation*, 8(2):127–136, 2004.