Drift and Scaling in Estimation of Distribution Algorithms

J. L. Shapiro jls@cs.man.ac.uk
Department of Computer Science, University of Manchester, Manchester M13 9PL UK

Abstract

This paper considers a phenomenon in Estimation of Distribution Algorithms (EDA) analogous to drift in population genetic dynamics. Finite population sampling in selection results in fluctuations which get reinforced when the probability model is updated. As a consequence, any probability model which can generate only a single set of values with probability 1 can be an attractive fixed point of the algorithm. To avoid this, parameters of the algorithm must scale with the system size in strongly problem-dependent ways, or the algorithm must be modified. This phenomenon is shown to hold for general EDAs as a consequence of the lack of ergodicity and irreducibility of the Markov chain on the state of probability models. It is illustrated in the case of UMDA, in which it is shown that the global optimum is only found if the population size is sufficiently large. For the needle-in-a haystack problem, the population size must scale as the square-root of the size of the search space. For the one-max problem, the population size must scale as the square-root of the problem size.

Keywords

Estimation of distribution algorithms, probability building genetic algorithms, Markov chain analysis, convergence analysis, needle-in-a-haystack problem, counting-ones problem.

1 Introduction

Estimation of Distribution algorithms (EDAs) have become an increasingly popular approach to search for good solutions to hard problems. They are an example of the *generate-and-test* approach to heuristic search, which means that instead of using specific knowledge about a problem, they generate candidate solutions to the problem, evaluate them, and on the basis of the results determine which candidates to generate next. Where EDAs differ from other algorithms is that they use probability models to generate the candidate solutions to the problem. These algorithms evolve the probability model, whereas most heuristic search techniques evolve a potential solution or population of potential solutions. Usually, the probability model is represented as a parameterised graphical model with some predetermined properties of the graph. The trend in EDA research has been in developing algorithms which allow increasingly complex structure in the graphs and finding new ways of determining the structure from the data, as well as trying the algorithms on various application domains. For reviews, see Pelikan et al. (2002); Larrañaga and Lozano (2002); Pelikan (2002), as well as the articles in this special issue.

In addition to developing new and more complex EDAs and new applications for them, it is important to try to understand how these algorithms work, and how to get them to work better. With every new general search algorithm comes a new

dynamical mechanism which we must try to understand if we are to make best use of the algorithm. EDAs are particularly novel algorithms, because they do not act directly on solutions to the problem, but act on the probability models which generate them.

The goal of this paper is to discuss one aspect of the dynamics of the EDA algorithm. EDAs are susceptible to what is analogous to premature convergence in genetic algorithms. For most EDAs, any probability function which generates a single vector of values with probability 1 will be a fixed point of the dynamics. What is important is the fact that these fixed points can be attractors of the dynamics of the algorithms, i.e. they can be stable fixed points. The properties of these attractors depend upon the particular problem being optimised. As a consequence, these attractors are only removed when aspects of the algorithms such as the population size or the learning rate are scaled with the system size in a suitable way. The required scaling depends very strongly on the problem being solved. For hard problems, exponential scaling will be required, whereas for easy problems polynomial scaling is necessary. This is a very important feature of these algorithms. The problem-dependent nature of the scaling means we will not know the scaling to use for a real problem. In this paper, the origin of this phenomenon will be discussed as consequence of the properties of the transition probabilities between probability models. The phenomenon will be illustrated in UMDA, a simple EDA. We will also discuss some mechanisms which have been proposed to remove this phenomenon.

2 Theoretical Issues involving EDAs

In this section, we will establish some basic notation, describe some of the theoretical issues surrounding EDAs, and review previous theoretical work. In describing EDAs, the terminology used will be that of evolutionary algorithms. A candidate solution to the problem (i.e. a state from the search space) will be referred to as a string, which in this paper will be a binary vector of length L. The measure of quality of a string is its fitness, which is to be maximised. In addition, one has a probability model which is described by continuous parameters and these parameters are (conditional) probabilities of combinations of variables, and discrete parameters describing the structure of the interactions between these variables from some allowed set of structures. Since there are two types of parameters: the parameters associated with the probability model (e.g. $\mathrm{prob}\,(x|y)$) and the parameters associated with the search algorithm (e.g. population size N), I will refer to the former as probability parameters and the later as algorithmic parameters.

The basic steps of a general EDA are as follows.

- Initialise the model to some prior (e.g. a uniform distribution);
- Repeat
  - **Sampling step**: generate a population of size N by sampling from the probability model;
  - **Evaluation step**: determine the fitnesses of the strings in the population;
  - **Selection step**: create an improved data set by selecting the better solutions and removing the worse ones from the population;
  - **Learning step**: (or modelling step) create a new probability model from the old model and the improved data set (e.g. as a mixture of the old model and the most likely model given the improved data set);

- until (stopping criterion met)

Different algorithms are largely distinguished by the complexity of the probability model used, the method for determining the probability from the data, or the state space on which they act.

The first questions which should be asked about these algorithms are the questions asked about any search algorithms: are they effective and for what types of problems, are they efficient, do they always find the solution, and so forth. There is empirical evidence that they are effective for certain problems. For example, one of the earliest and simplest examples of an EDA is Population-Based Incremental Learning (PBIL), proposed by Baluja (1994). He did a study in which he showed that PBIL is more effective than genetic algorithms and various hill-climbing algorithms on a range of problems (Baluja, 1997).

However, it also turns out that this algorithm does not always find a good solution even for very easy problems like *one-max*. This was first investigated by Gonzalez et al. (2001b). They showed that if the initial probability parameters are sufficiently close to 1 or 0, the probability that the global optimum is *never* found can be arbitrarily close to 1. This was shown for $N = L = 2$, but it is easy to generalise for arbitrary string lengths and population sizes. In another paper (Gonzalez et al., 2001a), the authors show that in the one-max problem PBIL does converge to the optimum if the learning rate, α, is sufficiently small. Thus, in combination, these papers show that the optimum is only found with sufficiently slow learning of the probability model.

The question of when PBIL fails to converge to the optimum and how small learning rate needs to be has been answered for specific problems by Shapiro (2003b). It was shown that PBIL does not converge to the global optimum unless the learning rate α is smaller than a critical value, $\alpha_c(L)$, which is a function of the string length and the specific problem. As examples, for the one-max problem, α_c is $O(1/\sqrt{L})$; for the needle-in-a-haystack problem α_c is $O(2^{-L})$. Empirical work (Correa and Shapiro, 2004) showed that for 3-SAT near its phase transition, α_c scales approximately as $\exp(-0.2L)$, and for the leading ones problem α_c scales as L^{-b} with $b = 1.44 \pm 0.03$. Thus, the algorithm does not give constant improvement, even if it is run forever. It gets stuck. The choice of algorithmic parameters which avoids this depends very strongly on the problem, and so cannot be predicted in advance.

Another important question concerning EDAs concerns the benefits of dependencies between the variables. Very simple variants of the algorithm, such as PBIL, treat all of the variables as independent. Other algorithms not only assume dependencies between the variables in the probability model, but try to learn what those dependencies are. This increased complexity has a computational cost. The task of computing the probability parameters tends to scale as $O(LN)$ for models which treat the parameters as independent, but it is a higher order polynomial in L when there are dependencies to be learnt. Are there increased benefits associated with this increased cost, and if so, why? This question is largely unanswered, although some suggestions regarding trapping functions and deception have been made (Pelikan, 2002). (In fact, some studies hide this cost; namely those which compare complex EDAs with simple ones with regard to number of *fitness evaluations*. Most of the increased cost in learning complex model structure is not counted with this comparison.)

It is tempting to think that these algorithms will teach us about the problem as well as solve it — they will show us the true interdependences of the variables of the problem. However, a health warning should be attached to this view. The complexity of the dependencies are limited by the need to sample from the probability model. We

can only sample efficiently from graphs with no loops. However, it is precisely loops which make problems hard. Any problem which can be represented as acyclic graph will be easy to optimise. Thus, the dependencies found by the EDA only be local picture at best.

In a study comparing PBIL with MIMIC (Johnson and Shapiro, 2001), it was found that the two algorithms behave comparably when all other aspects are controlled for, even for problems with chain-like and more complex variable interactions. The one exception was when the search problem was exactly a chain (a spin-glass chain) did MIMIC clearly outperform PBIL.

3 Drift and Scaling in EDAs

This is the crux of this paper. There is a fundamental flaw in the way the basic EDA works. The flaw is this: standard EDAs do not search space in an unbiased way. They are more likely to look where they have looked before. This is true even when there is no evidence that where they have looked before is any better than anywhere else. As a consequence, they are not ergodic; they do not search the entire space in a manner which is determined solely by the properties of the search space. The reason for this is simple. The population generated by sampling from the probability model will have similar statistics to the model, but there will be random fluctuations around them. These fluctuations will be some random direction. When the probability model is updated, this random direction can be incorporated into the new model. Therefore, the random direction is reinforced. When the algorithm is iterated, some of the movement is determined by this reinforced random direction. As the dimension of the state space is increased, there are an increasing number of random directions and the effect becomes more pronounced.

This effect is analogous to the phenomenon of *drift* in evolutionary dynamics, and I will refer to it as drift here. (In population genetics, the term drift refers to the loss of genetic diversity due to finite population sampling.) To see why this is, consider what happens when the algorithm is run without the selection. Obviously, the expected mean of any property of the sampled population would be the mean of the probability model. However, the variance of the sampled population will be less than that of the probability model. It is very well known that the variance of a sample of size N has a expected value of size $\sigma^2(1 - 1/N)$ where σ^2 is the variance in the parent distribution. Most EDAs do not compensate for this; when the new probability model is produced, it attempts to model the new population and therefore has a reduced variance. When this is iterated repeatedly, the variance of the sampled population gets smaller and smaller and decays to zero. The probability model evolves to one which can only generate identical configurations.

In an EDA acting on a real problem there is a conflict between the drift effect and the directed search introduced through selection. Good solutions are found only if the directed search is stronger than the random drift. Whether that is the case depends on the parameters of the algorithm and on the properties of the problem being solved. Thus, this effect means that the parameters must be tuned to the specific problem to make the drift effect weak relative to the directed search. This is very bad because one does not know how to set these parameters for the particular problem being solved. This drift effect, how it causes the algorithms to fail, and a possible mechanism to mitigate against it, is the topic of this paper.

The importance of the drift effect has been overlooked by many theorists for several reasons. First, it is not apparent in the expected behaviour, only in fluctuations

around the expected behaviour. Since there is no preferred direction to these fluctuations, it is not seen when only expected motion is considered. For example, in a theoretical study of PBIL carried out by Höhfeld and Rudolph (1997), it was shown that in the counting-one problem and similar problems, the mean of the parameters of PBIL converges to the global optimum. The authors concluded from this that PBIL *always* converges to the global optimum in these problems. In fact, for typical values of the learning rate, PBIL almost never finds the global optimum. The drift fluctuations mentioned above cause the parameters to move toward a random corner of the hypercube. Once they get sufficiently close to a corner, they have a vanishingly small probability of leaving, as was shown by Gonzalez et al. (2001b). So, although the mean converges, the algorithm almost never converges to the optimum except for a very limit range of values of the learning rate. An example illustrating this in a continuous EDA has already been presented in Gonzalez et al. (2002).

The second reason for ignorance of this essential effect is a tendency on the part of theorists of evolutionary algorithms to work in the limit of a large population size. In this limit, there are no fluctuations and the drift is completely removed. However, this is the wrong limit to work in to describe real evolutionary algorithms (Prügel-Bennett, 2003), because in that limit the asymptotics are dominated by population sizes which exceeds the size of the search space. As is standard elsewhere in computer science, a more appropriate limit is large L with parameters of the algorithm, *including the population size* held fixed, or increasing as a polynomial in L.

3.1 The Algorithm as a Markov Process

Here a more formal description of the problem will be given in terms of the transition probabilities between probability functions. The key idea here is to view an iteration as a Markov chain in the space of the parameters of the probability model. Then we will ask whether this Markov chain has suitable properties to act as an effective search algorithm. We will see that it does not, but standard methods can be used to make it have the desirable properties.

The starting point of an EDA is the probability model. It is assumed that this is parametrized by a set of continuous parameters, which I denote γ and possibly a set of structural parameters, which I denote θ. The generation of a population of test examples corresponds to sampling a population of N states $\mathbf{x}$ from this distribution, $P(\mathbf{x}|\gamma,\theta)$.

Let D_s denote the data after selection. This will depend on the problem, denoted $\mathcal{P}$. If we combine sampling and selection, we can write the probability of the selected set being a particular set D_s as $P_s(D_s|\gamma,\theta,\mathcal{P})$; it depends upon the parameters of the probability model and the problem being solved $\mathcal{P}$. (It also depends on the population size and parameters of the selection algorithm which are suppressed for notational simplicity.)

The next step is the use of the data D_s to update the probability model. This requires a learning algorithm. This will lead to a new set of parameters, γ',θ'. In general, there will be a probability of learning a set of parameters which depends on the selected set and the old value of the parameters, $P_L(\gamma',\theta'|D_s,\gamma,\theta)$ (or maybe just on the selected data D_s).

Combining all of this gives us a view of the dynamics of EDAs as transitions between probability models, $T_{\mathcal{P}}(\gamma',\theta'|\gamma,\theta)$. Here $T(x'|x)$ denotes the transition probability from x to x'. The transition probability depends on the problem, and is given

by

$$T_{\mathcal{P}}(\gamma', \theta'|\gamma, \theta) = \sum_{D_s} P_L(\gamma', \theta'|D_s, \gamma, \theta) P_s(D_s|\gamma, \theta, \mathcal{P}), \quad (1)$$

where the sum is over all possible selected populations (remember that P_s combines both the effects of sampling and selection). Thus, an EDA defines a Markov Chain on the parameter space of the probability model.

Unless the Markov Chain satisfies certain necessary conditions, the EDA will not be an effective search algorithm.

1. Wherever the algorithm is currently searching, it should have the potential to search all other parts of the search space;

2. If the search space is locally flat and therefore uninformative, the algorithm should search all points with equally likelihood.

The first point states, in the language of properties of Markov chains, the process should be irreducible and ergodic. Irreducible means that every state can communicate with every other. Ergodic means that the probability of returning to any state in finite time is 1. It is well-known that an irreducible, ergodic, aperiodic Markov chain has a unique fixed-point distribution (Motwani and Raghavan, 1995, chapter 6).

The second property is that on a flat landscape the fixed-point distribution should be the uniform distribution. Most heuristic search algorithms such as simulated annealing, standard genetic algorithms, and hill-climbing with restarts have these properties (obviously, hill-climbing without restarts lacks the first property). The second property is particularly important; any bias in the search should be introduced by the evidence from the problem, not be intrinsic to the algorithm.

In general, EDAs lack these properties in particularly strong ways. A general EDA has a large number of absorbing subspaces; any set of parameters with one or more of the probabilities equal to 0 or 1 cannot communicate with states with different values for those probabilities. Most EDAs use the frequency to model probabilities; if these are 0 or 1, no further variability can be produced in those variables. Thus, it is neither irreducible nor ergodic. The probability of returning to a state is less than one due to the finite probability of reaching an absorbing state first.

On a flat landscape, the absorbing subspaces are attractive. Once the EDA gets near them they are more likely to continue toward them than move away. This has been shown already for PBIL (Shapiro, 2003b); it will be shown in detail for UMDA in section 4.1. The algorithm is strongly non-ergodic; whole regions of the probability parameter space remain unexplored when the probability parameters stop changing and become fully absorbed. This is also true on non-flat landscapes for poor choices of algorithmic parameters.

The reason for the existence of absorbing subspaces is pretty obvious — once some parameters take degenerate values of 0 or 1 it will not be possible to sample any other values for those variables. That these are typically attractive is more subtle, and is caused by the decreasing variance as absorbing states are approached. Let γ be a set of parameters such that one of the conditional probabilities is close to one. To make it concrete, suppose $P(X_i|X_j, X_k)$ is very close to one, say $1 - \epsilon$. Here X_i, X_j and X_k are variable values. Let γ' be another set of parameters with that conditional probability even closer to one, say $1 - \epsilon'$ with $\epsilon' < \epsilon$. All other probability parameters of γ' are the same as in γ. In most EDA dynamics, the probability of going from γ to γ' is greater than the probability of going back, i.e. $T(\gamma'|\gamma) > T(\gamma|\gamma')$. Why? Because the variability

in the component X_i decreases as its probability approaches one. In the sample from γ', as long as the values X_j, X_k are realised, those two values are going to be very predictive of the value of the i variable. Most algorithms for choosing the structure are greedy algorithms, they look for the variable value combinations which are most correlated or most predictive by some measure. Thus, the algorithm for choosing the structure is more likely to chose this dependency than was the case in state γ. Once this dependency is chosen, the variability in the value of the ith variable is proportional to $\epsilon'(1-\epsilon')$ which is smaller than the variability in that component in the state γ, $\epsilon(1-\epsilon)$. Thus, there is less variability in the primed variables with which to take the system back to the unprimed state. The rate of states entering regions near the absorbing states is greater than the rate of states leaving these regions, and so the absorbing states are attractive.

3.2 Methods for alleviating drift

One method which have been proposed to fix this is to enforce detailed balance (Shapiro, 2003b). The detailed balance equation is a condition on the transition probabilities to make the chain converge to a given distribution (see, for example, Neal, 1993; Tanner, 1996). It is well known that in order to make a Markov Chain converge to a probability distribution $\pi(x)$, it is sufficient to have the transition probabilities obey the detailed balance condition,

$$T(x'|x)\pi(x) = T(x|x')\pi(x'). \tag{2}$$

In our case, it is desired that if the search space is flat, the stationary distribution will be uniform. Therefore, the condition is

$$T_{\mathcal{R}}(\gamma', \theta'|\gamma, \theta) = T_{\mathcal{R}}(\gamma, \theta|\gamma', \theta'). \tag{3}$$

Here the flat problem is denoted $\mathcal{R}$. One way to insure this is to use rejection sampling. That is, define a new transition probability T^{DB} in this way,

$$T^{DB}_{\mathcal{P}}(\gamma', \theta'|\gamma, \theta) = T_{\mathcal{P}}(\gamma', \theta'|\gamma, \theta)A(\gamma', \theta'|\gamma, \theta), \tag{4}$$

where A is the *acceptance probability*. The transition occurs with probability A and does not occur with probability $1-A$. If the transition does not occur, the parameters remain unchanged for the current iteration. The acceptance probability can be given by the Metropolis-Hastings formula,

$$A(\gamma', \theta'|\gamma, \theta) = \min\left[1, \frac{T_{\mathcal{R}}(\gamma, \theta|\gamma', \theta')}{T_{\mathcal{R}}(\gamma', \theta'|\gamma, \theta)}\right] \tag{5}$$

When the transition probability from the initial state to the final state is less than the transition probability back, that transition is always accepted. However, when the transition probability from the initial state to the final state is greater than the transition probability back, that transition may be rejected.

Since the transition probabilities are derived from the learning equations, detailed balance is tantamount to a change in the learning equations as follows,

$$P^{DB}_{L}(\gamma', \theta'|D, \gamma, \theta) = P_L(\gamma', \theta'|D, \gamma, \theta)A(\gamma', \theta'|\gamma, \theta), \tag{6}$$

where A is defined as in equation (5), which can be written in terms of the learning equations as,

$$A(\gamma', \theta'|\gamma, \theta) = \min\left[1, \frac{\sum_D P_L(\gamma, \theta|D, \gamma', \theta')P_s(D|\gamma', \theta')}{\sum_D P_L(\gamma', \theta'|D, \gamma, \theta)P_s(D|\gamma, \theta)}\right] \tag{7}$$

This is the fundamental equation of the detailed balance principle applied to EDAs.

The detailed balance equation is easy to state, but can be hard to implement, for several reasons. First and foremost, it requires an expression for the transition probabilities for the flat landscape. This seems particularly hard to derive when there are structural parameters, and it has not been done in that case. It is also a problem in some EDAs that it is hard to predict in advance what values the continuous parameters can take. For example, in standard PBIL, the transition back does not return to the same state (due to a variable step-size). This must be modified, else all moves will be rejected.

It is important to emphasise that this method only makes the dynamics obey detailed balance on the flat landscape, because that is the only case in which the appropriate final distribution π is known. Nonetheless, the same equations will be used for all problems. For arbitrary problems the fixed point distribution will not be known, but the use of the detailed balance equations removes the absorbing states at the corners of the hypercube, and makes the algorithm ergodic and more robust.

A different approach to counteract drift is to introduce a prior which biases the modelling toward the initial probability model. Mühlenbein (1997) proposed an operator for EDAs which estimates frequencies biased toward a random guess. Suppose $\tilde{\gamma}_i$ is the fraction of 1's at site i. Then, the appropriate estimate of the probability of a 1 at site i is

$$\gamma_i = \frac{\tilde{\gamma}_i + m}{1 + 2m}, \tag{8}$$

where m is a mutation-like parameter. This will be recognised as the maximum posterior estimate of the binomial distribution using as the prior a β-distribution with both parameters equal to $mN + 1$; the prior biases the estimate toward $1/2$. The parameter m is analogous to mutation in GAs, and this has been called Bayesian mutation. The result here is that on a flat landscape, $T(\gamma'|\gamma) > T(\gamma|\gamma')$ if γ' is closer to $1/2$ than γ. Thus the movement is away from, rather than toward the absorbing states.

Formally, Bayesian mutation and the Detailed Balance EDA are variations of the same principle. Rewrite equation (3) more generally as,

$$T_{\mathcal{R}}(\gamma', \theta'|\gamma, \theta)\pi(\gamma, \theta) = T_{\mathcal{R}}(\gamma, \theta|\gamma', \theta')\pi(\gamma', \theta') \tag{9}$$

where π is a prior which the algorithm should converge to given no evidence from the fitness function (i.e. a flat landscape). The detailed balance algorithm proposed in Shapiro (2003b) and studied further in Correa and Shapiro (2004) uses a uniform distribution as a prior. Bayesian mutation uses a product of beta functions, one for each conditional probability, as the prior. The former algorithm uses rejection sample (equation (5)) to implement the prior. Bayesian mutation is equivalent to Gibbs sampling (see, for example, Chapter 6 of Tanner (1996)).

One difference between Bayesian mutation and the detailed balance EDA, is that the former contains a (hyper)parameter, m, and the latter is parameter free. Which has the advantage is unclear. A parameter implies greater control, but it has to be set. It must be large enough to counteract drift and small enough not to counteract selection. This is discussed briefly in Shapiro (2003a). In the rest of this paper, we look at detailed balance in the simple case of UMDA.

4 Case Study: UMDA

Here the properties discussed abstractly in the previous section will be illustrated with a very simple EDA — UMDA algorithm acting on binary strings of length L. The

probability model treats each component of the string independently. The parameters of the model are the probabilities that each component takes the value 1, denoted,

$$\gamma_i \equiv P(x_i = 1). \tag{10}$$

At each iteration, N strings will be sampled. The i component of the μth string is denoted x_i^μ. Thus, the probability of generating a string $\mathbf{x}^\mu$ is

$$P(\mathbf{x}^\mu) = \prod_{i=1}^{L} \gamma_i^{x_i} \left(1 - \gamma_i\right)^{\left(1 - x_i^\mu\right)}, \tag{11}$$

and the probability of producing a population of N strings is just the product of equation (11) over μ.

Truncation selection will be used; the selected population consists of randomly selected strings with fitness greater than or equal to the mean fitness in the sampled population. The new model will be determined by the frequency of 1's at each site in the selected population. Algorithm 1 shows the algorithm. For completeness, Algorithm 2 and 3 show the detailed balance and Bayesian mutation versions.

Algorithm 1 Simple UMDA algorithm

Set $\gamma_i \leftarrow 1/2$ for all $i = 1 \ldots L$;
repeat
Sample N strings from the probability distribution equation (11) to make a population D.
Compute the mean fitness of the sampled population D.
Generate a new population D_s of N strings by selecting uniformly from those strings with fitness greater than or equal to the mean fitness.
for $i = 1$ to L **do**

$$\gamma_i \leftarrow \frac{1}{N} \sum_{\mathbf{x}^\mu \in D_s} x_i^\mu, \tag{12}$$

end for
until Stopping criterion met

4.1 UMDA on a flat landscape

The problematic properties of UMDA can be seen by considering its behaviour on a flat landscape. In this case, the selected population is just the sampled population, and transition probabilities are,

$$T_{\mathcal{R}}(\gamma'|\gamma) = \prod_i \left[\binom{N}{\gamma_i' N} \gamma_i^{N\gamma_i'} \left(1 - \gamma_i\right)^{N\left(1-\gamma_i'\right)} \right] \tag{15}$$

$$\approx \prod_i \left\{ \sqrt{\frac{N}{2\pi\gamma_i\left(1-\gamma_i\right)}} \exp\left[-\frac{N\left(\gamma_i - \gamma_i'\right)^2}{2\gamma_i\left(1-\gamma_i\right)} \right] \right\}. \tag{16}$$

The latter expression is a well-known Gaussian approximation to the binomial distribution; it is only good when both γ_i and γ_i' are not too close to 1 or 0. See, for example, Chapter 16 of Boas (1983) for a derivation.

Algorithm 2 Detailed Balance UMDA algorithm

Set $\gamma_i \leftarrow 1/2$ for all $i = 1 \ldots L$;
repeat
Sample N strings from the probability distribution equation (11) to make a population D.
Compute the mean fitness of the sampled population.
Generate a new population D_s of N strings by selecting uniformly from those strings with fitness greater than or equal to the mean fitness.
for $i = 1$ to L **do**

$$\gamma_i' \leftarrow \frac{1}{N} \sum_{\mathbf{x}^\mu \in S'} x_i^\mu, \tag{13}$$

$\gamma_i \leftarrow \gamma_i'$ with probability $min\left[1, \frac{T_{\mathcal{R}}(\gamma_i|\gamma_i')}{T_{\mathcal{R}}(\gamma_i'|\gamma_i)}\right]$ with $T_{\mathcal{R}}(\gamma_i|\gamma_i')$ given in equation (15).
end for
until Stopping criterion met

Algorithm 3 UMDA with Bayesian Mutation

Set $\gamma_i \leftarrow 1/2$ for all $i = 1 \ldots L$;
repeat
Sample N strings from the probability distribution equation (11) to make a population D.
Compute the mean fitness of the sampled population.
Generate a new population D_s of N strings by selecting uniformly from those strings with fitness greater than or equal to the mean fitness.
for $i = 1$ to L **do**

$$\gamma_i' \leftarrow \frac{1}{N} \sum_{\mathbf{x}^\mu \in S'} x_i^\mu, \tag{14}$$

$\gamma_i \leftarrow (\gamma_i' + m) \,/\, (1 + 2m)$.
end for
until Stopping criterion met

The approximation is included here because it can be seen by inspection that the transition probability is not symmetric, $T_{\mathcal{R}}(\gamma'|\gamma) \neq T_{\mathcal{R}}(\gamma|\gamma')$. If γ_i is closer to $1/2$ than γ_i', then the probability of moving from γ_i to γ_i' is greater than the probability of moving back. This is because the variance of the Gaussian in equation (15) is $\gamma_i\,(1-\gamma_i)$, whereas the variance around the state γ_i' is $\gamma_i'\,(1-\gamma_i')$ which is smaller. Although this approximation becomes invalid as the corners of the hypercube are approached, this asymmetry holds throughout the hypercube. The algorithm moves quickly toward corners of the hypercube, but more slowly away. States with $\gamma_i = 0$ or $\gamma_i = 1$ for any i are absorbing subspaces, corresponding to gene fixation in genetic algorithms. There are 2^L fixed point distributions; they have γ's equal to 0 or 1. In appendix B, another approximation is given which shows that near the corners of the hypercube, transitions toward the corners are more probable than those away.

To understand the rate of convergence to these attractors, we need to study the dynamics. Because the selected population is statistically equivalent to the sampled population, the parameters do not change in expectation over time,

$$\langle \gamma_i(t+1) \rangle = \gamma_i(t), \text{for all } i. \tag{17}$$

The angled brackets $\langle \cdot \rangle$ denote the expectation operator. Thus, the parameters remain unchanged on average.

Nonetheless, the system converges rapidly to one of the corners of the hypercube where it remains forever, it is just unbiased with regard to which corner. To see this, consider the expected variance of the sampled population,

$$v(t) \equiv \frac{1}{L^2} \sum_i \gamma_i(t)\left[1-\gamma_i(t)\right]. \tag{18}$$

Under the dynamics on a flat landscape,

$$\begin{aligned} \langle v(t+1) \rangle &= \left(1-\frac{1}{N}\right) \frac{1}{L^2} \sum_i \gamma_i(t)\left[1-\gamma_i(t)\right] && (19) \\ &= \left(1-\frac{1}{N}\right) \langle v(t) \rangle. && (20) \end{aligned}$$

The initial variance is $1/(4L^2)$, so the variance is

$$\langle v(t) \rangle = \frac{1}{4L^2} \left(1-\frac{1}{N}\right)^t. \tag{21}$$

This decays to zero in characteristic time approximately equal to the population size,

$$\tau = -1/\log\left(1-1/N\right) \approx N. \tag{22}$$

If one modelled algorithm as a random walk in parameter space, one would predict the expected time to the absorbing states to scale as $O(N^2)$, since an unbiased random walk takes N^2 steps to travel a distance N. Although the dynamics is unbiased, the decreasing variance as the absorbing states are approached makes this fixate so much faster.

As an absorbing subspace is approached, it becomes more likely that the system will fixate there. If the ith parameter is at γ_i, the probability if it getting stuck at $\gamma_i = 0$

is $1-\gamma_i$. To see this, let $w(\gamma_i)$ be the probability of fixating at 0 starting at γ_i. Since this probability is independent of time, w must obey the recursion,

$$w(\gamma) = \sum_{n=0}^{N} \binom{N}{n} \gamma_i^n \left(1-\gamma_i\right)^{(N-n)} w\left(n/N\right) \tag{23}$$

with the boundary conditions, $w(0)=1; w(1)=0$. That $1-\gamma_i$ is a solution can be tested directly.

The results on the flat landscape show that the algorithm will converge to a random corner of the hypercube in a time given by the population size. For any problem which has a flat subspace, it will be necessary to make the population size large enough so that the optimum will be found before this convergence takes place. In the next three sections, we will see that the required population size depends on the specific problem. Thus, one will not know in advance how large to make the population.

In high-dimensional spaces, flat subspaces are typical, and occur even in smooth problems such as the *one-max* problem. The reason for this is that for any given direction, there are many directions orthogonal to it. So if there is a direction in which the fitness function changes locally (a local gradient), there will typically be many directions orthogonal to that for which the fitness changes little or not at all. These are flat or approximately flat subspace, and the convergence described above will be important in determining the effectiveness of the algorithm. We shall see in section 4.3 that this phenomenon occurs in the one-max problem.

The detailed balance algorithm discussed in section 3.2 avoids these problems. It contains no absorbing states, because the detailed balance condition, equation (5) prevents the algorithm from every entering absorbing states. The system behaves like a random walk, thus the characteristic time to get near one of the corners of the hypercube is $O(N^2)$. The fixed point distribution is, by construction, uniform, so all parts of space are explored.

4.2 UMDA and the needle-in-the-haystack problem

The first application of the above results is the so-called needle-in-a-haystack problem. Here the fitness of all strings is 0 except for one special string (the "needle") which has a fitness of 1. Most search algorithms will find the needle if run long enough. It is shown here that UMDA will almost never find the needle unless the population size is exponential in the string length.

Obviously, the algorithm cannot find the needle with an expected time less than 2^{L-1} since that is the expected time of exhaustive search, and exhaustive search is optimal for this problem. Since the search space is flat as long as the needle is not found, the convergence time found above in equation (22) applies and the algorithm will converge in a time of the order of the population size N. Each iteration searches over N states. Thus naively, we might expect that if $N^2 << 2^{L-1}$ the algorithm will converge before it ever samples the needle. Thus, in order for UMDA to solve this problem, the population size has to be $O(2^{L/2})$. This turns out to be nearly the correct scaling.

Let T_N be the time that the needle is first sampled. The shorthand $T_N=\infty$ will mean that the needle is never sampled. A bound can be derived for the probability that the needle is never sampled,

$$\mathrm{prob}\left(T_N=\infty\right) \geq B(N,L). \tag{24}$$

The asymptotics of the bound are found to be given by the following result.

Theorem 1 *In the limit that $L \to \infty$ such that $LN^2 2^{-L} \to 0$, $B(N, L) = 1 - O\left(\frac{N^2 L}{2^L}\right)$.*

Thus, unless the population is exponentially large, the optimum will never be found. The proof of this is found in appendix A. The idea of the proof is this: model the algorithm as random search for some times shorter than some time t. The algorithm cannot be faster than random search. For times longer than t, and for those cases in which the needle has not been found in time t, the fact that the expected variance is small (equation 21) allows us find how close to zero at least one of the probability parameters is. The probability that this component fixates in the next iteration is a lower bound for the probability of never finding the needle. Then t is chosen to make the bound tight as possible.

The derivation suggests that the bound is not tight, and it does seem to get the scaling slightly wrong, by a factor L. Figure 1 shows the result of simple UMDA on the needle problem for different size problems. What is shown is the fitness of the best string in the final population, averaged over many trials. The fitness of the optimal string is 1. The algorithm has been run until either the optimal string is found, or until all of the parameters are within a predetermined distance to fixation. When the population size is small, the needle is found in very few of the trials. Figure 2 shows the same data as in figure 1, but with the population size scaled as $\sqrt{L2^L}$. This shows that the final fitness is a function of $N/\left(\sqrt{L}2^{L/2}\right)$, and so N must scale as $\sqrt{L}2^{L/2}$ for the needle to be found. This is consistent with the lower bound found in theorem 1, which stated that N must be at least as big as $2^{L/2}/\sqrt{L}$ for the needle to be found.

The run time of the algorithm (given the stopping criterion used here) should be of the order of the population size when the needle is not found, and of the order of 2^L when it is. Figure 3 shows the "efficiency" of the algorithm as a function of the population size, as scaled in figure 2. Efficiency is defined as the number of strings processed by the algorithm (number of generations times population size) divided by 2^L. If this is 1.0, the algorithm is efficient, because it has processed 2^L strings. (This is really a measure of inefficiency, because a larger value means *less* efficient, that is, it has processed more than 2^L strings.) To the right of the peaks in the curve, the algorithm is finding the needle, and it is reasonably efficient.

Although the needle-in-a-haystack problem is an artificial problem, it is still disappointing that an algorithm would fail to solve it even if it were to run for an infinite length of time. A sensible search algorithm would find the needle, but would take a long time. UMDA will not find the needle ever, unless a very large population size is chosen.

Detailed balance UMDA always finds the needle. In simulations on the same size problems using the same number of trials and the same stopping criterion, detailed balance UMDA *never* failed to find the needle. This is not surprising; detailed balance UMDA is ergodic. Figure 4 shows the time the algorithm takes to first find the needle. The choice of population size does affect the efficiency of the algorithm, but not whether it works at all.

4.3 UMDA and one-max

It is perhaps not surprising that the results from the flat landscape from section 4.1 predicted the results of the algorithm on the needle-in-the-haystack problem, which is indeed flat almost everywhere. One might also predict that these results will not be relevant for "smoother" problems, and that simple UMDA will work fine on these. To show that this intuition is wrong, we now consider the so-called one-max problem,

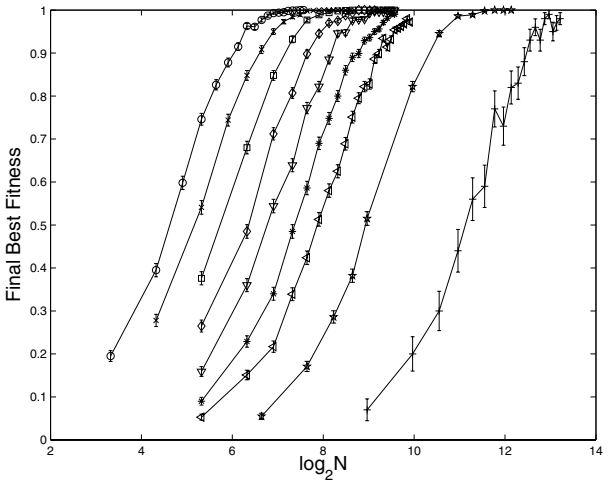

Figure 1: Simulations on UMDA on the needle-in-a-haystack problem for $L = 8, 9, 10, 11, 12, 13, 14, 16, 20$, (respectively $\circ, x, \square, \diamond, \nabla, *, \triangleleft, \star, +$). This shows the fitness of the best string in the final population against the log of the population size. The fitness of the needle is 1. The algorithm is run until the needle is found, or until all parameters are within 5% of fixation, and averaged over 1000 trials, except for $L = 20$ which is averaged over 100 trials. Figure shows fitness of the best member of the population at convergence versus $\log_2 N$. As the system size increases the population size must increase to get the same level of performance.

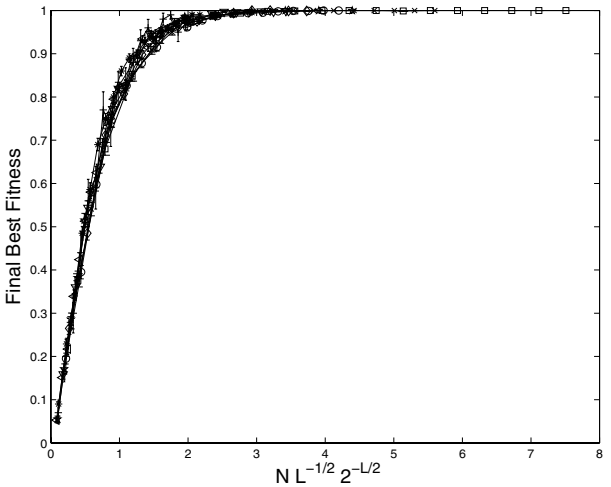

Figure 2: The same data as the previous figure, but with population size N scaled by $\sqrt{L}2^{L/2}$. The data approximately collapses, which shows that as L increases, N must increase by $\sqrt{L}2^{L/2}$ to get the same performance.

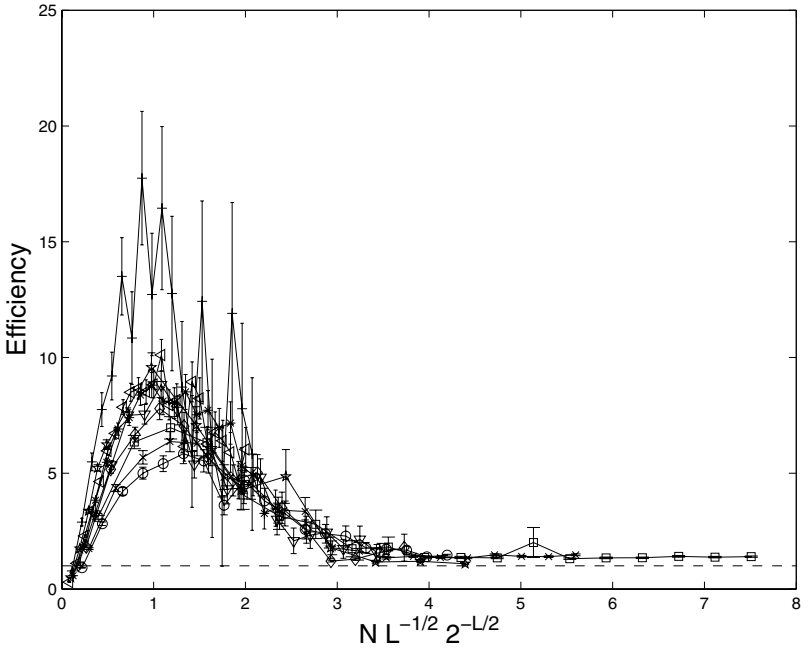

Figure 3: This shows the "efficiency" of the algorithm. The efficiency is the number of strings processed (number of generations run multiplied by population size) divided by 2^L. If the algorithm finds the optimum with an efficiency of 1, it is optimal; larger than 1 is less efficient. Comparison with figure 2 shows that the peak occurs where the algorithm starts to find the optimum.

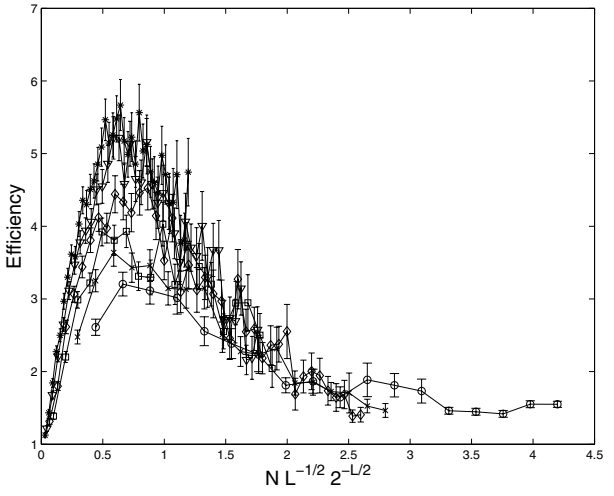

Figure 4: Efficiency of detailed balance UMDA. The needle is always found, the population size determines how long it takes.

which is often thought to be the smoothest and easiest problem for a local search algorithm to solve.

The fitness of any string in this problem is proportional to the string length minus the Hamming distance between it and the single global optimum. The optimum is often taken to be the string of all 1's, in which case the fitness is the number of 1's in the string. (This problem goes under other names: counting ones, multiplicative landscape, etc.) The fitness used here is

$$F(\mathbf{x}) = \frac{1}{L}\sum_{i}^{L} x_i, \tag{25}$$

so the fitness of the global optimum is 1. This problem is very easy for hill-climbing algorithms, because every change which increases the fitness decreases the Hamming distance to the global optimum. Typical hill-climbing algorithms solve this problem in time $O(L \log L)$, which is easily derived.

This problem also contains a large flat subspace, and the performance of standard UMDA will again be dominated by the conflict between convergence on the flat subspace, and movement toward the optimum. To see this, note that any set of parameters γ can be decomposed to a component parallel to the optimum and components perpendicular to that direction. Movement along the perpendicular direction is neutral, it does not change the expected fitness of the sampled strings. Only movement toward or away from the optimum changes the expected fitness. In a high dimensional space (large L) almost all of the space is along neutral directions. The fitness of a random string, such as is generated at the start of the algorithm, is $1/2$ with fluctuations of $O(1/\sqrt{L})$. Thus, initially, movement is in a direction which projects into the neutral subspace $1 - O(1/\sqrt{L})$ and in the direction of the optimum $O(1/\sqrt{L})$.

Thus, there will be convergence due to drift within the neutral subspace at a rate approximately given by the inverse population size (the convergence rate is the inverse characteristic time; see equation (21) and associated discussion). To approximate the rate of convergence toward the optimum, we will use a Gaussian approximation for truncation selection. The effect of truncation selection on a Gaussian with mean m and variance σ^2 is

$$m(t+1) = m(t) + \sqrt{\frac{2}{\pi}}\sigma(t) \tag{26}$$

$$\sigma(t+1)^2 = \sigma(t)^2\left(1 - \frac{2}{\pi}\right). \tag{27}$$

Results like this have been derived in several places (Mühlenbein, 1997; Prügel-Bennett and Shapiro, 1997; MacKay, 2003; Blickle and Thiele, 1995). In these equations, m and σ are the mean and standard deviation in the distribution of fitnesses. The first equation shows that the movement toward the optimum is of order $O(1/\sqrt{L})$ initially, since that is the size of the standard deviation in early stages of the algorithm.

After sampling, the expected mean will remain as given in the above equation. However, sampling will restore the variance to nearly its value in the previous generation, but will be reduced by the factor $1 - 1/N$. Of course, the variance after sampling is given by,

$$\sigma^2(t+1) = \frac{1}{L^2}\sum_{i}\gamma_i(t+1)\left[1 - \gamma_i(t+1)\right](1 - 1/N). \tag{28}$$

If we assume that the excess of ones is distributed independently of the sites, each $\gamma_i(t+1)$ will be approximated by $\gamma_i(t) + \sqrt{2/\pi}\sigma(t)/L$. Again, restricting consideration to early in the algorithm, when the standard deviation is $O(1/\sqrt{L})$ and $\gamma_i = 1/2 + O(1/\sqrt{L})$,

$$\sigma^2(t+1) = \left[\sigma^2(t) + O(1/L)\right](1 - 1/N). \tag{29}$$

Assuming that the population size $N << \sqrt{L}$ and that L is large, the reduction in the variance will be solely due to the finite population sampling reduction factor $(1-1/N)$.

Using this approximation, the equations to iterate are,

$$m(t+1) = m(t) + \sqrt{\frac{2}{\pi}}\sigma(t) \tag{30}$$

$$\sigma(t+1)^2 = \sigma(t)^2\left(1 - \frac{1}{N}\right). \tag{31}$$

Using the fact that the initial variance is $1/(4L)$ and the initial mean is $1/2$, the expected mean fitness at convergence is

$$m(\infty) = \frac{1}{2} + \sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{4L}}\sum_{t=0}^{\infty}\left(1 - \frac{1}{N}\right)^{t/2}, \tag{32}$$

$$= \frac{1}{2} + \sqrt{\frac{1}{2\pi L}}\left(\frac{1}{1 - \sqrt{1 - \frac{1}{N}}}\right) \tag{33}$$

$$\approx \frac{1}{2} + \sqrt{\frac{2N^2}{\pi L}}. \tag{34}$$

The last approximation assumes that N is large, although we continue to assume that $N/\sqrt{L}$ is small. I claim that this (without the large N approximation) is an upper bound on the expected mean fitness. It is an upper bound because: 1) the convergence of the expected variance cannot be slower than on a flat space, and will in general be faster; and 2) the Gaussian approximation over-estimates the change in the mean over that of the true binomial distribution. This becomes especially important when the mean approaches 1. It is the second point which makes it a claim, it has not been shown rigorously.

This shows that if the population is small compared to the square root of the string length, the mean fitness will not change much before the parameters converge. Thus, the optimal string is very unlikely to be sampled. Only if the population size is large compared to the $\sqrt{L}$ will UMDA find the optimal string. Although this is a smooth fitness function, there is a still convergence along the neutral directions orthogonal to the direction to the optimum.

Simulations confirm the scaling given in equation (32). The algorithm does not find the global optimum unless the population size is $O(\sqrt{L})$ as shown in figure 5. In fact the best fitness in the final population is a function of $N/\sqrt{L}$ as is shown in figure 6. The efficiency of the algorithm is show in figure 7 which shows that the algorithm processes $\sqrt{L}N$ strings before reaching the global optimum, if N is large enough that it does reach the optimum.

In contrast, detailed balance finds the optimum for any population size tried in the experiments. The population size *does* determine the efficiency of the algorithm. As

the population size decreases, the rejection probability increases (see equation 5) and the run time increases. In the extreme case that the population size approaches $N = 2$, the algorithm approaches random search, since the size of change and therefore the rejection rate is large. For this reason, ordinary UMDA with a large population size is more efficient than detailed balance UMDA with a small one.

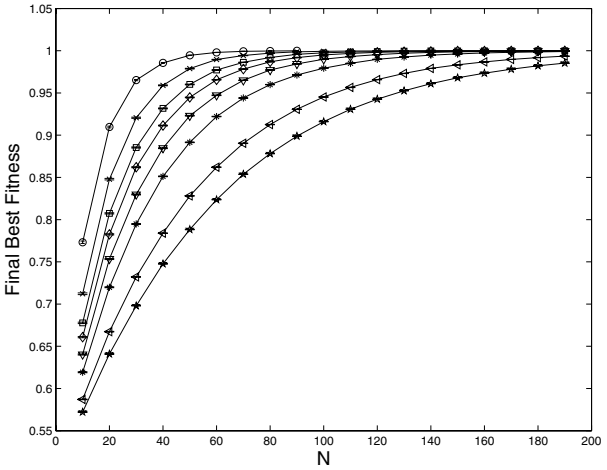

Figure 5: Simulations on UMDA on the one-max problem for $L = 64, 128, 196, 256, 352, 1024, 1536$, (respectively $\circ, x, \square, \diamond, \nabla, *, \triangleleft, \star$, and hexagon). The algorithm is run until the optimum is found, or until all parameters are within 5% of fixation, and averaged over 1000 trials. This figure shows fitness of the best member of the population at convergence versus N. As the system size increases the population size must increase to get the same level of performance.

5 Conclusions and Future Work

There is a flaw in EDA dynamics which makes setting algorithmic parameters strongly problem dependent. We have argued that this is the case formally, by considering the transition probability between parameters of the probability function. We have argued that the underlying Markov process is dominated by absorbing subspaces which cannot communicate with other spaces of the system. These absorbing subspaces can be attractors, because the probability to move toward them is greater than the probability to move away on a flat space. This can be avoided in non-flat fitness landscapes only by choosing algorithm parameters so that movement toward the optimum dominates over movement toward the other absorbing states. This is problem dependent, and gives rise to very poor scaling behaviour of the algorithms in principle.

We have illustrated this behaviour in UMDA, in which this poor scaling is very pronounced. The required population size to find the optimum can be a very low power of the system size or exponentially large in the system size. This means to get UMDA to work, much experimentation must be done to set the appropriate population size. This had previously been shown to occur in PBIL, where the learning rate had to be set smaller than a problem-dependent size which could be a similar low power or exponential. We have shown that detailed balance can go some way to improving the scaling, in that the detailed balance algorithm does find the optimum for any popula-

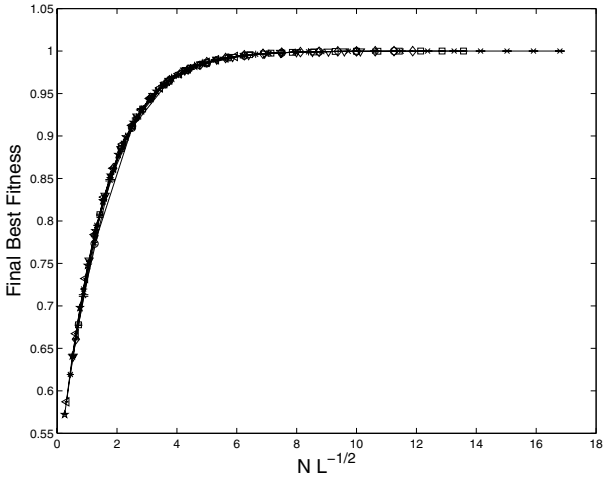

Figure 6: As previous, but with population size N scaled by $\sqrt{L}$. The data approximately collapses, which shows that as L increases, N must increase by $\sqrt{L}$ to get the same performance.

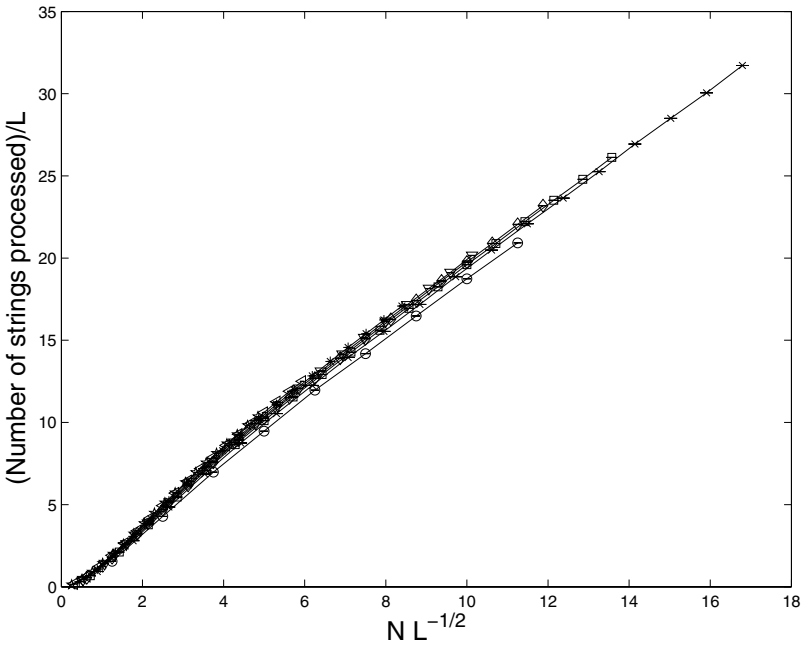

Figure 7: This shows the "efficiency" of the algorithm. The efficiency is the number of strings processed (number of generations run multiplied by population size) divided by L. This shows that the number of iterations required becomes constant, and the computation time is proportion to N. The run time is scaling as approximately as L.

tion size, but the efficiency time does depend on the population size. Other methods such as Bayesian mutation will also mitigate against this effect, although it will still affect the scaling of any parameters that other methods introduce.

One question is, how important is this issue for more complex models that learn dependencies? The answer is not known at this time. Since the arguments in given in section 2 are general, the phenomenon must exist in these models, but the scaling could be more benign. If it is important, some interesting questions arise. Can the scaling be elucidated, and can methods like detailed balance be used?

One argument suggests that it might be less important. These complex models have more parameters, and as the new dependencies are found, new parameters can be generated and old ones unused. Thus, there is much more variability in these models, and each parameter may change less with each iteration. Thus, the phenomenon, though present, could act more slowly and be less important than in the simple models studied here. If that is the case, it raises the question, is this the reason why complex models outperform simple ones in certain studies?

On the other hand, there is an argument that the effect will be *more pronounced* in complex models, at least in the case of models with fixed structure. The effect is due to finite sampling fluctuations, and it increases as the population size gets smaller. For complex models, a smaller fraction of the population contributes to each parameter. So effectively, there is a smaller population size for each parameter. For example, a typical parameter in UMDA might be $P(x_1 = 1)$ and every member of the population contributes to estimating its value. A typical parameter in a complex model might be $P(x_1 = 1|x_3 = 0, x_2 = 1, x_5 = 0)$. Only the members of the population with $x_3 = 0$, $x_2 = 1$, and $x_5 = 0$ contribute to the estimation of its value. This will typically be a fraction of the population. For the estimation of this parameter, finite population effects will be greater than that of a root variable with no dependencies. In general, the question about the role of the phenomenon identified here on general EDAs is still open.

A Proof of theorem 1

In section 4.2 it was claimed that the probability of never sampling the needle by UMDA is bounded from below by a function $B(L, N)$. As $L \to \infty$ such that $N^2 L/2^L \to 0$, $B(L, N) = 1 - O(N^2 L/2^L)$.

Proof: Let T_N be the time it takes to find the needle. By convention, $T_N = \infty$ means that the needle is never found. Run the algorithm for t iterations, where t will be chosen later. Obviously the probability of never finding the needle decomposes into,

$$\text{prob}\,(T_N = \infty) = \text{prob}\,(T_N = \infty | T_N > t)\,\text{prob}\,(T_N > t). \tag{35}$$

Since random search is optimal for this problem, the probability that UMDA does not find the needle in time t is greater than $\left(1 - 2^{-L}\right)^{tN}$. The N is because time is measured in the number of iterations of the algorithm; in each iteration, N strings are considered.

$$\text{prob}\,(T_N > t) \geq \left(1 - 2^{-L}\right)^{tN}. \tag{36}$$

To compute $\text{prob}\,(T_N = \infty | T_N > t)$ first observe that at time t, the expected variance is given by equation (21) (since the needle was never sampled, the space is effectively flat). Due to the fact that the variance is positive, the largest the true variance can be with a probability greater than or equal to $1 - \epsilon$ is $\langle v \rangle / \epsilon$, for any ϵ

between O and 1 (see, for example, (Motwani and Raghavan, 1995, page 46)). In other words,

$$\text{prob}\left[v(t) \leq \frac{\langle v(r)\rangle}{\epsilon}\right] \geq 1-\epsilon. \tag{37}$$

Let $f(v)$ be any probability function which is monotonically increasing as $v(t)$ decreases. Then $f(v) \geq f(\langle v\rangle/\epsilon)$ with probability greater than or equal to $1-\epsilon$. Thus, $f(v) \geq (1-\epsilon)f(\langle v\rangle/\epsilon)$. Applying this to the probability of not finding the needle yields,

$$\text{prob}\left(T_N = \infty | T_N > t\right) > (1-\epsilon)\left[1 - 2\frac{\langle v(t)\rangle}{\epsilon}L\right]^N. \tag{38}$$

Since the variance is small, all of the γ's are close to zero or close to one. Those γ's which are close to 1 have little effect on the probability of finding the needle. To make the probability of finding the needle as large as possible, the γ's close to 0 must be made as large as possible. That is done by setting all of those parameters close to 1 equal to 1. Let $\mathcal{N}$ be the number of probability parameters close to 1. Then,

$$v = \frac{1}{L}\sum_{i=\mathcal{N}+1}^{L} \gamma_i\left(1-\gamma_i\right), \tag{39}$$

which implies that

$$\gamma_i = \frac{1}{2} - \frac{\sqrt{1-\frac{4v}{\tilde{L}}}}{2}, \tag{40}$$

where $\tilde{L} = \left(L-\mathcal{N}\right)/L$. This gives upper bounds for the γ's. The expressions can be simplified at the expense of the bound tightness if we use

$$\sqrt{1-\Delta} > 1-\Delta; \text{ for } 0<\Delta<1, \tag{41}$$

and assume that only one γ is close to zero ($\mathcal{N} = L-1$). Of course, it much more likely that about half of the γ's are near 0; considering these more accurately would tighten the bound. From these approximations it follows that

$$\gamma_1 < 2vL. \tag{42}$$

for that γ close to 0.

Because γ is very small, it is possible that it will be zero after the next iteration, that is, it could *fixate* at 0 in the next iteration. The probability of doing that is $(1-\gamma)^N$. The probability of this component ever fixating is at least as great as this, and once a component fixates the needle cannot be found. The probability of never finding the needle is at least as great as the probability that at least one component fixates. Thus,

$$\text{prob}\left(T_N = \infty | T_N\right) > (1-2vL)^N \tag{43}$$

Putting equations (35) (36) (38) and (43) together yields,

$$\text{prob}\left(T_N > \infty\right) > (1-\epsilon)\left[1 - L\frac{1}{2\left(\epsilon\right)}\left(1-\frac{1}{N}\right)^t\right]^N \left\{\left(1-2^{-L}\right)^{tN}\right\}. \tag{44}$$

The above equation is true for all t. To get the greatest bound, maximise it with respect to t. The value of t which maximises this satisfies,

$$\frac{L}{2\epsilon}\left(1-\frac{1}{N}\right)^t = \frac{\log\left(1-2^{-L}\right)}{\log\left(1-2^{-L}\right)+\log\left(1-N^{-1}\right)}. \tag{45}$$

The solution to this is,

$$t(\epsilon) \approx -\frac{L\log 2+\log\left(\frac{L}{2\epsilon}\right)+\log\left(-\mathcal{L}\right)}{\mathcal{L}}, \tag{46}$$

where $\mathcal{L}$ is $\log\left(1-1/N\right)$. To get bounds, we need the following two inequalities which hold for all $N>0$,

- $\log\left(1-1/N\right) < -1/N$;
- $\log\left(1-1/N\right) \geq -1/N - 1/[2N(N-1)]$.

Consider each factor equation (44) in turn. The first factor is $1-\epsilon$; the size of ϵ will be set at the end. The second factor is

$$\begin{aligned}\frac{L}{2\epsilon}\left(1-\frac{1}{N}\right)^t &\approx \frac{2^{-L}}{\mathcal{L}} &(47)\\ &\leq \frac{N2^{-L}}{1+\frac{1}{2(N-1)}} &(48)\end{aligned}$$

from the second inequality above. Thus,

$$\begin{aligned}\left[1-\frac{L}{2\epsilon}\left(1-\frac{1}{N}\right)^t\right]^N &\geq \left[1-\frac{N2^L}{1+\frac{1}{2(N-1)}}\right]^N &(49)\\ &\approx 1-O(N^2 2^{-L}). &(50)\end{aligned}$$

The third factor is $\left(1-2^{-L}\right)^{tN} \approx 1-tN2^{-L}$. We need to bound t. Using the inequalities as above,

$$t \leq \left[NL\log 2+N\log\left(\frac{L}{2\epsilon}\right)+N\log N\right]\left[1+\frac{1}{2(N-1)}\right]^{-1}. \tag{51}$$

As long as $\log\epsilon$ is not exponential in L, the upper bound on t is $O(NL)$. Therefore,

$$\left(1-2^{-L}\right)^t \geq 1-O(N^2L2^{-L}). \tag{52}$$

Since we can take ϵ smaller than N^2L2^{-L}, the result is shown.

□

B The transition probability on a flat landscape

The Gaussian approximation for the transition probability, equation (15), shows that the transitions away from $\gamma=1/2$ are more likely than those back. This approximation does not hold close to the corners of the hypercube, however. Here it is shown that this is true near the hypercube corners, at least for sufficiently large N.

Consider a single component of the model, and let $n = \gamma N$. Define

$$a(n, n') = \frac{T_{\mathcal{R}}(n'|n)}{T_{\mathcal{R}}(n|n')}, \tag{53}$$

which is the ratio of the probability of the transision from n to n' to the probability of the transition back. The point is to show that if $n < N/2$ and $n' < n$, this ratio is greater than 1. We will derive an approximation for this and show that under this approximation for sufficiently large N, $a(n, n') > 1$.

Using equation (15) a recursion relation can be found,

$$a(n+1, n') = a(n, n') \left(\frac{N-n'}{N-n}\right) \left(\frac{n+1}{n'}\right) \left(\frac{n+1}{n}\right)^{n'} \left(1 - \frac{1}{N-n}\right)^{N-n'}, \tag{54}$$

with the obvious base case $a(n', n') = 1$. Write the recursion as $a(n+1, n') = f_n a(n, n')$. The goal is to show that for $N/2 > n > n'$, $f_n > 1$.

The inequalities used in the previous appendix will be used here to find that

$$\left(1 - \frac{1}{N-n}\right)^{N-n'} > \exp\left[-1 - \frac{n-n'}{N-n} - \frac{1}{2(N-n-1)} - \frac{n-n'}{2(N-n)(N-n-1)}\right]. \tag{55}$$

For sufficiently large N the last two terms in the exponential will be very small and can be ignored.

Firstly, it is obvious that $a(n, 0) = \infty$ because $n' = 0$ is an absorbing state. In addition,

$$a(2,1) > \left(\frac{N-1}{N-2}\right) \frac{9}{2} \exp\left[-1 - \frac{1}{N-n} - \frac{1}{2(N-3)} - \frac{1}{2(N-2)(N-3)}\right] \tag{56}$$

which is greater than 1 for any N greater than 5. (For $N < 5, a(2,1)$ may be greater than 1, but this inequality is not tight enough to show this.)

To show that $a(n, n')$ is greater than 1 for other values, differentiate $\log a(n, n')$ with respect to n, and use inequality (55). This yields,

$$\frac{d \log f_n}{dn} = \frac{n-n'}{n^2+n} - \frac{n-n'}{(N-n)^2} - C(N). \tag{57}$$

Since $a(n, n') = \prod_{k=n'}^{n} f_k$, if this is positive, $a(n, n')$ will increase by an increasingly large factor as n increases. The sum of the first two terms is positive for any $n < N/2-1$, and is very positive as n gets much less than $N/2$. Thus, if the correction term $C(N)$ is ignored, this shows that $a(n, n')$ is an increasing function of n and thus would be bigger than 1 for $N/2 > n > n'$. The correction is, $O(1/N^2)$. Thus, for N sufficiently large, $a(n, n')$ will still be an increasing function, and the result will hold.

As an example, consider $n <= \sqrt(N)$ which is a regime where the Gaussian approximation (equation (15)) breaks down. In this regime,

$$\frac{d \log f_n}{dn} = \frac{n-n'}{N+\sqrt{N}} - O(1/N^2) > 1. \tag{58}$$

For all values of $n > n'$ in this regime, transitions from n to n' occur with higher probability than the back transitions.

By symmetry, similar results will hold for $N/2 < n < n'$.

References

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competive learning. Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University.

Baluja, S. (1997). Genetic algorithms and explicit search statistics. In M.C.Mozer, Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 319–325. MIT Press.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland.

Boas, M. L. (1983). *Mathematical Methods in the Physical Sciences, Second Edition*. Wiley Publishing.

Correa, E. S. and Shapiro, J. L. (2004). A study of the effect of detailed balance on PBIL in k-sat and the p-median problem. In Lotfi, A. and Garibaldi, J. M., editors, *Applications and Science in Soft Computing*, Advances in Soft Computing, pages 255–262. Springer. ISBN: 3-540-40856-8.

Gonzalez, C., Lozano, J., and Larrañaga, P. (2001a). Analyzing the population based incremental learning algorithm by means of discrete dynamical systems. *Complex Systems*, 12(4):465–479.

Gonzalez, C., Lozano, J., and Larrañaga, P. (2001b). The convergence behaviour of the PBIL algorithm: a preliminary approach. In *Proceedings of Fifth International Conference on Neural Networks and Genetic Algorithms, ICANNGA 2001*, pages 228–231.

Gonzalez, C., Lozano, J., and Larrañaga, P. (2002). Mathematical modelling of umcac algorithm with tournament selection: Behaviour on linear and quadratic functions. *International Journal of Approximate Reasoning*, 31(3):313–340.

Höhfeld, M. and Rudolph, G. (1997). Towards a theory of population-based incremental learning. In *Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, volume 1, pages 1–5.

Johnson, A. and Shapiro, J. L. (2001). The importance of selection mechanisms in distribution estimation algorithms. In *Proceedings of the 5th International Conference on Artificial Evolution AE01*.

Larrañaga, P. and Lozano, J. A. (2002). *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

MacKay, D. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. Chapter 19.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. Cambridge University Press.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto. http://www.cs.toronto.edu/ radford/review.abstract.html.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023.

Pelikan, M., Goldberg, D. E., and Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20. Also IlliGAL Report No. 99018.

Prügel-Bennett, A. (2003). Modelling finite populations. In Jong, K. A. D., Poli, R., and Rowe, J. E., editors, *Foundations of Genetic Algorithms 7*, pages 99–114. Morgan Kaufmann.

Prügel-Bennett, A. and Shapiro, J. L. (1997). The dynamics of a genetic algorithm for simple Ising systems. *Physica D*, 104:75–114.

Shapiro, J. L. (2003a). Scaling of probability-based optimization algorithms. In Obermayer, K., editor, *Advances in Neural Information Processing Systems 15*, pages 399–406. MIT Press.

Shapiro, J. L. (2003b). The sensitivity of pbil to its learning rate and how detailed balance can remove it. In Jong, K. A. D., Poli, R., and Rowe, J. E., editors, *Foundations of Genetic Algorithms 7*, pages 115–132. Morgan Kaufmann.

Tanner, M. A. (1996). *Tools for Statistical Inference*. Springer Series in Statistics. Springer, New York, third edition.