Wise Breeding GA via Machine Learning Techniques for Function Optimization

Xavier Llorà and David E. Goldberg

Illinois Genetic Algorithms Laboratory (IlliGAL),
National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign,
104 S. Mathews Avenue, Urbana, IL 61801.
{xllora,deg}@illigal.ge.uiuc.edu

Abstract. This paper explores how inductive machine learning can guide the breeding process of evolutionary algorithms for black-box function optimization. In particular, decision trees are used to identify the underlying characteristics of good and bad individuals, using the mined knowledge for wise breeding purposes. Inductive learning is complemented with statistical learning in order to define the breeding process. The proposed evolutionary process optimizes the fitness function in a dual manner, both maximizing and minimizing it. The paper also summarize some tuning and population sizing issues, as well as some preliminary results obtained using the proposed algorithm.

1 Introduction

Recently, a new interest in the genetic algorithms (GA) community has been growing. The work published by Baluja [1,2], Juels & Wattenberg [3], and Mühlenbein & Paaß [4]—among others—sparked a new way to approach to GA. Instead of recombining genes, as in a traditional GA, this new approach proposes the usage of explicit statistics as the main breeding force. These kind of GA are known as probabilistic model building GA (PMBGA), or estimation of distribution algorithms (EDAs) [5]. Instead of using crossover or mutation operators, these GA breed a new population of individuals sampling a learned probabilistic model that describes the good individuals in the population.

Some early efforts done in PMBGA assume that the probability distribution of each gene can be computed independently. This assumption, not true in many real-world problems, leads to some well-know algorithms. Some examples of algorithms that assume gene independence are: PBIL (*Population Based Incremental Learning*) [1], UMDA (*Univariate Marginal Distribution Algorithms*) [6], and the cGA (*compact Genetic Algorithm*) [7]. Dependences among genes, and what that implies to the distributions to be learned, have also been studied by several authors. Relevant work deals with gene dependence modeling and linkage learning. For an overview of these approaches please see [5,8]. An example of this kind of algorithms is BOA (*Bayesian Optimization Algorithm*) [9]. BOA bases its probability distribution on a Bayesian network. This network describes

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 1172–1183, 2003.

the probability distribution of the good individuals in the population, as well as the dependences among genes.

There are other non-statistical approaches to model building GA for breeding purposes. Some of them are provided by inductive machine learning techniques. An example of this kind of approach is LEM (*Learnable Evolution Model*) proposed by Michalski [10,11]. LEM combines two different learning paradigms, evolutionary learning and inductive rule learning. On the one hand, the evolutionary learning uses a simple GA for function optimization. On the other hand, LEM1 and LEM2 uses AQ15 [12] and AQ18 [13] for rule learning. However, LEM is a hybrid approach that still relies on traditional GA mechanisms, although they can be turned off, moving it away from the ideas that inspired PMBGA and EDAs.

This paper presents an alternative approach. The work combines statistical models and inductive machine learning for guiding the breeding process. The goal is to create an algorithm that exploits the knowledge that can be inferred from the current population. In order to mine the population looking for useful breeding knowledge, we combine two different machine learning methods under a supervised learning paradigm. This is achieved casting the optimization task into a problem that can be solved using machine learning techniques. The goal is boosting the evolutionary search process using the mined knowledge. This population based learning algorithm, SI3E (*statistical and inductive tree based evolution*), is applied to some optimization problems, showing its competence, as well as some interesting evolutionary dynamics, optimizing the fitness function in a dual manner, both maximizing and minimizing it. Moreover, casting the optimization into an inductive learning problem provides an elegant way for population sizing of the proposed algorithm.

The paper is structured as follows. Section 2 reviews some background, needed for understanding SI3E. Then, section 3 describes the evolutionary model proposed by SI3E. This section also discusses the parameter tuning and population sizing for a competent usage of the algorithm. Section 4 summarizes the preliminary results obtained using the algorithm on two different well-known black-box optimization functions. Finally, section 5 presents some conclusions about the work presented in this paper.

2 Background

The goal of the work presented in this paper is to combine the ideas that inspired PMBGA, EDAs, and inductive machine learning. SI3E (*statistical and inductive tree based evolution*) is the result of mixing a PMBGA and inductive decision trees. Properly speaking, SI3E defines a common framework based on PBIL [1] and ID3 [14,15]. This section describes both algorithms briefly, PBIL and ID3, in order to explain how SI3E integrates both for obtaining a model-based breeding GA in the next section.

2.1 Population Based Incremental Learning (PBIL)

PBIL was introduced by Baluja [1], and was later improved in [2,16]. The goal of PBIL is to solve a black-box optimization problem. The target function is defined over a binary space $\Omega = \{0,1\}^{\ell}$. Thus, each individual in the population can be represented by a binary string. However, PBIL does not maintain the population. Instead, PBIL models the population using a probability vector $p(x)$ of size ℓ:

$$p(x) = (p_t(x_1), p_t(x_2), \ldots, p_t(x_\ell)) \tag{1}$$

where $p_t(x_i)$ refers to the probability of finding a 1 in the i^{th} position of D_t, the population of individuals in the t^{th} generation. Thus, PBIL models the population using the probability vector $p(x)$.

The algorithm samples the distribution defined by $p(x)$ generating m individuals. These individuals are evaluated using the fitness function. Then, the best n individuals ($n \leq m$) are selected. These individuals can be denoted, as shown in [5], by:

$$x^t_{1:m}, \ldots, x^t_{i:m}, \ldots, x^t_{n:m} \tag{2}$$

Then, $p(x)$ is updated using these selected individuals. These update is done using the following rule:

$$p_{t+1}(x) = (1-\alpha)\, p_t(x) + \alpha \frac{1}{n} \sum_{k=1}^{n} x^t_{k:m} \tag{3}$$

where α is a parameter that controls the learning rate.

2.2 Induction of Decision Trees (ID3)

The induction of decision trees was born in the machine learning community. The goal is the learning of a concept from a set of illustrative examples, or supervised learning. The first well-known approach to the induction of decision trees, ID3, was proposed by Quinlan [14]—improved later as C4.5 [15]. ID3 is a tree inducer based on a divide and conquer strategy. Given a set of examples of the target concept, the algorithm picks an attribute for building the root node of the tree. Once the root node is chosen, the data set is divided according to the values of the selected attribute. For each of these divided data sets, the process is repeated recursively creating the branches of the root node. The process stops when all the examples in the data set belong to the same class of the target concept.

The selection of the splitting attribute becomes critical for the induced decision tree. ID3 measures the purity of the split introduced, by picking an attribute, using an *information gain* heuristic. The goal is to minimize the impurity that introduces the split, looking to build a compact decision tree. The *information gain* heuristic is based on the *entropy* measure. The *entropy* of a given set D [17],

relative to a classification task of c classes (generalized version of the concept learning problem from positive and negative examples) is:

$$Entropy(D) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \tag{4}$$

where p_i is the proportion of D belonging to class i. Using the *entropy* measure, the *information gain*, $Gain(D, A)$, of an attribute A is the expected reduction in *entropy* caused by partitioning the examples according to this attribute. The *information gain* is then defined as

$$Gain(D, A) \equiv Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v) \tag{5}$$

where $Values(A)$ is the set of all possible values for attribute A, and D_v is the subset of D for which attribute A has value v (i.e. $D_v = \{d \in D | (A(d) = v\}$).

3 Statistical Learning + Inductive Learning = SI3E

The previous section presented some basic background for SI3E. This section explains how statistical learning and inductive learning can be mixed in order to provide a breeding model for GA. As explained in the introduction, the goal is to create an algorithm that exploits the knowledge that can be mined from the current population. Inductive learning is going to be in charge of this task, however, as we will show later, it has some limitations if it is to become the core of a breeding model for GA. Nevertheless, this limitations can be overcome by combining inductive learning and statistical learning.

3.1 Knowledge Extraction from a Population Using ID3

The work presented in this paper deals with black-box optimization functions defined on a binary space, $\Omega = \{0, 1\}^\ell$. Thus, an individual x of the population is defined by a binary string, $x \in \Omega$. The target function is used as a fitness function, $f : \Omega \mapsto \Re$. Using the fitness function $f(x)$, we can sort the individuals in a population $\mathcal{P}$. Therefore, we know which are the best and worst individuals. As proposed in LEM [11], we mark the best individuals as *positive* examples, $\mathcal{P}^\oplus$. In the same way, we can mark the worst individuals as *negative* examples, $\mathcal{P}^\ominus$. These sets contain the best and worst individuals seen so far in the evolutionary process. This approach differentiates SI3E from PMBGA and EDAs. The main reason for this two sets is the result of casting the optimization problems into a supervised learning problem. Using this approach, we artificially create a data set, $D = \mathcal{P}^\oplus \cup \mathcal{P}^\ominus$, from which we can extract knowledge using any supervised learning algorithm from the *machine learning* literature. All the instances in the data set D are individuals of the population. Therefore, we deal with a binary classification task (*positive* and *negative* examples) where all ℓ attributes

are binary. Then, the goal is to find an explicit representation of the differences between *positive* and *negative* instances—individuals of the population. Since the classification problem to be solved is quite simple, ID3 is suited for inducing a compact tree, leaving more complex approaches (i.e C4.5 [15]) for further study.

Table 1. Turning the population into a data set D for supervised learning. Using D, the rules induced by ID3 are: $\mathcal{R}$={`0***`:⊖, `10**`:⊖, `11**`:⊕}.

Artificially created data set $\mathcal{D}$												
Rank	*Genotype*	$f(x)$	*Class*	*Rank*	*Genotype*	$f(x)$	*Class*	*Rank*	*Genotype*	$f(x)$	*Class*
0	`1111`	4	⊕	4	`1101`	3	not used	8	`0101`	2	⊖
1	`1110`	3	⊕	5	`1110`	3	not used	9	`1010`	2	⊖
2	`1110`	3	⊕	6	`0111`	3	not used	10	`0001`	1	⊖
3	`1110`	3	⊕	7	`1100`	2	not used	11	`0001`	1	⊖

Table 1 shows how a randomly generated population $\mathcal{P}$ can be turned into a data set $\mathcal{D}$ for supervised learning. In this example, the size of the individuals in the population is $\ell = 4$, whereas the population size $|\mathcal{P}| = 12$. The fitness $f(x)$ is computed using the OneMax function [18]. After sorting the population according to $f(x)$, we split the population into three subsets: the best individuals, the worst individuals, and the mediocre ones. We will discuss more about how this split is performed later on. Once we have built $\mathcal{D}$, we used ID3 for inducing a decision tree, extracting the set of equivalent rules. For further details, please see [14,15,17]. In the *condition* part of the rules, `0` or `1` denotes the allele to be used at the given gene, whereas `*` indicates that the gene can take any value (i.e. `0` or `1`). The *class* part of the rules identifies whether the rule describes a characteristic of a good individual (⊕) or a bad one (⊖). We can take the interpretation of these rules one step further. They can be seen as a kind of notation of the underlying *schemas* [19] for good and bad individuals. This point can be assumed if we agree that the knowledge mined by ID3 is actually the *building blocks* of good and bad individuals.

The rules produced by ID3 are the base of the breeding model used by SI3E. However, there is a question that must be answered before using such a thing. The rules produced by ID3 split the binary space of the genotype in a non-overlapping and recursive way. This means that rules cannot be combined in any useful way. Moreover, there are a large number of genes marked as `*` in the rule. This means that if we pick a rule in order to generate a new good individual, the genotype will be under-specified, because the rule is giving no clue about the value to be substituted in place of the `*`. This is where PMBGA can help solve this problem.

3.2 Can PBIL Ideas Help?

The rules obtained using ID3 show the underlying patterns of good and bad individuals. Nevertheless, there are several approaches to fill the empty genes

described by the rules. PBIL provides a simple one using the *positive examples* ($\mathcal{P}^{\oplus}$) and the *negative examples* ($\mathcal{P}^{\ominus}$) sets. Using both data sets we can compute the appearance probability of the alleles, `0` and `1`, for each gene. Then, given a rule produced by ID3, we can fill the unspecified positions probabilistically.

For choosing an allele for the empty genes we compute two probability vectors, $p(x|\mathcal{P}^{\oplus})$ and $p(x|\mathcal{P}^{\ominus})$, using the *positive* and *negative* examples data sets. Given an example data set $\mathcal{S}$, the probability vector $p(x|\mathcal{S})$ of size ℓ is

$$p\left(x|\mathcal{S}\right) = \left(p\left(x_1|\mathcal{S}\right), p\left(x_2|\mathcal{S}\right), \ldots, p\left(x_\ell|\mathcal{S}\right)\right) \tag{6}$$

where $p(x_i|\mathcal{S})$ refers to the probability of finding a `1` in the i^{th} variable of $\mathcal{S}$.

Thus, combining the rules with the statistical information, we can generate new good and bad individuals. However, there are still two issues to solve before we can use this approach for breeding new individuals. They are explained in the next subsection.

3.3 Fixed Loci and Example Set Formation

Before showing the algorithmic description of SI3E, we present the last two remaining issues to solve. The first one arises from the appearance of genes with fixed values across $\mathcal{P}^{\oplus}$ and $\mathcal{P}^{\ominus}$. The second is the formation of the data sets, $\mathcal{P}^{\oplus}$ and $\mathcal{P}^{\ominus}$, using the current population $\mathcal{P}$.

A fixed locus happens when for all the instances of a data set $\mathcal{S}$, the same allele appears on a given gene. This property can be expressed as

$$fixed(x_i, \mathcal{S}) \Leftrightarrow \mathcal{S}^{j}_{x_i} = \mathcal{S}^{j+1}_{x_i}, \forall j = 1, 2, \ldots, |\mathcal{S}| - 1 \tag{7}$$

where x_i is the i^{th} gene, $\mathcal{S}$ the available data set, and $\mathcal{S}^j$ the j^{th} instance of the $\mathcal{S}$ data set. The set of fixed locus can be defined as

$$\phi(X, \mathcal{P}^{\oplus}, \mathcal{P}^{\ominus}) = \{x_i \in X | fixed(x_i, \mathcal{P}^{\oplus}) \wedge fixed(x_i, \mathcal{P}^{\ominus}) \wedge \mathcal{P}^{\oplus}_{x_i} \neq \mathcal{P}^{\ominus}_{x_i}\} \tag{8}$$

where $\mathcal{P}^{\oplus}_{x_i}$ is allele of the fixed position x_i in the $\mathcal{P}^{\oplus}$ data set, as well as $\mathcal{P}^{\ominus}_{x_i}$ is the allele of the fixed position x_i in the $\mathcal{P}^{\ominus}$ data set.

Fixed loci mislead ID3 and the *information gain* heuristic. The problem arises when the same gene appears as a fixed locus on both $\mathcal{P}^{\oplus}$ and $\mathcal{P}^{\ominus}$ sets, but the fixed locus represent different alleles. An example of a data set D with a fixed locus is: $\mathcal{R}$={ `1100`:$\ominus$, `0001`:$\ominus$, `0110`:$\oplus$, `1111`:$\oplus$ } The gene x_3 contains the allele `0` for all the instances in $\mathcal{P}^{\ominus}$, whereas the allele `1` is fixed in the $\mathcal{P}^{\oplus}$ data set. If we compute the *gain* using the data set $D = \mathcal{P}^{\ominus} \cup \mathcal{P}^{\oplus}$ with the fixed gene x_3, we obtain $Gain(D, x_3) \equiv 1$. $Gain(D, x_3)$ is the maximum gain using the four genes available. Moreover, if we select x_3 and split the data set D, we obtain a perfect description of good and bad individuals in data set $\mathcal{D}$. Therefore, ID3 would stop there and produce two different rules, `**0*`:$\ominus$ and `**1*`:$\oplus$.

Fixed loci turn the breeding process into a PBIL equivalent. Nevertheless, fixed loci do not provide much breeding information, since they stop ID3 from exploring linkages among genes. This situation can be avoided by removing the

fixed loci from the set of available genes to explore. This fact does not imply that these fixed genes are ignored, on the contrary, $p(x|\mathcal{P}^{\oplus})$ and $p(x|\mathcal{P}^{\ominus})$ contains enough breeding information for their correct usage.

Another critical point in SI3E is the formation of $p(x|\mathcal{P}^{\oplus})$ and $p(x|\mathcal{P}^{\ominus})$ data sets. In this paper we use a simple approach suggested by LEM [11]. As shown in table 1, we split the current population $\mathcal{P}$, ordered by fitness, into three equally sized disjoint subsets. The best individuals form $\mathcal{P}^{\oplus}$, the worst ones form $\mathcal{P}^{\ominus}$, and the rest are not used. This process looks for obtaining opposite instances of the concept to be learned (good or bad individuals), removing the regular ones in order to avoid adding extra noise.

However, if we use this splitting policy strictly, we may be introducing some noise. We can see in table 1, that the last best individual of $\mathcal{P}^{\oplus}$ has a fitness value of 3. There are three more individuals that also have this fitness value that were not chosen. The same happens with the last worst individual that form $\mathcal{P}^{\ominus}$ having a fitness value of 2, leaving 1 individuals with the same fitness out of the set. These random choices introduce noise in the learning process of ID3. We softened this splitting policy, adding these equivalent fitness individuals. However, we still maintain that the fitness of the worst individual in $\mathcal{P}^{\oplus}$ should be better than the fitness of the best individual in $\mathcal{P}^{\ominus}$.

3.4 Everything in Place: SI3E Algorithms and Their Tuning

The algorithms that implement the evolutionary process of SI3E are presented in figures 1 and 2. Inspecting the algorithms, it can be seen that there are two parameters for tuning SI3E. These parameters are α, the learning rate, and the population size. In this paper we tune these parameters using previous work. The first one, α, is usually set between [0.1,0.2]. These values are common in the *reinforcement learning* community. Please refer to [17,20] for more information.

The other parameter to tune is the population size. Our population sizing model is based on guarantee successful breeding. That is, we size the population in terms of $\mathcal{P}^{\oplus}$ and $\mathcal{P}^{\ominus}$. Moreover, if we assume the policy presented in section 3.3, the population size $|\mathcal{P}|$ is approximated by $|\mathcal{P}| \approx 3|\mathcal{P}^{\oplus}|$. Thus, the sizing model for $\mathcal{P}$ is obtained determining the sizing model of $|\mathcal{P}^{\oplus}|$. The size of $\mathcal{P}^{\oplus}$, relies on the capabilities of the learning algorithms, ID3 and PBIL. In this paper, we will only focus on the worst case provided by ID3, trying to estimate the lower bound of $|\mathcal{P}^{\oplus}|$. In other words, we need to estimate the minimum size of $\mathcal{P}^{\oplus}$ that leads ID3 to learn a competent description of the target concept (characteristics of good and bad individuals).

Our population sizing problem can be translated into a well-known problem in the *machine learning* community. This problem is to determine the number of instances (samples of the hypotheses space) that a learning algorithm requires for learning the target concept. Haussler [21] compute this lower bound using the theoretical framework proposed by Valiant [22]. This computation relies on the *probably approximately correct* (PAC) model. PAC learners, like ID3, learn target concepts from some concept class C, using training examples drawn at random according to an unknown, but fixed, probability distribution. It requires

```
SI3E(P)
   t ← 0
   initialize P(t), p(x|P⊕)(t), and p(x|P⊖)(t)
   evaluate and sort P(t)
   WHILE ¬ end-criteria-satisfied
   DO
      split P(t) into P⊕ and P⊖
      compute p(x|P⊕)(t+1) and p(x|P⊖)(t+1)
      identify fixed locus φ(X, P⊕, P⊖) using
            p(x|P⊕)(t+1) and p(x|P⊖)(t+1)
      obtain the rule set R using ID3, φ(X, P⊕, P⊖), P⊕, and P⊖
      p(x|P⊖)(t+1) ← (1 − α) · p(x|P⊖)(t)+α · p(x|P⊖)(t+1)
      p(x|P⊕)(t+1) ← (1 − α) · p(x|P⊕)(t)+α · p(x|P⊕)(t+1)
      breed a new population P(t+1) using
            R, p(x|P⊕)(t+1), p(x|P⊖)(t+1), P⊕, and P⊖,t
      t ← t+1
      evaluate and sort P(t)
   DONE
RETURN P
```

Fig. 1. Algorithm implemented by SI3E.

that the learner (with probability at least [1-δ]) learns an hypothesis that is approximately (within error ϵ) correct.

Within the setting of the PAC learning model, any consistent learner using a finite hypothesis space H where $C \subseteq H$ (where $|H| = 2^\ell$ in SI3E), with a probability (1-δ), output a hypothesis within error ϵ of the target concept after observing m randomly drawn training examples, as long as sufficient examples are provided. This bound is computed [21] as

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ell \ln 2\right) \tag{9}$$

Equation 9 suggest an interesting result for SI3E, when analyzed from an asymptotically: m must grow at $O(\ell)$. Therefore, the number of new individuals generated, MAX (see figure 2), should grow linearly to the length ℓ of the individuals in the population; that is, logarithmically to the size of the search space. For further detail about the PAC model, please refer to [22,21,17].

4 Experiments

In this section we present some preliminary results obtained using SI3E. The conducted experiments involved SI3E and two different functions (OneMax, introduced in section 3, and the concatenation of 4-bit deceptive traps, see [18].)

```
Breed(P,R,p(x|P⊕),p(x|P⊖),P⊕,P⊖,t)
   P(t+1) ← P⊕ ∪ P⊖
   i ← 0
   WHILE (i<MAX)
   DO
        draw a rule r from R at random
        IF (r ∈ ⊕)
        THEN
            x ← fill the unspecified possitions of r using p(x|P⊕)
        ELSE
            x ← fill the unspecified possitions of r using p(x|P⊖)
        FI
        P(t+1) ← P(t+1)∪{x}
        i ← i+1
   DONE
RETURN P
```

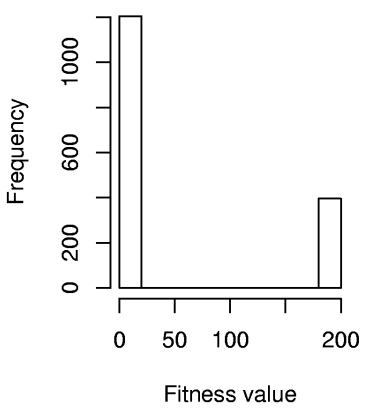
Fig. 2. Breeding algorithm used by SI3E.

These experiments were designed for showing the viability of the approach suggested by SI3E, as well as, for studying its scalability. SI3E was tuned as follows. α was set to 0.2 (please refer to [17,20]). The population size was set using the results presented in the previous section. The population was parameterized as follows: MAX=4ℓ, $|\mathcal{P}^{\oplus}| = \ell$, and $|\mathcal{P}^{\ominus}| = 3\ell$, biasing the population towards the *negative* instances. The reason for this bias is that *negative* usually outnumber the *positive* ones (i.e 4-bit deceptive traps).

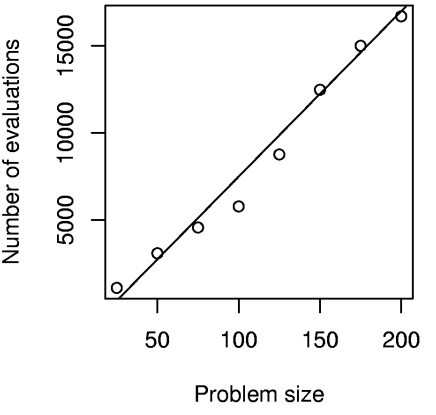
Figures 3(a) and 3(c) shows the distribution of the population evolved by SI3E solving OneMax and the concatenation of 4-bit deceptive traps. As explained in section 3, SI3E is optimizing the fitness function in a dual manner (both maximizing and minimizing it). Thus, the evolved population collapses on two different regions. On the right-hand size, the best ones (highest fitness set $\mathcal{P}^{\oplus}$), whereas on the left-hand size the worst evolved individuals (lowest fitness set $\mathcal{P}^{\ominus}$) can be found. The amount of individuals on each set is the result of the ratio of good and bad individuals (1:3).

We also conducted some preliminary analysis about the scalability of SI3E. Figures 3(b) and 3(d) summarizes the obtained results using SI3E solving the OneMax and the concatenation of 4-bit deceptive traps functions. Each result is the average of 50 independent runs. SI3E scales linearly in the OneMax problem as shown in figure 3(b). These results was not unexpected. PBIL performs in $O(\ell)$ in the OneMax problem. Hence, the building block identification introduced by ID3 do not degrade the overall performance.

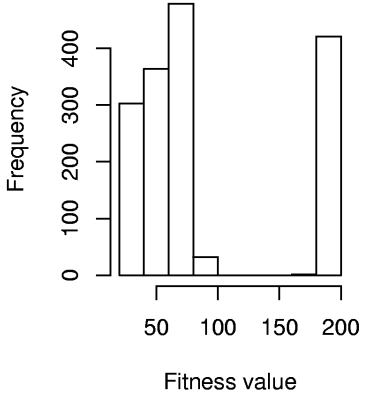
The building block identification, performed by ID3 in SI3E, proves its usefulness when solving the concatenation of 4-bit deceptive traps. As mentioned before, this deceptive function has a clear underlying patterns. Thus, SI3E can



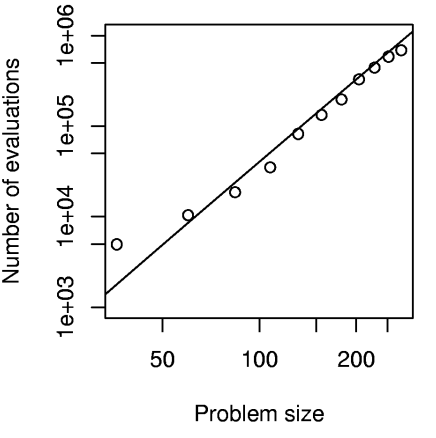
(a) Final population distribution for OneMax (ℓ=200).



(b) Growth for OneMax ($O(\ell)$). Figure plotted using linear axes.



(c) Final population distribution for 4-bit *deceptive trap* (ℓ=200).



(d) Growth for 4-bit traps ($O(\ell^3)$). Figure plotted using logarithmic axes.

Fig. 3. Results obtained using SI3E. Figures show the results solving two different optimization functions.

learn these underlying patterns, being later exploited in the breeding phase of the algorithm. Theoretical studies show that simple GA scales exponentially

with the length of individuals when solving deceptive trap functions, while competent GA, like BOA [8], achieve this goal in subquadratic time [18]. Figure 3(d) summarizes the results obtained. Results suggested polynomial scalability of the number of evaluations related to the problem size. Empirical results suggested, at most, $O(\ell^3)$ scalability. Although this boundary is one order of magnitude bigger than well-known competent GA, these preliminary results need a deeper analysis. In particular, the population sizing criteria adopted and the artificial $\mathcal{D}$ data set formation criteria should be studied more deeply.

5 Conclusions

This paper has explored the usage of inductive and statistical learning for improving the breeding process of GA. The paper relies on the existence of learnable patterns in the population. These patterns are usually the result of the existing regularities introduced by the fitness function. Thus, using the fitness function, the individuals can be ranked and split in two different subsets. The first one contains the best individuals seen so far in the evolution. Similarly, the second one is formed by the worst individuals seen so far. As a result of this kind of splitting of the population, it can be mined using inductive and statistical learning, looking for the underlying patterns that describe both sets.

SI3E implements these ideas. Using a sorted population, where the best and worst individuals have been identified, SI3E uses ID3, as well as some PBIL ideas, obtaining the underlying existing schemas in the population. These schemes are used later to produce a new population of good and bad individuals. Thus, the breeding phases is wisely guided by these mined patterns. Preliminary results obtained using SI3E suggest the competence of this machine learning based GA for function optimization. This fact is also suggested by the empirical analysis of the scalability of the algorithm, although it should be studied deeply, as summarized in section 4.

Acknowledgments. This work was supported by the Technology Research, Education and Commercialization Center (TRECC), a program of the University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research (N00014-01-1-0175). Funding was also provided the Air Force Office of Scientific Research, USAF, (F49620-00-0163), and the National Science Foundation (DMI-9908252).

References

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

2. Baluja, S.: An empirical comparation of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University (1995)
3. Juels, A., Wattenberg, M.: Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical Report CSD-94-834, Computers Science Department, University of California at Berkeley, USA (1995)
4. Mühlenbein, H., Paaß, G.: From recombination of genes to estimation of distributions I. In: Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV, Springer (1996) 178–187
5. Larrañaga, P., Lozano, J.: Estimation of Distribution Algorithms. GENA 2. Kluwer Academic Publishers (2002)
6. Mühlenbein, H.: The equation for response to selection and its use for prediction. Evolutionary Computation **5** (1998) 303–346
7. Harik, G., Lobo, F., Goldberg, D.: The compact genetic algorithm. In: Proceedings of the IEEE Conferrence on Evolutionary Computation, IEEE press (1998) 523–528
8. Pelikan, M.: Bayesian Optimization Algorithm: from single level to hierarchy. PhD thesis, University of Illinois at Urbana Champaign, Urbana, Illinois, USA (June, 2002)
9. Pelikan, M., Goldberg, D., Cantú-Paz, E.: BOA: the Bayesian Optimization Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), Morgan Kaufmann (1999) 525–532
10. Michalski, R.: Learnable Evolution: Combining Symbolic and Evolutionary Learning. In: Proceedings of the 4th International Workshop on Multistrategy Learning, Decenzano del Garda, Italy. (June 11-14, 1998)
11. Michalski, R.: Learnable Evolution Model: Evolutionary Processes Guided by Machine Learning. Machine Learning **38** (2000) 9–40
12. Wnek, J., Kaufman, K., Bloedorn, E., Michalski, R.: Inductive Learning System AQ15c: The Method and User's Guide. Reports of the Machine Learning and Inference Laboratory, MLI 95-4, George Manson University, Fairfax, VA (March, 1995)
13. Kaufman, K., Michalski, R.: The AQ18 Machine Learning and Data Mining System: An Implementation and User's Guide. Reports of the Machine Learning and Inference Laboratory, George Manson University, Fairfax, VA (1999)
14. Quinlan, J.R.: Induction of decision trees. Machine Learning **1** (1986) 81–106
15. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann (1993)
16. Baluja, S., Caruana, R.: Removing the genetics from standard genetic algorithms. In: International Conference on Machine Learning, Morgan Kaufmann (1995) 38–46
17. Mitchell, T.M.: Machine Learning. McGraw Hill (1997)
18. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers (2002)
19. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press/Bradford Books edition (1975)
20. Gonzalez, C., Lozano, J., Larranarraga, P.: Analyzing the pbil algorithm by means of discrete dynamical systems. Complex Systems **12** (2001) 465–479
21. Haussler, D.: Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. Artificial Intelligence **36** (1988) 177–221
22. Valiant, L.: A theory of the learnable. Communications of ACM **27** (1984) 1134–1142