The Sampling-and-Learning Framework:
A Statistical View of Evolutionary Algorithms

Yang Yu*, Hong Qian

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China

Abstract

Evolutionary algorithms (EAs), a large class of general purpose optimization algorithms inspired from natural
phenomena, are widely used in various industrial optimizations and often show excellent performance. This
paper presents an attempt towards revealing their general power from a statistical view of EAs. We summarize a
large range of EAs into the sampling-and-learning framework. The framework directly admits a general analysis
on the probable-absolute-approximate (PAA) query complexity. We study the framework with the learning
subroutine being restricted as binary classifications, which results in the sampling-and-classification (SAC)
algorithms. With the learning theory results, we give a general upper bound on the PAA query complexity of SAC
algorithms. We further compare SAC algorithms with the uniform search in different situations. Under the error-
target independence condition, we show that firstly, when a problem costs the uniform search a super-polynomial
PAA query complexity, it can also cost a SAC algorithm a super-polynomial complexity; but secondly, SAC
algorithms can achieve a polynomial reduction of the complexity of the uniform search. Under the error-
successive-level independence condition, we show that a super-polynomial reduction of the complexity can be
achieved. This work only touches the surface of the sample-and-learning framework, of which the power under
other conditions is still open.

Key words: Evolutionary Algorithms, Computational Complexity of Algorithms, Stochastic Optimization,
Heuristic Search

1. Introduction

In many practical optimization problems, the objective functions are hidden or too complicated to be analyzed.
Under this kind of circumstances, direct optimization algorithms are appealing, which follow the trial-and-
error style using some heuristics. Evolutionary algorithms (EAs) are a large family of such algorithms,
including genetic algorithms [17], evolutionary programming [28], evolutionary strategies [6], particle swarm optimization [27], ant colony optimization [12], estimation of distribution algorithms [32], etc. EAs share the trial-and-error structure while incorporating heuristics inspired from different nature phenomena. Numerous real-world applications have been reported where EAs showed extraordinary optimization performance (e.g. [20, 29, 21, 5]). It is therefore interesting and important to investigate and understand the effectiveness of the heuristics in EAs.

Theoretical studies of EAs have been developed rapidly in the recent decades, particularly noticeable of the blooming of running time analysis [35, 2, 22]. With the development of several analysis techniques (e.g. [18, 45, 10, 39]), EAs have been theoretically investigated on problems from simple synthetic ones (e.g. [14]) to combinatorial problems (e.g. [37]) as well as NP-hard problems (e.g. [46]). During these analyses, effects of EA components have been disclosed [44], including the crossover operators (e.g. [23, 34, 11, 36]), the population size (e.g. [26, 38, 42, 7]), etc. Measures of the performance also have developed to cover the approximation complexity (e.g. [19, 16, 46, 31]), the fixed-parameter complexity (e.g. [30, 40], the complexity under fixed-budget computation [24], etc. While most of these analysis studied instances of EAs on problem cases, general performance analysis may even be more desired for supporting the applications of EAs in various tasks. The famous No-Free-Lunch Theorem [43] used a quite general framework of EAs and gave a general conclusion that any two EAs are with the same performance (at least on discrete domains) given no prior knowledge of the problem distribution, of which the general running time is exponential [45]. When the problem size is bounded, a general convergence lower bound can be derived for evolutionary strategies [15]. For more general EAs, the Black-Box model can derive the best possible performance [13, 133, 9].

Different with the previous studies, this work tries to understand the heuristics in EAs from a statistical view. Since most EAs follow the trial-and-error style, we summarize their structure as the sampling-and-learning (SAL) framework. EAs commonly employ some heuristic to reproduce solutions, which is modeled as the sampling stage; they also distinguish the quality of the reproduced solutions to guide the next sampling (e.g. genetic algorithms remove a portion of the worst solutions), which is modeled as the learning stage. SAL framework can cover a wide range of EAs instances by specifying the sampling strategy and the learning strategy. We analyze the SAL framework using the probable-absolute-approximate (PAA) query complexity, which counts the number of fitness evaluations before reaching to an approximate solution, with a probability. The PAA query complexity is close to the intuitive evaluation of EAs in practices.

We show that the SAL framework immediately admits a general PAA bound. For the SAC algorithms, a specific version of SAL that uses classifications, a tighter general PAA bound is given. Comparing with the uniformly random search, we disclose that, under the error-target independence condition, SAC algorithms can polynomially reduce the complexity of the uniform search, but not super-polynomially; while the error-successive-level independence condition allows a super-polynomial improvement. This study shows the error factor can drasti-
cally effect the performance of EAs, which was not noticed before. We also notice that a good learning algorithm may not be necessary for a good SAL algorithm.

The rest of this paper is organized as follows: Section II introduces the SAL framework. In Section III, we compare the SAC algorithms, a specific version of the SAL framework, with the uniform search. Finally, Section IV concludes the paper.

2. The Sampling-and-Learning Framework

In this paper, we consider general minimization problems. We always denote \( X \) as the whole solution space which an algorithm will search among, \( U \) as the uniform distribution over \( X \). We use \( D \) to denote sub-regions of \( X \), and \( D \) to denote probability distributions. By \( \text{poly}(\cdots) \), we mean the set of all polynomials of the related variables, and by \( \text{superpoly}(\cdots) \), we mean the set of all functions that grow faster than any function in \( \text{poly}(\cdots) \) with the related variables.

**Definition 1** (Minimization Problem)

A minimization problem consists of a solution space \( X \) and a function \( f : X \to \mathbb{R} \). The goal is to find a solution \( x^* \in X \) such that \( f(x^*) \leq f(x) \) for all \( x \in X \).

In the rest of the paper, we assume without loss of generality that the value of \( f \) is bounded in \([0, 1]\), i.e., \( \forall x \in X : f(x) \in [0, 1] \). Given an arbitrary function \( g \) with bounded value range over the input domain, the bound can be implemented by a simple normalization \( f(x) = \frac{g(x) - g(x^*)}{\max_{x'} g(x') - g(x^*)} \). Therefore, under our assumption the optimal value is 0.

For the performance evaluation, we will analyze the **probable-absolute-approximate** (PAA) query complexity, which is the number of fitness evaluations that an algorithm takes before reaching an approximate quality, as defined in Definition[2]. The PAA query complexity closely reflects our intuitive evaluation of EAs in real-world practice, where we expect EAs to achieve some good enough solutions with a sufficient probability.

**Definition 2** (Probable-Absolute-Approximate Query Complexity)

Given an algorithm \( A \) and a problem \( f \), the probable-absolute-approximate (PAA) query complexity is the number of calls to \( f(\cdot) \) such that, with probability at least \( 1 - \delta \), \( A \) finds a solution \( x \) with \( f(x) \leq \alpha^* \), for any \( \alpha^* > 0 \) and \( \delta > 0 \).

2.1. The General Framework

Most EAs share a common trial-and-error structure with several important properties:

a) directly access the solutions space, generate solutions, and evaluate the solutions;
b) the generation of new solutions depends only on a short history of past solutions;

c) both “global” and “local” heuristic operators are employed to generate new solutions.

We present a sampling-and-learning framework in Algorithm 1 to capture these properties. The SAL framework starts from a random sampling in Step 1 like all EAs. Step 2 and Step 13 record the best-so-far solutions throughout the search. It follows a cycle consists of learning and sampling stages. In Step 7, it learns a hypothesis \( h_t \) (i.e., a mapping from \( X \) to \( \mathbb{R} \)) via the learning algorithm \( L \). The learning algorithm allows to take the current data set \( T_t \), the last data set \( T_{t-1} \), and the last hypothesis \( h_{t-1} \) into account. We will see different EAs may make different use of them. Step 8 initializes the sample set for the next iteration. The sample set can be initialized as an empty set, or to preserve some good solutions from \( T_t \). In Steps 9 to 12, it samples from the distribution transformed from the hypothesis as well as from the whole solutions space balanced by a probability. The distribution \( T_{h_t} \) implies the potential good regions learned by \( h_t \).

It should be noted that the SAL framework is not a concrete optimization algorithm but an abstract summary of a range of EAs, nor does the learning stage of the framework imply an accurate learning. We explain in the follows how we could mimic several different EAs by the SAL framework. It is noticeable that the explanation is not a rigorous proof, but an intuitive illustration that the SAL framework can correspond to various implementations.

The genetic algorithms (GAs) deal with discrete solutions space, which consists of solutions represented as a vector of vocabulary. The element-wise mutation operator changes every element of a solution to a randomly selected word from the vocabulary with a probability. Convert this operation probability to the probability of generating a certain solution, it is easy to calculate that only the solutions having constant changes to a current solution are with a high (polynomially large) probability to be generated, and the remaining solutions are with a super-polynomially small probability to be generated. This distribution can be approximately regarded as a combination of the uniform distribution over the entire input space, which results in a super-polynomially small probability to generate every solution, and the local distribution around the present solutions, which is defined by changing the solution within a constant number of elements. Most GAs also employ the crossover operators, which is a kind of local search operator and thus the resulting distribution can be compiled into the local distribution. After generating a set of new solutions, genetic algorithms will select to keep only a part of the best solutions from the collected solutions. This can be simulated in SAL by configuring the learning algorithm: firstly label the selected solutions as positive and the remaining ones as negative, and learn a classification hypothesis from the labeled data that follows the local distribution. GAs often use a probabilistic selection, which can be simulated by selecting the initial solution set. In the simulation of GAs, the learning step does not utilize the last generated hypothesis and the last data set, and thus the initial hypothesis does not matter.

The estimation of distribution algorithms (EDAs) have a straightforward matching with the SAL framework. In every iteration, EDAs learn a probability distribution model from a part of the best solutions, and sample
Algorithm 1 The sampling-and-learning (SAL) framework

Input:

\( \alpha^* > 0 \): Approximation level

\( T \in \mathbb{N}^+ \): Number of iterations

\( m_0, \ldots, m_T \in \mathbb{N}^+ \): Number of samples

\( \lambda \in [0, 1] \): Balancing parameters

\( \mathcal{L} \): Learning algorithm

\( \mathcal{T} \): Distribution transformation of hypothesis

Procedure:

1: Collect \( S_0 = \{ x_1, \ldots, x_{m_0} \} \) by i.i.d. sampling from the uniform distribution over \( X \)
2: \( \hat{x} = \arg\min_{x \in S_0} f(x) \)
3: Generate the initial hypothesis \( h_0 \)
4: \( T_0 = \emptyset \)
5: for \( t = 1 \) to \( T \) do
6: Construct \( T_t = \{ (x_1, y_1), \ldots, (x_{m_{t-1}}, y_{m_{t-1}}) \} \), where \( x_i \in S_{t-1} \) and \( y_i = f(x_i) \)
7: \( h_t = \mathcal{L}(T_t, T_{t-1}, h_{t-1}, t) \), the learning step
8: Initialize \( S_t \) from \( T_t \)
9: for \( i = 1 \) to \( m_t \) do
10: Sample \( x_i \) from \( T_{h_t} \), with probability \( \lambda \)
11: \( S_t = S_t \cup \{ x_i \} \) with probability \( 1 - \lambda \)
12: end for
13: \( \hat{x} = \arg\min_{x \in S_t \cup \{ \hat{x} \}} f(x) \)
14: end for
15: return \( \hat{x} \)

solutions from the model. To simulate an EDA with a probabilistic local model, a SAL algorithm can be configured by selecting the same best solutions and letting the \( h_t \) be the distribution model and disabling the uniform sampling (i.e., \( \lambda = 1 \)); and for an EDA with a probabilistic model spreading over the entire solution space, we can firstly decompose the model as a combination of a local model and a nearly uniform distribution over the solution space, which can then be approximated by a SAL algorithm. Noting that for EDAs, the learning algorithms only utilize the current data (i.e., \( T_t \)) and may utilize the last hypothesis, but not the last data set.

The any colony optimization algorithms (ACOs) deal with a (underlying) graph and seek the path with the optimal evaluation. ACOs store in each node of the graph the probability distribution of selecting each outgoing path. A new solution is sampled from the distribution in every node of the graph, in other words, the joint
probability distribution of all nodes. Then the quality of every solution is used to update the distribution. To simulate an ACO, a SAL algorithm can be configured as that, first, the initial hypothesis is derived from the initial joint distribution of all nodes. The learning algorithm updates the last hypothesis as the ACO formula. Noting that an ACO commonly employs a small randomization in every choice of the path, this can be modeled by the sampling from the uniform distribution. For ACOs, the learning algorithms utilize the current data \((T_t)\) and the last hypothesis \((h_{t-1})\), but not the last data set.

The particle swarm optimization algorithms (PSOs) \([27]\) maintain a set of “flying” particles each with a location (representing a solution) and a velocity vector. In PSOs, the location of a particle in the next iteration is determined by its current location and current velocity, and the velocity is updated by the current velocity and the locations of the “globally” and “personally” best particles. To simulate a PSO, a SAL algorithm uses the initial hypothesis that leads to the sampling distribution same as that resulted from the initial velocity. Let \(S_t\) be an ordered set to contain the globally best particle and the personally best particles in Step 8. The learning algorithm in the SAL algorithm can be set to utilize the current data set and the last data set to recover the velocity, and utilize the last hypothesis and the globally and personally best particles recorded through \(S_t\) to generate the new hypothesis that simulates the movement of particles in the PSO.

Overall, the SAL framework captures the trial-and-error structure as well as the global–local search balance, while leaving the details of the local sampling distribution being implemented by different heuristics. The SAL framework directly admits a general upper bound of the PAA query complexity, as stated in Theorem 1.

**Theorem 1**

For any minimization problem \(f\) and any approximation level \(\alpha^* > 0\), with at least \(1 - \delta\) probability, a SAL algorithm will output a solution \(x\) with \(f(x) \leq \alpha^*\) using the number of queried samples \(m_{\Sigma}\) bounded above as

\[
O \left( \max \left\{ \frac{1}{(1 - \lambda) \Pr_u + \lambda \Pr_h} \ln \frac{1}{\delta}, m_0 + \sum_{t=1}^{T} m_{Pr_{ht}} \right\} \right),
\]

where

\[
\Pr_u = \int_{X_{\alpha^*}} u_X(x) \, dx
\]

is the success probability of uniform sampling,

\[
\Pr_h = \frac{\sum_{t=1}^{T} m_t \cdot \int_{X_{\alpha^*}} T_{ht}(x) \, dx}{\sum_{t=1}^{T} m_t}
\]

is the average success probability of sampling from the learnt hypothesis, \(m_{Pr_{ht}}\) is the sample size required to obtain \(\Pr_{ht}\), and \(X_{\alpha^*} = \{x \in X : f(x) \leq \alpha^*\}\).

**Proof.** The \(m_0 + \sum_{t=1}^{T} m_{Pr_{ht}}\) part is nature. We prove the rest part of the bound. Let’s consider the probability that after \(T\) iterations, the SAL algorithm outputs a bad solution \(x\) such that \(f(x) > \alpha^*\). Since the \(x\) is the best
solution among all sampled examples, the probability is the joint of events that every step of the sampling does not generate such a good solution.

1. For the sampling from uniform distribution over the whole solution space $X$, the probability of failure is $1 - \Pr_u$.

2. For the sampling from the learnt hypothesis $T(h_t)$, the probability of failure is denoted as $1 - \Pr_{h_t}$.

Since that every sampling is independent, we can expand the probability of overall failures, i.e.,

$$\Pr(f(x) > \alpha^*) = (1 - \Pr_u)^{m_0} \prod_{t=1}^{T} \sum_{i=0}^{m_t} \binom{m_t}{i} (1 - \lambda)^i \lambda^{m_t-i}$$

$$= (1 - \Pr_u)^{m_0} \prod_{t=1}^{T} ((1 - \lambda)(1 - \Pr_u) + \lambda(1 - \Pr_{h_t})))^{m_t}$$

$$= (1 - \Pr_u)^{m_0} \prod_{t=1}^{T} (1 - (1 - \lambda)\Pr_u - \lambda\Pr_{h_t})^{m_t}$$

$$\leq e^{-\Pr_u \cdot m_0} \prod_{t=1}^{T} e^{-(1-\lambda)\Pr_u m_t + \lambda \Pr_{h_t} m_t}$$

$$= e^{-\Pr_u \cdot m_0 + (1-\lambda) \sum_{t=1}^{T} \Pr_u m_t + \lambda \sum_{t=1}^{T} \Pr_{h_t} m_t}$$

where the inequality is by $(1 - x) \leq e^{-x}$ for $x \in [0, 1]$.

At the same time, letting $\Pr(f(x) > \alpha^*) < \delta$, we get

$$e^{-\Pr_u \cdot m_0 + (1-\lambda) \sum_{t=1}^{T} \Pr_u m_t + \lambda \sum_{t=1}^{T} \Pr_{h_t} m_t} < \delta,$$

which results the theorem by noticing that $m_\Sigma > \sum_{t=1}^{T} m_t$. 

2.2. The Sampling-and-Classification Algorithms

If we focus on GAs, we can have a simplified version of the SAL framework that employs a classification algorithm in the learning stage. We call this type of algorithms as the sampling-and-classification (SAC) algorithms. In the learning stage of a SAC algorithm, as described in Algorithm 2, the learning algorithm first uses a threshold to transform the data set into a binary labeled data set. Note that $\text{sign}(v) = +1$ if $v \geq 0$ and $-1$ if $v < 0$. Also note that SAC algorithms use the current data set in the learning algorithm, but not the last data set and the last hypothesis. We always set $S_t = \emptyset$ for SAC algorithms. In the sampling step of a SAC algorithm, $T(h)$ will be some distribution over the positive area of $h$.

By these specifications, we can have a general PA A performance for SAC algorithms. According to Theorem 1, we need to estimate an upper bound of $\Pr_h$, i.e., how likely the distribution $T_{h_t}$ will lead to a good solution.
Algorithm 2 Learning sub-procedure for the sampling-and-classification (SAC) algorithms

**Input:**
- $T, T'$, $h', t$: The input variables
- $\alpha_1, \ldots, \alpha_t$: Preset parameters
- $C$: Classification algorithm

**Procedure:**
1. Let $B = \{(x_1, z_1), \ldots, (x_{m-1}, z_{m-1})\}$, where $(x_i, y_i) \in T$ and $z_i = \text{sign}[\alpha_t - y_i]$ for all $i$
2. $h = C(B)$
3. return $h$

Denote $D_{\alpha^*} = \{x \in X \mid f(x) \leq \alpha^*\}$ for any scaler $\alpha^*$, $D_h = \{x \in X \mid h(x) = +1\}$ for any hypothesis $h$, $U_{D_h}$ as the uniform distribution over $D_h$, $D_{KL}$ as the KL-divergence, and $|D| = \int_D 1dx$. We have a lower bound of the success probability as in Lemma 1 which implies that without any prior knowledge, uniform distribution is as the best worst case.

**Lemma 1**

For any minimization problem $f$, any approximation level $\alpha^* > 0$, any hypothesis $h$, the probability that a sample sampled from an arbitrary distribution $T_h$ defined on $D_h$ will lead to a solution in $D_{\alpha^*}$ is lower bounded as

$$
\Pr_h \geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - \frac{1}{2} D_{KL}(T_h || U_{D_h})
$$

**Proof.** The proof starts from the definition of the probability,

$$
\Pr_h = \int_{D_h} T_h(x) \cdot I(x \in D_{\alpha^*})dx
$$

$$
= \int_{D_h} (T_h(x) - U_{D_h}(x) + U_{D_h}(x)) \cdot I(x \in D_{\alpha^*})dx
$$

$$
= \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} + \int_{D_h} (T_h(x) - U_{D_h}(x)) \cdot I(x \in D_{\alpha^*})dx
$$

$$
\geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - \int_{D_h} \sup_{x'} |T_h(x') - U_{D_h}(x')| \cdot I(x \in D_{\alpha^*})dx
$$

$$
\geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - \sqrt{\frac{1}{2} D_{KL}(T_h || U_{D_h})} \int_{D_h} I(x \in D_{\alpha^*})dx
$$

$$
= \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - |D_{\alpha^*} \cap D_h| \sqrt{\frac{1}{2} D_{KL}(T_h || U_{D_h})},
$$

where the last inequality is by the Pinsker’s inequality.

We cannot determine $D_h$, but we know that $h$ is derived by a binary classification learning algorithm from a data set which is labeled according to $D_{\alpha_t}$ for some $t$. For the binary classification, we know that the generalization
error, which is the expected misclassification rate, can be bounded above by the training error, which is the misclassification rate in the seen examples, as well as the generalization gap involving the complexity of the hypothesis space indicated by the VC-dimension [26], as in Lemma 2.

**Lemma 2** ([26])

Let $\mathcal{H} = \{h : X \rightarrow \{-1, +1\}\}$ be the hypothesis space containing a family of binary classification functions and $\text{VC}(\mathcal{H}) = d$, if there exist $m$ samples i.i.d. from $X$ according to some fixed unknown distribution $\mathcal{D}$, then, \( \forall h \in \mathcal{H} \) and \( \forall 0 < \eta < 1 \), the following upper bound holds true with probability at least \( 1 - \eta \):

$$
\epsilon_D \leq \hat{\epsilon}_D + \sqrt{8m^{-1}(d \log (2emd^{-1}) + \log (4\eta^{-1}))}
$$

where \( \epsilon_D \) is the expected error rate of \( h \) over \( D \) and \( \hat{\epsilon}_D \) is the error rate in the sampled examples from \( D \), and when \( \hat{\epsilon}_D = 0 \),

$$
\epsilon_D \leq 2m^{-1}(d \log (2emd^{-1}) + \log (2\eta^{-1})).
$$

Again by the Pinsker’s inequality, we know that the error \( \epsilon_D \) under the distribution \( D \) can be converted to the error \( \epsilon_U \) under the uniform distribution, as

$$
\epsilon_U \leq \frac{\epsilon_D}{1 - |X| \sqrt{\frac{1}{2}D_{KL}(D \| U)}} \leq \frac{\hat{\epsilon}_D + \sqrt{\frac{8}{m} (d \log 2emd^{-1} + \log 4\eta^{-1})}}{1 - |X| \sqrt{\frac{1}{2}D_{KL}(D \| U)}},
$$

where we only take count the event that the generalization inequality holds with probability \( 1 - \eta \). For simplicity, we denote the right-hand part as $\Psi^{m,\eta}_{\epsilon_D,d,D_{KL}(D \| U)}$, which decreases with $m$ and $\eta$, and increases with $\epsilon_D$, $d$, and $D_{KL}(D \| U)$.

We can use this result to eliminate the need of $D_h$ in Lemma 1. In every iteration of SAC algorithms, there are $m_t$ samples collected.

**Theorem 2**

For any minimization problem $f$, any constant $\eta > 0$, and any approximation level $\alpha^* > 0$, the average success probability of sampling from the learnt hypothesis of any SAC algorithm is bounded below as

$$
\Pr_h \geq 1 - \eta \sum_{t=1}^T \sum_{i=1}^{m_t} \left( \frac{|D_{\alpha^*}| - 2\Psi^{m_t,\eta}_{\epsilon_D,d,D_{KL}(D_h \| U_X)} - 2|\alpha^*| \sqrt{\frac{1}{2}D_{KL}(T_h, ||U_h)||U_D)} \right),
$$

where $D_h = \lambda T_h + (1 - \lambda)U_X$ is the sampling distribution at iteration $t$, $\epsilon_D$, is the training error rate of $h_t$, $d$ is the VC-dimension of the learning algorithm.

**Proof.** By set operators,

$$
|D_{\alpha^*} \cap D_{h_t}| = |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{h_t}|
$$
\[
\begin{align*}
&\geq |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{\alpha_t}| - |D_{\alpha_t} \Delta D_{h_t}| \\
&= |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{\alpha_t}| - \epsilon_{U,t} \\
&= |D_{\alpha^*} \cup D_{h_t}| + |D_{\alpha^*} - |D_{\alpha_t}| - \epsilon_{U,t},
\end{align*}
\]

where \(\Delta\) is the symmetric difference operator of two sets, the first inequality is by the triangle inequality, and the last equation is by that \(D_{\alpha^*}\) is contained in \(D_{\alpha_t}\).

Since \(|D_{h_t}| - |D_{\alpha_t}|\) \(\leq |D_{h_t} \Delta D_{\alpha_t}| = \epsilon_{U,t}\), we can bound \(|D_{h_t}|\) as \(|D_{\alpha_t}| + \epsilon_{U,t} \geq |D_{h_t}| \geq |D_{\alpha_t}| - \epsilon_{U,t}\).

Assume \(\frac{1}{|D_{h_t}|} \geq \sqrt{\frac{1}{2} D_{KL}(T_h||U_{D_{h_t}})}\), since \(Pr_h\) can not be smaller than 0. Therefore, the success probability of sampling from \(h_t\) is lower bounded

\[
Pr_{h_t} \geq \left(\frac{|D_{\alpha^*} \cup D_{h_t}| + |D_{\alpha^*} - |D_{\alpha_t}| - \epsilon_{U,t}|}{|D_{\alpha^*}| - \epsilon_{U,t}}\right) \cdot \left(\frac{1}{|D_{h_t}|} - \sqrt{\frac{1}{2} D_{KL}(T_{h_t}||U_{D_{h_t}})}\right)
\]

\[
\geq \frac{|D_{\alpha^*}| - 2\epsilon_{U,t}}{|D_{\alpha^*}| - \epsilon_{U,t}} - 2|D_{\alpha^*}| \sqrt{\frac{1}{2} D_{KL}(T_{h_t}||U_{D_{h_t}})}.\]

Substituting this lower bound and the probability \(1 - \eta\) of the generalization bound into \(Pr_h\) obtains the theorem.

Combining Theorem 1 and Theorem 2 results an upper bound on the sampling complexity of SAC algorithms. Although the expression is sophisticated, it can still reveal relative variables that generally effect the complexity. One could design various distributions for \(T_h\) to sample potential solutions, however, without any a priori knowledge, the uniform sampling will lead to the best worst case performance. Meanwhile, without any a priori knowledge, a small training error at each stage from a learning algorithm with a small VC-dimension will improve the performance, which was not aware previously.

3. SAC Algorithms v.s. Uniform Search

When EAs are applied, we usually expect that they can achieve a better performance than some baselines. The uniform search can serve as a baseline, which searches the solution space always by randomly sampling solutions uniformly at random. In other words, the uniform search is the SAL algorithm with \(\lambda = 0\). In this section, we study the performance of SAC algorithms relative to uniform search.

SAC algorithms will degenerate to uniform search if \(\lambda = 0\). Thus, it is easy to know that the PAA query complexity of uniform search is

\[
\Theta\left(\frac{1}{Pr_{\alpha}} \cdot \ln \frac{1}{\delta}\right).
\]
How much a SAC algorithm improves from the uniform search depends on the average success probability $\mathbb{P}_{\mathbb{F}_h}$ that is from the learnt hypothesis. A SAC algorithm is not always better than the uniform search. Without any restriction, $\mathbb{P}_{\mathbb{F}_h}$ can be zero and thus the SAC algorithm is worse. We are interested in investigating the conditions under which SAC algorithms can accelerate from the uniform search.

3.1. A Polynomial Acceleration Condition

Condition 1 (Error-Target Independence)

In SAC algorithms, for any $t$ and any approximation level $\alpha^* > 0$, when sampling a solution $x$ from $\mathcal{U}_X$, the event $x \in D_{h_t} \Delta D_{\alpha_t}$ and the event $x \in D_{\alpha^*}$ are independent.

We call SAC algorithms that are under the error-target independence condition as SAC$_I$ algorithms. The condition is defined using the independence of random variables, which is equivalent to

$$|D_{\alpha^*} \cap (D_{\alpha_t} \Delta D_{h_t})| = |D_{\alpha^*}| \cdot |(D_{\alpha_t} \Delta D_{h_t})|,$$

from the set perspective. Under the condition, we can have a lower bound on the probability of sampling a good solution, as stated in Lemma 3.

Lemma 3

For SAC$_I$ algorithms, it holds for all $t$ that

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{|D_{\alpha^*}|(1 - \epsilon_{\mathbb{F}_t})}{|D_{\alpha_t}| + \epsilon_{\mathbb{F}_t}},$$

where $\epsilon_{\mathbb{F}_t}$ is the error rate of $h_t$ under uniform distribution $\mathbb{U}$ over $X$.

Proof. For the numerator,

$$|D_{\alpha^*} \cap D_{h_t}| = |D_{\alpha^*}| - |D_{\alpha^*} \cap (D_{\alpha_t} \Delta D_{h_t})|$$

$$= |D_{\alpha^*}| - |D_{\alpha^*}| \cdot |D_{\alpha_t} \Delta D_{h_t}|$$

$$\geq |D_{\alpha^*}|(1 - \epsilon_{\mathbb{F}_t}),$$

where the first equation is by $D_{\alpha^*} \subseteq D_{\alpha_t}$, the second equality is by the error-target independence condition.

For the denominator, we consider the worst case that all errors are out of $D_{h_t}$ and thus $|D_{h_t}| = |D_{\alpha_t}| + \epsilon_{\mathbb{F}_t}$.

$\square$

Similar to Theorem 2, we can bound the average success probability of sampling from the learnt hypothesis,

$$\mathbb{P}_{\mathbb{F}_h} \geq \frac{1}{\sum_{t=1}^{T} \sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \left( \frac{|D_{\alpha^*}|(1 - \epsilon_{\mathbb{F}_t})}{|D_{\alpha_t}| + \epsilon_{\mathbb{F}_t}} - 2|D_{\alpha^*}| \sqrt{\frac{1}{2} D_{KL} (\mathbb{F}_h || \mathbb{U}_{D_{h_t}}) } \right)$$

11
We compare uniform search with the SAC$_1$ algorithms using uniform sampling within $D_{h_t}$, i.e., $D_{KL}(T_{h_t} \| U_{D_{h_t}}) = 0$, which is an optimistic situation. Then by Lemma 4

$$
\Pr_h \geq \frac{1}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \left( \frac{|D_{\alpha^*}|(1 - \epsilon_{t,t})}{|D_{\alpha^*}| + \epsilon_{t,t}} \right).
$$

By plugging $\epsilon_{t,t} \leq \frac{\epsilon_{D_{h_t}}}{1 - |X|\sqrt{\frac{D_{KL}(D_t \| U_t)}}} = Q \cdot \epsilon_{D_t}$, where $\epsilon_{D_t}$ is the error rate of $h_t$ under its training distribution $D_t$ and $Q = \left( 1 - |X|\sqrt{\frac{D_{KL}(D_t \| U_t)}} \right)^{-1}$,

$$
\Pr_h \geq \frac{1}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \left( \frac{|D_{\alpha^*}|(1 - Q \cdot \epsilon_{D_t})}{|D_{\alpha^*}| + Q \cdot \epsilon_{D_t}} \right). \tag{1}
$$

Note that, as long as a concept is learnable, the convergence rate of the error is $\tilde{O}(\frac{1}{m})$ ignoring other variables and logarithmic terms from Lemma 2. We then compare the query complexity of SAC$_1$ using learning algorithms with convergence rate $\tilde{\Theta}(\frac{1}{m})$ with that of uniform search. First, we find that the SAC$_1$ algorithms cannot exponentially improve the uniform search in the worst case, as Proposition I.

**Proposition 1**

Using learning algorithms with convergence rate $\tilde{\Theta}(\frac{1}{m})$, for any problem $f$ and any approximation level $\alpha^* > 0$, if the query complexity of uniform search is $\text{superpoly}(\frac{1}{\alpha^*}, n)$, the query complexity of SAC$_1$ algorithms is also $\text{superpoly}(\frac{1}{\alpha^*}, n)$ in the worst case.

**Proof.** The query complexity of uniform search being $\text{superpoly}(\frac{1}{\alpha^*}, n)$ implies that

$$
\frac{1}{\Pr_u} = \frac{|X|}{|D_{\alpha^*}|} = \text{superpoly}(\frac{1}{\alpha^*}, n).
$$

For the SAC$_1$ algorithms, if we ask the learning algorithm to produce a classifier with error rate in $\text{superpoly}(\frac{1}{\alpha^*}, n)$, it would require $\text{superpoly}(\frac{1}{\alpha^*}, n)$ number of samples in the worst case, so that the proposition holds. To avoid this, we can only expect the error rate to be in $\text{poly}(\frac{1}{\alpha^*}, n)$ in order to keep the query complexity in each iteration small.

Meanwhile, we can only have $T = \text{poly}(\frac{1}{\alpha^*}, n)$ iterations otherwise we will have super-polynomial number of samples.

Following the optimistic case of Eq. (3.1), since $Q \geq 1$, we consider one more optimistic situation that $Q = 1$. Even though, we have that

$$
\Pr_h \geq \frac{m_T}{\sum_{t=1}^{T} m_t} \left( \frac{|D_{\alpha^*}|(1 - \epsilon_{D_{T}})}{|D_{\alpha^*}| + \epsilon_{D_T}} \right) = \frac{\text{poly}(\frac{1}{\alpha^*}, n)}{|D_{\alpha^*}|} \cdot \frac{1}{\text{superpoly}(\frac{1}{\alpha^*}, n)} = \frac{1}{\text{superpoly}(\frac{1}{\alpha^*}, n)},
$$

where the first inequality is by considering only $t = T$ instead of the sum, and the last equation is since $|D_{\alpha^*}|$ does not matter. The equality is achievable from the inequality in the worst case. Then substituting $\Pr_h$ into Theorem I obtains the total samples $m_{\Sigma} = \text{superpoly}(\frac{1}{\alpha^*}, n)$, which proves the proposition. \qed
The proposition implies that the SAC\textsubscript{1} algorithms face the same barrier as that of the uniform search. However, the SAC\textsubscript{1} algorithms can still improve the uniform search within a polynomial factor. We show this by case studies.

**On Sphere Function Class:**

Given the solution space \(X_n = \{(x_1, \ldots, x_n) \mid \forall i = 1, \ldots, n : x_i \in [0, 1]\}\), the Sphere Function class is \(\mathcal{F}^n_{\text{sphere}} = \{f_{\text{sphere}}^x : x \in X_n\}\) where

\[
    f_{\text{sphere}}^x(x) = \frac{1}{n} \|x - x^*\|_2^2 = \frac{1}{n} \sum_{i=1}^n (x_i - x_i^*)^2.
\]

Obviously, \(|X| = 1, f_{\text{sphere}}^x \in [0, 1]\) is convex, and the optimal value is always 0. It is important to aware that the volume of an \(n\)-dimensional hyper-sphere with radius \(r = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2} + 1\right)} r^n\), where \(\Gamma(s) = \int_0^\infty t^{s-1} e^{-t} \, dt\), so that \([D_n] = \frac{\pi^\frac{n}{2}}{\Gamma\left(\frac{n}{2} + 1\right)} (n\alpha)^{n/2} = C_0(\alpha)^{n/2}\) for any \(\alpha > 0\), where \(C_0 = \Theta\left(\frac{(2\pi)^\frac{n}{2}}{\sqrt{n\pi}}\right)\), since the radius leading to \(f_{\text{sphere}}^x(x) = \frac{1}{n} \|x - x^*\|_2^2 \leq \alpha\) is \(\sqrt{n\alpha}\).

It is straightforward to obtain that, minimizing any function in \(\mathcal{F}^n_{\text{sphere}}\) using uniform search, the PAA query complexity with approximation level \(\alpha^* > 0\) is

\[
    O\left(\left(\frac{1}{\alpha^*}\right)^2 \ln \frac{1}{\delta}\right).
\]

with probability at least \(1 - \delta\).

We configure the SAC\textsubscript{1} algorithm to use the learning algorithm \(L_{\text{sphere}}\), which searches in the hypothesis space \(\mathcal{H}_n\), consisting of all the hyper-spheres in \(\mathbb{R}^n\) to find a sphere that is consistent with the training data. We simply assume the search of the consistent sphere is feasible. Note that \(VC(\mathcal{H}_n) = n + 1\).

**Lemma 4**

For any \(h_t\), denote \(\epsilon_U\) be the error rate of \(h_t\) under the uniform distribution over \(X\), and \(\epsilon_D\) be the error rate of \(h_t\) under the distribution \(D_t = \lambda U_{D_{h_t}} + (1 - \lambda) U_X\), then it holds that

\[
    \epsilon_U \leq \frac{1}{1 - \lambda} \epsilon_D,
\]

where \(\lambda \in [0, 1]\) and \(U_{D_{h_t}}\) is the uniform distribution over \(D_{h_t}\).

**Proof.** Let \(D_\neq\) be the area where \(h_t\) makes mistakes. We split \(D_\neq\) into \(D_\neq^+ = D_\neq \cap D_{h_t}\) and \(D_\neq^- = D_\neq \setminus D_{h_t}^+\). We can calculate the probability density that \(D_t(x) = \lambda \frac{|D_{h_t}|}{|X|} + (1 - \lambda) \frac{|D_{h_t}^+|}{|X|}\) for any \(x \in D_\neq^+\), and \(D_t(x) = (1 - \lambda) \frac{|X \setminus D_{h_t}|}{|X|} \frac{1}{|X \setminus D_{h_t}|} = (1 - \lambda) \frac{1}{|X|}\) for any \(x \in D_\neq^-\). Thus,

\[
    \epsilon_D = \int_X D_t(x) I[h_t \text{ makes mistake on } x] \, dx
    = \int_{D_\neq^-} D_t(x) \, dx + \int_{D_\neq^+} D_t(x) \, dx + \int_{D_\neq} D_t(x) \, dx
\]

13
\[ \geq \int_{D^+} (1 - \lambda) \frac{1}{|X|} dX + \int_{D^-} (1 - \lambda) \frac{1}{|X|} dX \]
\[ = (1 - \lambda) \epsilon_{\alpha_t}, \]

which proves the lemma. \[\square\]

We then obtain the PAA complexity as in Proposition 4.

\textbf{Proposition 2}

For any function in \( \mathcal{F}_{sphere}^n \) sphere and any approximation level \( \alpha^* > 0 \), SAC algorithms can achieve the PAA query complexity, for any \( n \geq 2 \),

\[ O \left( \frac{1}{\alpha^*} \log \frac{1}{\alpha^*} \ln \frac{1}{\delta} \right) \]

with probability at least \( 1 - \delta \).

\textbf{Proof.} We choose \( \alpha_t = \frac{1}{2^t} \) for all \( t \), and use the number of iterations \( T \) to approach \( |D_{\alpha_T}| = \sqrt{|D_{\alpha^*}|} \), for the approximation level \( \alpha^* \). Solving this equation with the sphere volume results in \( T = \log \left( \frac{C_0}{\sqrt{\alpha^*}} \right) \). We let the SAC algorithm run \( T = \log \frac{1}{\sqrt{\alpha^*}} \) number of iterations. We assume \( \log \frac{1}{\sqrt{\alpha^*}} \) is an integer for simplicity, which does not affect the generality.

In iteration \( t \), using \( L_{sphere} \), we want the error of the hypothesis \( h_t, \epsilon_{D_t} \), to be \( \frac{1}{2^t} \). Since the \( L_{sphere} \) produces a hypothesis with zero training error, from \( \epsilon_{D_t} = \frac{1}{2^t} \leq 2m^{-1} \left[ d \log (2emd^{-1}) + \log (2\eta^{-1}) \right] \),

we can solve the required sample size with \( \eta \) being a constant,

\[ m_t \leq m_T = O(n2^{T}T) = O \left( \frac{n}{\sqrt{\alpha^*}} \log \frac{1}{\sqrt{\alpha^*}} \right) \]

using the inequality \( \log x \leq cx - (\log c + 1) \) for any \( x > 0 \) and any \( c > 0 \). We thus obtain \( m_0 + \sum_{t=1}^{T} m_t = O \left( \frac{n}{\sqrt{\alpha^*}} (\log \frac{1}{\sqrt{\alpha^*}})^2 \right) \).

We then follow Eq. (3.1). We use uniform sampling within \( D_{h_t} \), then \( Q = \frac{1}{\sqrt{\lambda}} \). Letting the SAC algorithms use \( m_T \) number of samples in every iteration and \( \lambda = 0.5 \), we have

\[ \Pr_{Fh} \geq \frac{1}{\log \frac{1}{\sqrt{\alpha^*}}} \sum_{t=1}^{\log \frac{1}{\sqrt{\alpha^*}}} \left( \frac{|D_{\alpha^*}| (1 - Q\epsilon_{D_t})}{|D_{\alpha^*}| + Q\epsilon_{D_t}} \right) \]
\[ \geq \frac{C_0(\alpha^*)^{2}}{\log \frac{1}{\sqrt{\alpha^*}}} \sum_{t=1}^{\log \frac{1}{\sqrt{\alpha^*}}} \frac{1 - 2 \frac{1}{2^t}}{C_0 \left( \frac{1}{2^t} \right)^2 + 2 \frac{1}{2^t}} \]
\[ \geq \frac{C_0(\alpha^*)^{2}}{\log \frac{1}{\sqrt{\alpha^*}}} \frac{1}{2(C_0 + 2)} \sum_{t=2}^{\log \frac{1}{\sqrt{\alpha^*}}} \frac{1}{2^t} \]
\[ = \frac{C_0(\alpha^*)^{2}}{\log \frac{1}{\sqrt{\alpha^*}} (C_0 + 2)} = \Omega \left( \frac{(\alpha^*)^{\frac{n-1}{2}}}{\log \frac{1}{\sqrt{\alpha^*}}} \right). \]
So we obtain the query complexity from Theorem 1

\[ O \left( \max \left\{ \frac{1}{\alpha^*} \frac{n}{2} \log \frac{1}{\sqrt{\alpha^*}} \ln \frac{1}{\delta}, \frac{n}{2} (\log \frac{1}{\sqrt{\alpha^*}})^2 \right\} \right) \]

which is

\[ O \left( \frac{1}{\alpha^*} \frac{n}{2} \log \frac{1}{\sqrt{\alpha^*}} (\ln \frac{1}{\delta} + n \log \frac{1}{\sqrt{\alpha^*}}) \right). \]

We can see that the SAC algorithms can accelerate uniform search by the factor near \( \frac{1}{\sqrt{\alpha^*}} \). The closer approximation, the more acceleration.

**On Spike Function Class**

As modeling EAs, SAL algorithms should be expected to be applied on complexity problems, while the Sphere Function class only consists of convex functions. Inherited from EAs, SAL algorithms can handle problems with small local optima. We show this by comparing SAC with uniform search on the Spike function class defined below.

Define regions \( A_{1,k} = \left[ \frac{3k}{20}, \frac{3k+2}{20} \right] \) where \( 0 \leq k \in \mathbb{Z} \leq 6 \) and \( A_{2,k} = \left( \frac{3k-1}{20}, \frac{3k}{20} \right) \) where \( 1 \leq k \in \mathbb{Z} \leq 6 \), and define \( g(x) \) over \([0, 1]\) that

\[ g(x) = \begin{cases} 
  x - \frac{k}{10}, & x \in A_{1,k} \\
  -x + \frac{k}{5}, & x \in A_{2,k} 
\end{cases} \]

Let \( X = [-\frac{1}{2}, \frac{1}{2}] \) be the one-dimensional solution space. The Spike Function class is \( \mathcal{F}_{\text{spike}} = \{ f_{\text{spike}}^x \mid \forall x^* \in X \} \), where \( f_{\text{spike}}^x(x) = g(|x - x^*|) \) and \( x \in X \). It is easy to know for any \( f \in \mathcal{F}_{\text{spike}} \) that \( \min_{x \in X} f(x) = 0 \) and \( \max_{x \in X} f(x) \leq 1 \). For any \( \alpha > 0 \), we can bound the area \( |D_\alpha| \in [\alpha, 6\alpha] \)

The Spike functions are non-convex and non-differentiable with local optima, as depicted in Figure [1].

It is again straightforward to obtain that, minimizing any function in \( \mathcal{F}_{\text{spike}} \) using uniform search, the PAA query complexity with approximation level \( \alpha^* > 0 \) is

\[ O \left( \frac{1}{\alpha^*} \ln \frac{1}{\delta} \right). \]

with probability at least \( 1 - \delta \).

We configure the SAC algorithm to use the learning algorithm \( \mathcal{L}_{\text{line}} \) that searches the shortest line that covers all the samples labeled as positive, of which the VC-dimension is 2. Note that since the function is non-convex, the \( \mathcal{L}_{\text{line}} \) may output a line that also covers some negative examples, and thus with some training error. Using this SAC algorithm to minimize any member in the function class \( \mathcal{F}_{\text{spike}} \), we obtain the PAA query complexity as in Proposition [3].
Proposition 3

For any function in $\mathcal{F}_{\text{spike}}$ and any approximation level $\alpha^* > 0$, SAC algorithms can achieve the PAA query complexity

$$O \left( \frac{1}{\sqrt{\alpha^*}} \left( \log \frac{1}{\sqrt{\alpha^*}} \left( \log \frac{1}{\sqrt{\alpha^*}} + \ln \frac{1}{\delta} \right) \right) \right),$$

with probability at least $1 - \delta$.

**Proof.** For any function in $\mathcal{F}_{\text{spike}}$, we note that the function is convex in $D_\alpha$ when $\alpha$ is smaller than 0.05. We set $\alpha_t = \frac{1}{2^t}$, so that when $t \geq 5$, using $L_{\text{spike}}$ the SAC algorithm will deal with a convex function and thus the training error is zero. We use the number of iterations $T$ to achieve $|D_{\alpha T}| = \sqrt{|D_{\alpha^*}|}$. Since $|D_{\alpha}| \in [\alpha, 6\alpha]$, we can solve $T \geq \log \frac{6}{\sqrt{\alpha^*}}$, which we simply assume as an integer.

In iteration $t \geq 5$, we want the error of the hypothesis $h_t$, $\epsilon_{D_t}$, to be $\frac{1}{2^t}$. Since the training error can be zero, we can solve the required sample size $m_t \leq m_T = O \left( \frac{1}{\sqrt{\alpha^*}} \log \frac{1}{\sqrt{\alpha^*}} \right)$. We thus obtain $m_0 + \sum_{t=1}^{T} m_t = O \left( \frac{1}{\sqrt{\alpha^*}} \left( \log \frac{1}{\sqrt{\alpha^*}} + \frac{\ln 1}{\delta} \right) \right)$.

We then follow Eq. (3.1). We use uniform sampling within $D_{h_t}$, then $Q = \frac{1}{1 - \lambda}$. Letting the SAC algorithms use $m_T$ number of samples in every iteration and $\lambda = 0.5$, we have

$$\Pr_h \geq \frac{1}{\log \frac{\alpha^*}{\sqrt{\alpha^*}}} \sum_{t=5}^{T} \frac{\left( |D_{\alpha^*}|(1 - Q^2) \right)}{|D_{\alpha^*} + Q^2 \epsilon_{D_t}|} \geq \frac{2\alpha^*}{\log \frac{\alpha^*}{\sqrt{\alpha^*}}} \sum_{t=5}^{T} \frac{1 - 2 \frac{1}{2^t}}{6 \cdot 2^t + 2 \cdot 2^t} \geq \frac{2\alpha^*}{\log \frac{\alpha^*}{\sqrt{\alpha^*}}} \frac{15}{128} \sum_{t=5}^{T} \frac{2^t}{2^t} = \frac{\alpha^*}{\log \frac{\alpha^*}{\sqrt{\alpha^*}}} \frac{15}{64} \left( \frac{12}{\sqrt{\alpha^*}} - 2^5 \right) = \Omega \left( \frac{(\alpha^*)^{\frac{1}{2}}}{\log \frac{\alpha^*}{\sqrt{\alpha^*}}} \right).$$

So we obtain the query complexity from Theorem 1

$$O \left( \frac{1}{\sqrt{\alpha^*}} \left( \log \frac{1}{\sqrt{\alpha^*}} \left( \log \frac{1}{\sqrt{\alpha^*}} + \ln \frac{1}{\delta} \right) \right) \right)$$

as max is bounded by plus. \(\square\)

It can be observed from the proof that the non-convexity can result in non-zero training error for the learning algorithms in SAC algorithms, and thus the search process is interfered. But as long as the non-convexity is not quite severe, like the Spike functions, SAC algorithms can still be better than uniform search.

### 3.2. Super-Polynomial Acceleration Conditions

We have shown that, SAC algorithms under the error-target independence condition can not super-polynomially improve from uniform search in the worst case. An interesting question is therefore raised that when the super-polynomial improvement is possible. A straightforward way is to use a learning algorithm with exponentially
improved sample complexity, i.e., $\tilde{O}(\ln \frac{1}{\epsilon})$, such as the active learning algorithms in some circumstances (e.g. [8, 41]). Proposition 4 shows the result of using such a learning algorithm $L_{\text{sphere}}^\text{in}$. 

**Proposition 4**

For any function in $F_{n}$ and any approximation level $\alpha^* > 0$, SACI algorithms using $L_{\text{sphere}}^{\text{in}}$ can achieve the PAA query complexity, for any $n \geq 2$

$$O \left( n(\log \frac{1}{\alpha^*})^2 \ln \frac{1}{\delta} \right)$$

with probability at least $1 - \delta$.

**Proof.** We choose $\alpha_t = \frac{1}{2^t}$ for all $t$, and use the number of iterations $T$ to approach $|D_{\alpha_T}| = |D_{\alpha^*}|$, for the approximation level $\alpha^*$. Solving this equation with the sphere volume results in $T = \log \left( \frac{C_0}{\alpha^*} \right)$. We let the SACI algorithm run $T = \log \frac{1}{\alpha^*}$ number of iterations. We assume $\log \frac{1}{\alpha^*}$ is an integer for simplicity, which does not affect the generality.

In iteration $t$, using $L_{\text{sphere}}^{\text{in}}$, we want the error of the hypothesis $h_t$, $\epsilon_{D_t}$, to be $\frac{1}{2^t \eta}$. Since the $L_{\text{sphere}}^{\text{in}}$ has the sample complexity $O(\ln \frac{1}{\epsilon})$, we ask for a hypothesis with zero training error, which requires the sample size $m_t = O(n) = O(n \log \frac{1}{\alpha^*})$ with $\eta$ being a constant. We thus obtain $m_0 + \sum_{t=1}^{T} m_t = O \left( n \left( \log \frac{1}{\alpha^*} \right)^2 \right)$.

We then follow Eq.(5.1). We use uniform sampling within $D_{h_t}$, then $Q = \frac{1}{\epsilon^2}$. Letting the SACI algorithms use $m_T$ number of samples in every iteration and $\lambda = 0.5$, we have

$$P_{\text{tr}} \geq \frac{1}{\log \frac{1}{\alpha^*}} \sum_{t=1}^{\log \frac{1}{\alpha^*}} \left( \frac{|D_{\alpha^*}|(1 - Q \epsilon_{D_t})}{|D_{\alpha_t}| + Q \epsilon_{D_t}} \right)$$

$$\geq \frac{C_0(\alpha^*)^2}{\log \frac{1}{\alpha^*}} \sum_{t=1}^{\log \frac{1}{\alpha^*}} \frac{1 - 2(\frac{1}{\alpha^*})^2}{C_0(\frac{1}{\alpha^*})^2 + 2(\frac{1}{\alpha^*})^2}$$

$$\geq \frac{C_0(\alpha^*)^2}{\log \frac{1}{\alpha^*}} \frac{1}{2(C_0 + 2)} \sum_{t=2}^{\log \frac{1}{\alpha^*}} \frac{1}{(\frac{1}{\alpha^*})^2}$$

$$\geq \frac{C_0(\alpha^*)^2}{\log \frac{1}{\alpha^*}} \frac{1}{2(C_0 + 2)} \left( \frac{1}{\log \frac{1}{\alpha^*}} \right)^2 = \Omega \left( \frac{1}{\log \frac{1}{\alpha^*}} \right).$$

So we obtain the query complexity from Theorem 1

$$O \left( \max \left\{ \log \frac{1}{\alpha^*} \ln \frac{1}{\delta}, n(\log \frac{1}{\alpha^*})^2 \right\} \right) \quad (17)$$
which is $O \left(n(\log \frac{1}{\alpha})^2 \ln \frac{1}{\delta}\right)$. However, we are more interested in exploring another condition under which the super-polynomial improvement is possible without requiring such learning algorithms. For that, we find the error-successive-level independence condition.

**Condition 2** (Error-Successive-Level Independence)

In SAC algorithms, for any $t$ and any approximation level $\alpha^* > 0$, when sampling a solution $x$ from $\mathcal{U}_X$, the event $x \in D_{h_t} \Delta D_{\alpha_t}$ and the event $x \in D_{\alpha_t+1}$ are independent.

We call SAC algorithms that are further under this condition as SAC$^2$ algorithms.

**Lemma 5**

For SAC$^2$ algorithms, it holds for all $t$ that

$$|D_{\alpha_{t+1}} \cap D_{h_t}| = |D_{\alpha_{t+1}}|(1 - |D_{\alpha_t} \Delta D_{h_t}|).$$

**Proof.** Since $D_{\alpha_{t+1}} \subseteq D_{\alpha_t}$ for any $t = 1, \ldots, T-1$, one has $D_{\alpha_{t+1}} = (D_{\alpha_{t+1}} \cap (D_{\alpha_t} \Delta D_{h_t})) \cup (D_{\alpha_{t+1}} \cap D_{h_t})$. Besides, $(D_{\alpha_{t+1}} \cap (D_{\alpha_t} \Delta D_{h_t}))$ and $(D_{\alpha_{t+1}} \cap D_{h_t})$ do not intersect. Combining this fact with the condition of error-successive-level independence, we have

$$|D_{\alpha_{t+1}}| = |D_{\alpha_{t+1}} \cap (D_{\alpha_t} \Delta D_{h_t})| + |D_{\alpha_{t+1}} \cap D_{h_t}|$$

$$= |D_{\alpha_{t+1}}| \cdot |D_{\alpha_t} \Delta D_{h_t}| + |D_{\alpha_t+1} \cap D_{h_t}|.$$

Rearranging the equation above proves the lemma.

**Lemma 6**

For SAC$^2$ algorithms, it holds for all $t$ that

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{|D_{\alpha_{t+1}}|}{|D_{h_t}|} \cdot \frac{|D_{\alpha^*}|(1 - \epsilon_{\ell,t})^2}{|D_{\alpha_{t+1}}|},$$

where $\epsilon_{\ell,t}$ is the error rate of $h_t$ under uniform distribution $\mathcal{U}$ over $X$.

**Proof.** Since the SAC$^2$ algorithm is also a SAC$^1$ algorithm,

$$|D_{\alpha^*} \cap D_{h_t}| = |D_{\alpha^*}| - |D_{\alpha^*} \cap (D_{\alpha_t} \Delta D_{h_t})|$$

$$= |D_{\alpha^*}| - |D_{\alpha^*}| \cdot |D_{\alpha_t} \Delta D_{h_t}| = |D_{\alpha^*}|(1 - |D_{\alpha_t} \Delta D_{h_t}|).$$

Meanwhile, by Lemma 5

$$1 = \frac{|D_{\alpha_{t+1}} \cap D_{h_t}|}{|D_{\alpha_{t+1}} \cap D_{h_t}|} = \frac{|D_{\alpha_{t+1}} \cap D_{h_t}|}{|D_{\alpha_{t+1}}|(1 - |D_{\alpha_t} \Delta D_{h_t}|)}.$$

Hence,

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} = \frac{|D_{\alpha^*}|(1 - |D_{\alpha_t} \Delta D_{h_t}|)}{|D_{h_t}| \cdot 1}. $$
\[ = \frac{|D_{\alpha_{t+1}}|}{|D_{h_t}|} \cdot \frac{|D_{\alpha^*}|(1 - |D_{\alpha^*} \Delta D_{h_t}|)^2}{|D_{\alpha_{t+1}} \cap D_{h_t}|} \geq \frac{|D_{\alpha_{t+1}}|}{|D_{h_t}|} \cdot \frac{|D_{\alpha^*}|(1 - |D_{\alpha^*} \Delta D_{h_t}|)^2}{|D_{\alpha_{t+1}}|} \]

where the first second equation is by replacing the 1.

We configure the SACₙ algorithm to use the learning algorithm as follows. First, at iteration \( t \), it invokes the learning algorithm \( L_{sphere} \) as that used in the SAC₁ algorithm for the Sphere Function class to produce a hypothesis \( \tilde{h}_t \). It then produces a hypothesis \( h_t \) by randomly turning a ratio of the positive area of \( \tilde{h}_t \) into negative, which can be done by intersecting \( \tilde{h}_t \) with a random mask, so that \(|D_{h_t}| \leq \beta |D_{\alpha_{t+1}}| \). We assume the intersecting is feasible. So that when the error-successive-level independence condition holds for \( \tilde{h}_t \), it also holds for \( h_t \). Note that this intersecting will increase the error of \( h_t \) from \( \tilde{h}_t \) by at most \( \frac{|D_{\alpha_{t+1}}|}{|X|} \) under the uniform distribution, since the error of \( \tilde{h}_t \) under uniform distribution \( \tilde{\epsilon}_{\tilde{t}, t} \) \( X \) = \( |D_{\tilde{h}_t} \cup D_{\alpha_t}| - |D_{\tilde{h}_t} \cap D_{\alpha_t}| \geq |D_{h_t} \cup D_{\alpha_t}| - |D_{h_t} \cap D_{\alpha_t}| - |(D_{\tilde{h}_t} \setminus D_{h_t}) \cap D_{\alpha_t}| \geq \epsilon_{\tilde{t}, t} \) \( X \) - \( |D_{\alpha_t}| \). Thus when \( \frac{|D_{\alpha_{t+1}}|}{|X|} \) is smaller than a constant, the error increases only a constant. We then use the SACₙ with this configuration on the Sphere Function class, on which SAC₁ algorithms bear a super-polynomial PAA complexity, and obtain Proposition 5.

**Proposition 5**

For any function in \( \mathcal{F}_{sphere}^\alpha \) and any approximation level \( \alpha^* > 0 \), SACₙ algorithms can achieve the PAA query complexity,

\[ O \left( \frac{1}{\alpha^*} \ln \frac{1}{\delta} \right) \cdot \text{poly} \left( \frac{1}{\alpha^*}, n \right) \]

with probability at least \( 1 - \delta \).

**Proof.** By Lemma 6

\[ \frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{1}{\beta} \cdot \frac{|D_{\alpha^*}|(1 - \epsilon_{\tilde{t}, t})^2}{|D_{\alpha_{t+1}}|} \geq \frac{1}{\beta} \cdot \frac{|D_{\alpha^*}|(1 - |D_{\alpha^*} \Delta D_{h_t}|)^2}{|D_{\alpha_{t+1}}|} \geq \frac{1}{\beta} \cdot \frac{|D_{\alpha^*}|(1 - |D_{\alpha^*} \Delta D_{h_t}|)^2}{|D_{\alpha_{t+1}}|}, \]

where \( \epsilon_{\tilde{t}, t} \) and \( \epsilon_{D_t} \) are the errors of \( \tilde{h}_t \) (i.e., the hypothesis before intersecting with the mask) under the uniform and the training distribution, respectively.

Let \( \alpha_t = \frac{1}{2^t} \) for all \( t \), and use the number of iterations \( T \) to approximate \( |D_{\alpha_{T+1}}| = |D_{\alpha^*}| \), for the approximation level \( \alpha^* \). Solving this equation with the sphere volume results in \( T = \log \left( \frac{C_0}{\alpha^*} \right)^2 - 1 \). We let the SACₙ algorithm run \( T = \log \frac{1}{\alpha^*} - 1 \) number of iterations. We assume \( \log \frac{1}{\alpha^*} \) is an integer for simplicity, which does not affect the generality.
In iteration $t$, using $L_{sphere}$, we want the error of the hypothesis $\hat{h}_t, \bar{\epsilon}_{D_t}$, to be a constant $\frac{1}{8}$. Since the $L_{sphere}$ produces a hypothesis with zero training error, $\bar{\epsilon}_{D_t}$ requires a constant number of samples, denote as $M$. We thus obtain $m_0 + \sum_{t=1}^{T} m_t = O \left( \log \frac{1}{\alpha^*} \right)$. We also notice that for any $t \geq 9$, $|D_{\alpha_t}| \leq \frac{1}{8}$.

We then follow Eq. (3.1). We use uniform sampling within $D_{h_t}$, then $Q = 1 - \frac{1}{\lambda}$. Letting the SAC algorithm use $M$ number of samples in every iteration and $\lambda = 0.5$, we have

$$P_{T_h} \geq \frac{1}{\log \frac{1}{\alpha^*} - 1} \sum_{t=1}^{\log \frac{1}{\alpha^*} - 1} \left( \frac{|D_{\alpha_t}|(1 - Q\epsilon_{D_t} - |D_{\alpha_t}|)}{\beta|D_{\alpha_{t+1}}|} \right)^2$$

$$\geq \frac{1}{\log \frac{1}{\alpha^*} - 1} \sum_{t=9}^{\log \frac{1}{\alpha^*} - 1} \left( \frac{|D_{\alpha_t}|(1 - Q\epsilon_{D_t} - \frac{1}{8})}{\beta|D_{\alpha_{t+1}}|} \right)^2$$

$$= \frac{C_0(\alpha^*)^{\frac{3}{2}}}{\log \frac{1}{\alpha^*}} \frac{1}{4\beta} \sum_{t=9}^{\log \frac{1}{\alpha^*} - 1} \frac{1}{|D_{\alpha_{t+1}}|}$$

$$= \frac{C_0(\alpha^*)^{\frac{3}{2}}}{\log \frac{1}{\alpha^*}} \frac{1}{4\beta} \left( \frac{1}{C_0(\frac{1}{2^{t+1}})^{\frac{3}{2}}} \right)$$

$$= \Omega \left( \frac{1}{\beta \log \frac{1}{\alpha^*}} \right),$$

where the last inequality is by considering only $t = T$.

So we obtain the query complexity from Theorem 1, choosing $\beta = poly \left( \frac{1}{\alpha^*}, n \right)$ for the SAC algorithm.

$$O \left( \beta \log \frac{1}{\alpha^*} \ln \frac{1}{\delta} \right) = O \left( \log \frac{1}{\alpha^*} \ln \frac{1}{\delta} \right) \cdot poly \left( \frac{1}{\alpha^*}, n \right),$$

which proves the proposition.

Proposition 5 shows a super-polynomial improvement from the complexity of the uniform search. It is interesting to note that we only ask for a constant learning error in the proof of Proposition 5. This implies that a good learning algorithm may not be necessary for a good SAL algorithm.

4. Discussions and Conclusions

This paper describes the sampling-and-learning (SAL) framework which is an abstract summarization of a range of EAs. We show that the SAL framework directly admits a general upper bound on the PA A query complexity, which is the number of fitness evaluations before an approximate solution is found with a probability.

Focusing on SAC algorithms, which are SAL algorithms using classification learning algorithms, we give a more specific performance upper bound, and compare with uniform random search. We find two interesting conditions of SAC algorithms which control their power. Under the error-target independence condition, which assumes the error of the learned classifier in each iteration is independent with the target approximation area,
the SAC algorithms can obtain a polynomial improvement over the uniform search, but not a super-polynomial improvement. We demonstrate the improvement using the Sphere Function class consisting of convex functions as well as the Spike Function class consisting of non-convex functions. Under the error-successive-level independence condition, which assume the error of the learned classifier in each iteration is independent with the successive learning target, the SAC algorithms can obtain a super-polynomial improvement over the uniform search.

In this work we show that the factor of classification error in SAC algorithms has a great impact on the performance, which was as far as we know not touched in previous empirical or theoretical studies. Finding more factors that have a strong effect on the performance of SAL algorithms is an interesting future work. Also, we expect the work could guide the design of novel search algorithms.

In the case study on the Sphere Function class, we find that under the error-successive-level independence condition the super-polynomial improvement is achieved with a constant learning error rate. This implies that an accurate learning algorithm is not necessary for a good SAC algorithm. It is interesting that it was noticed in a recent work [4] that learnable concept is not necessary for the trial-and-error search with a computation oracle. We thus wander if the learnable concept is also not necessary for a good SAC algorithm.

Moreover, in the SAC algorithms analyzed in this paper, the learning algorithm does not utilize the last hypothesis or the last data set. It would be interesting to investigate whether considering them will bring any significant difference.

5. Acknowledgments

to be added...

References

[1] G. Anil and R. P. Wiegand. Black-box search by elimination of fitness functions. In Proceedings of the 10th ACM SIGEVO International Workshop on Foundations of Genetic Algorithms, pages 67–78, Orlando, FL, 2009.

[2] A. Auger and B. Doerr. Theory of Randomized Search Heuristics - Foundations and Recent Developments. World Scientific, Singapore, 2011.

[3] T. Bäck. Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford, UK, 1996.
[4] X. Bei, N. Chen, and S. Zhang. On the complexity of trial and error. In Proceedings of the 45th Annual ACM Symposium on Symposium on Theory of Computing, pages 31–40, Palo Alto, CA, 2013.

[5] E. Benkhelifa, G. Dragffy, A. Pipe, and M. Nibouche. Design innovation for real world applications, using evolutionary algorithms. In 2009 IEEE Congress on Evolutionary Computation, pages 918–924, Trondheim, Norway, 2009.

[6] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. Natural Computing, 1.

[7] T. Chen, J. He, G. Sun, G. Chen, and X. Yao. A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 39(5):1092–1106, 2009.

[8] S. Dasgupta, A. T. Kalai, and C. Monteleoni. Analysis of perceptron-based active learning. Journal of Machine Learning Research, 10:281–299, 2009.

[9] B. Doerr and C. Winzen. Towards a complexity theory of randomized search heuristics: Ranking-based black-box complexity. In A. Kulikov and N. Vereshchagin, editors, Computer Science – Theory and Applications, volume 6651 of Lecture Notes in Computer Science, pages 15–28. Springer Berlin Heidelberg, 2011.

[10] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. Algorithmica, 64:673–697, 2012.

[11] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile. More effective crossover operators for the all-pairs shortest path problem. Theoretical Computer Science, 471:12–26, 2013.

[12] M. Dorigo, V. Maniezzo, and A. Colomi. Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics–Part B, 26(1):29–41, 1996.

[13] S. Droste, T. Jansen, K. Tinnefeld, and I. Wegener. A new framework for the valuation of algorithms for black-box optimization. In Proceedings of the 7th ACM SIGEVO International Workshop on Foundations of Genetic Algorithms, pages 253–270, Torremolinos, Spain, 2002.

[14] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science, 276(1-2):51–81, 2002.

[15] H. Fournier and O. Teytaud. Lower bounds for comparison based evolution strategies using VC-dimension and sign patterns. Algorithmica, 59(3):387–408, 2011.

[16] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. Evolutionary Computation, 18(4):617–633, 2010.
[17] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[18] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.

[19] J. He and X. Yao. An analysis of evolutionary algorithms for finding approximation solutions to hard optimisation problems. In *Proceedings of 2003 IEEE Congress on Evolutionary Computation (CEC’03)*, pages 2004–2010, Canberra, Australia, 2003.

[20] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.

[21] G. S. Hornby, A. Globus, D. S. Linden, and J. D. Lohn. Automated antenna design with evolutionary algorithms. In *Proceedings of 2006 American Institute of Aeronautics and Astronautics Conference on Space*, pages 19–21, San Jose, CA, 2006.

[22] T. Jansen. *Analyzing Evolutionary Algorithms*. Springer-Verlag, Berlin, Germany, 2013.

[23] T. Jansen and I. Wegener. The analysis of evolutionary algorithms – A proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.

[24] T. Jansen and C. Zarges. Fixed budget computations: A different perspective on run time analysis. In *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference*, pages 1325–1332, Philadelphia, PA, 2012.

[25] T. Jansen, K. Jong, and I. Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440, 2005.

[26] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

[27] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN’95)*, volume 4, pages 1942–1948, Perth, Australia, 1995.

[28] J. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

[29] J. R. Koza, M. A. Keane, and M. J. Streeter. What’s AI done for me lately? Genetic programming’s human-competitive results. *IEEE Intelligent Systems*, 18(3):25–31, 2003.

[30] S. Kratsch and F. Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. *Algorithmica*, 65(4):754–771, 2013.
[31] X. Lai, Y. Zhou, J. He, and J. Zhang. Performance analysis of evolutionary algorithms for the minimum label spanning tree problem. *IEEE Transactions on Evolutionary Computation*, in press, 2014.

[32] P. Larrañaga and J. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[33] P. K. Lehre and C. Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642, 2012.

[34] P. K. Lehre and X. Yao. Crossover can be constructive when computing unique input output sequences. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning*, pages 595–604, Melbourne, Australia, 2008.

[35] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity*. Springer-Verlag, Berlin, Germany, 2010.

[36] C. Qian, Y. Yu, and Z.-H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119, 2013.

[37] J. Scharnow, K. Tinnefeld, and I. Wegener. Fitness landscapes based on sorting and shortest paths problems. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN’02)*, pages 54–63, Granada, Spain, 2002.

[38] T. Storch. On the choice of the parent population size. *Evolutionary Computation*, 16(4):557–578, 2008.

[39] D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.

[40] A. M. Sutton and F. Neumann. A parameterized runtime analysis of evolutionary algorithms for the euclidean traveling salesperson problem. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, Toronto, Canada, 2012.

[41] W. Wang and Z.-H. Zhou. Multi-view active learning in the non-realizable case. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2388–2396. Curran Associates, Inc., 2011.

[42] C. Witt. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120, 2008.

[43] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[44] X. Yao. Unpacking and understanding evolutionary algorithms. In J. Liu, C. Alippi, B. Bouchon-Meunier, G. Greenwood, and H. Abbass, editors, *Advances in Computational Intelligence*, volume 7311 of *Lecture Notes in Computer Science*, pages 60–76. Springer Berlin Heidelberg, 2012.
[45] Y. Yu and Z.-H. Zhou. A new approach to estimating the expected first hitting time of evolutionary algorithms. *Artificial Intelligence*, 172(15):1809–1832, 2008.

[46] Y. Yu, X. Yao, and Z.-H. Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, (180-181):20–33, 2012.