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

Yang Yu

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
Email: yuy@lamda.nju.edu.cn

Hong Qian

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
Email: qianh@lamda.nju.edu.cn

Abstract—**Evolutionary algorithms (EAs), a large class of general purpose optimization algorithms inspired from the 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. By summarizing a large range of EAs into the *sampling-and-learning* framework, we show that the framework directly admits a general analysis on the *probable-absolute-approximate* (PAA) query complexity. We particularly focus on the framework with the learning subroutine being restricted as a binary classification, which results in the *sampling-and-classification* (SAC) algorithms. With the help of the learning theory, we obtain 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 SAC algorithms can achieve polynomial speedup to the uniform search, but not super-polynomial speedup. Under the *one-side-error* condition, we show that super-polynomial speedup can be achieved. This work only touches the surface of the framework. Its power under other conditions is still open.**

I. 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 follows the trial-and-error style with some heuristics. Evolutionary algorithms (EAs) [3] are a large family of such algorithms. The family includes genetic algorithms [17], evolutionary programming [26], evolutionary strategies [5], and also covers other nature-inspired heuristics including particle swarm optimization [25], ant colony optimization [11], estimation of distribution algorithms [29], etc.

Theoretical studies of EAs have been developed rapidly in the recent decades, particularly noticeable of the blooming of running time analysis [32], [2], [20]. With the development of several analysis techniques (e.g. [18], [44], [9], [37]), EAs have been theoretically investigated on problems from simple synthetic ones (e.g. [13]) to combinatorial problems (e.g. [35]) as well as NP-hard problems (e.g. [43]). During these analyses, effects of EAs components have been disclosed [42], including the crossover operators (e.g. [22], [31], [8], [33]), the population size (e.g. [21], [36], [40], [6]), etc. Measures of the performance also have developed to cover the approximation complexity (e.g. [19], [16], [43], [28]), the fixed-parameter complexity (e.g. [27], [38]), the complexity under fixed-budget computation [23], etc. While most of these analyses studied instances of EAs on problem cases, general performance

analysis may even be more desired, as the application of EAs is nearly unlimited. The famous No-Free-Lunch Theorem [41] 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 [44]. When the complexity of a problem class is bounded, a general convergence lower bound can be derived for a class of EAs [15]. For more general EAs, the Black-Box model can derive the best possible performance [12], [1], [30], [10]. We have learned that a general performance analysis relies on a general framework of EAs.

It has been noticed that various implementations of EAs share a common structure that consists of a cycle of sampling and model building [47]. In this work, we propose to study the *sampling-and-learning* (SAL) framework. EAs commonly employ some heuristic to reproduce solutions, which is captured by the sampling step of SAL; and 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 captured by the learning step of SAL. The SAL framework can simulate a wide range of EAs as well as other heuristic search methods, by specifying the sampling and the learning strategies.

We evaluate this framework by the probable-absolute-approximate (PAA) query complexity. PAA complexity counts the number of fitness evaluations before reaching to an approximate solution with a probability, which is close to the intuitive evaluation of EAs in practice. We show that the SAL framework immediately admits a general PAA upper bound. For a specific version of SAL that uses classification algorithms, named the SAC algorithms, we obtain a tighter PAA upper bound by incorporating the learning theory results. Further 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 *one-side-error* condition further allows a super-polynomial improvement. This study shows that the classification error is an important effecting factor, 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.

II. The Sampling-and-Learning Framework

In this paper, we consider general minimization problems f. We always denote X as the whole solution space which an algorithm will search among. In the analysis of this paper, we consider $X \subseteq \mathbb{R}^n$ is a compact set (in the Euclidean space, the compact set is equivalent to the bounded and closed set) and $f : X \to \mathbb{R}$ is a continuous function. Thus there must exist at least one solution $x^* \in X$ such that $f(x^*) = \min_{x \in X} f(x)$. We use D to denote sub-regions of X and define $|D| = \int_D 1 \mathrm{d}x$. For the sake of convenience for the analysis, we assume without loss of generality that $|X| = 1$ since X is a bounded and closed set. Denote $D_\alpha = \{x \in X | f(x) \le \alpha\}$ for any scaler α, $\mathcal{U}_X$ as the uniform distribution over X, $\mathcal{T}$ and $\mathcal{D}$ as the probability distributions. Besides, by $\mathrm{poly}(\cdots)$, we mean the set of all polynomials with the related variables, and by $\mathrm{superpoly}(\cdots)$, we mean the set of all functions that grow faster than any function in $\mathrm{poly}(\cdots)$ with the related variables.

Definition 1 (Minimization Problem) *A minimization problem consists of a continuous solution space X and a continuous function $f : X \to \mathbb{R}$, where $X \subseteq \mathbb{R}^n$ and X is a compact set. The goal is to find a solution $x^* \in X$ such that $f(x^*) \le f(x)$ for all $x \in X$.*

Since X is a compact set and f is a continuous function, there must exist one solution $x' \in X$ such that $f(x') = \max_{x \in X} f(x)$. Namely, f is bounded in $[f(x^*), f(x')]$. Therefore, 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 $f(x) = \frac{g(x) - g(x^*)}{\max_{x'} g(x') - g(x^*)}$. Thus we assume in the rest of this paper that every minimization problem has its minimum value 0.

In real-world applications, we expect EAs to achieve some good enough solutions with a not quite small probability, which corresponds to approximation (e.g. [43]) and probabilistic performance (e.g. [45]). Combining the two, we study 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 practice.

Definition 2 (Probable-Absolute-Approximate Query Complexity) *Given a minimization problem f, an algorithm $\mathcal{A}$, and any $0 < \delta < 1$ as well as any approximation level $\alpha^* > 0$, then the probable-absolute-approximate (PAA) query complexity is the number of calls to $f(\cdot)$ such that, with probability at least $1 - \delta$, $\mathcal{A}$ finds a solution x with $f(x) \le \alpha^*$.*

A. The General Framework

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

   a)   directly access the solution 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.

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: $\tilde{x} = \mathrm{argmin}_{x \in S_0} f(x)$
3: Initialize the 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 $\begin{cases} \mathcal{T}_{h_t}, & \text{with probability } \lambda \\ \mathcal{U}_X, & \text{with probability } 1 - \lambda \end{cases}$
11:         $S_t = S_t \cup \{x_i\}$
12:     **end for**
13:     $\tilde{x} = \mathrm{argmin}_{x \in S_t \cup \{\tilde{x}\}} f(x)$
14: **end for**
15: **return** $\tilde{x}$

We present a sampling-and-learning (SAL) framework in Algorithm 1 to capture these properties. The SAL framework starts from a random sampling in Step 1 like all EAs. Steps 2 and 13 record the best-so-far solutions throughout the search. SAL follows a cycle 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 $\mathcal{L}$. Note that 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. 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 the previous iteration. In Steps 9 to 12, it samples from the distribution transformed from the hypothesis as well as from the whole solution space balanced by a probability. The distribution $\mathcal{T}_{h_t}$ 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 following 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) [17] deal with discrete solution spaces consisting 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. Converting this operation probability to the probability of generating a certain solution, let $P_m(x'|x)$ be the probability of generating the solution

x' from x via the element-wise mutation, thus $P_m(x'|x) = (\frac{p}{|V|-1})^{\|x'-x\|_H}(1-p)^{n-\|x'-x\|_H}$, where n is the length of the solution, $|V|$ is the vocabulary size, $\|\cdot\|_H$ is the Hamming distance, and p is the probability of changing the element that is commonly $\frac{1}{n}$. It is easy to calculate that $P_m(x'|x)$ is $\frac{1}{\text{poly(n)}}$ only when $\|x'-x\|_H$ is a constant (and otherwise $P_m(x'|x) = \frac{1}{\text{superpoly(n)}}$). Given any set of solutions $S = \{x_1, x_2, \ldots, x_m\}$, we divide the search space into two sets that $X_{poly}(S) = \{x \in X \mid \exists x' \in S : \|x - x'\|_H = O(1)\}$ and $X_{super}(S) = X - S_{poly}(S)$. SAL can simulate the GA as that, for every population S of the GA, SAL learns the hypothesis h that circles the area $X_{poly}(S)$, and uses $\mathcal{T}_h$ as $\mathcal{T}_h(x) = \frac{\sum_{x \in S} P_m(x'|x)}{\sum_{x'' \in X_{poly}(S)} \sum_{x \in S} P_m(x''|x)}$ for solutions in $X_{poly}(S)$. And for the area $X_{super}(S)$, SAL uses the uniform distribution to approximate the sampling with super-polynomially small probability. In this way, SAL can mimic the behavior of the GA. We have discussed a simplified GA. 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. Many GAs also employ a probabilistic selection, which can be simulated by selecting the initial solution set S_t in the same way.

It has been argued that model-based search algorithms including the estimation of distribution algorithms (EDAs) [29], the ant colony optimization algorithms (ACOs) [11], the cross-entropy method [34] can be unified in the sampling and model building framework [47], which respectively correspond to the sampling and learning steps in the SAL framework. The particle swarm optimization algorithms (PSOs) [25] is particularly interesting since the simulation is perhaps the most sophisticated. A PSO algorithm maintains a set of "flying" particles each with a location (representing a solution) and a velocity vector. 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 needs to use the initial hypothesis resulting the same sampling distribution as that 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 probability at least $1 - \delta$, a SAL algorithm will output a solution x with $f(x) \leq \alpha^*$ using m_Σ number of queried samples bounded from above by*

$$O\left(m_0 + \max\left\{\frac{1}{(1-\lambda)\mathbf{Pr}_u + \lambda\overline{\mathbf{Pr}}_h}\ln\frac{1}{\delta}, \sum_{t=1}^{T} m_{\mathbf{Pr}_{h_t}}\right\}\right),$$

where $\mathbf{Pr}_u = \int_{D_{\alpha^*}} \mathcal{U}_X(x)\,dx$ is the success probability of uniform sampling,

$$\overline{\mathbf{Pr}}_h = \frac{\sum_{t=1}^{T} m_t \cdot \mathbf{Pr}_{h_t}}{\sum_{t=1}^{T} m_t} = \frac{\sum_{t=1}^{T} m_t \cdot \int_{D_{\alpha^*}} \mathcal{T}_{h_t}(x)\,dx}{\sum_{t=1}^{T} m_t}$$

is the average success probability of sampling from the learnt hypothesis, $m_{\mathbf{Pr}_{h_t}}$ is the required sample size realizing $\mathbf{Pr}_{h_t}$, and $D_{\alpha^*} = \{x \in X | f(x) \leq \alpha^*\}$.

Proof. m_0 is the initial sample size. In every iteration, we need $m_{\mathbf{Pr}_{h_t}}$ samples to realize the probability $\mathbf{Pr}_{h_t}$ (generally the higher the probability the larger the sample size, but it depends on the concrete implement of the algorithm), thus $\sum_{t=1}^{T} m_{\mathbf{Pr}_{h_t}}$ number of samples is naturally required. We prove the rest 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 intersection 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 $1 - \mathbf{Pr}_u$.
2. For the sampling from the learnt hypothesis h_t according to the distribution $\mathcal{T}_{h_t}$, the probability of failure is denoted as $1 - \mathbf{Pr}_{h_t}$.
Since every sampling is independent, we can expand the probability of overall failures, i.e., for any solution x belongs to the all sampled examples,

$$\mathbf{Pr}(f(x) > \alpha^*)$$
$$= (1 - \mathbf{Pr}_u)^{m_0} \cdot$$
$$\prod_{t=1}^{T} \sum_{i=0}^{m_t} \binom{m_t}{i}(1-\lambda)^i \lambda^{m_t - i}(1-\mathbf{Pr}_u)^i(1-\mathbf{Pr}_{h_t})^{m_t - i}$$
$$= (1 - \mathbf{Pr}_u)^{m_0} \prod_{t=1}^{T} \left(1 - (1-\lambda)\mathbf{Pr}_u - \lambda\mathbf{Pr}_{h_t}\right)^{m_t}$$
$$\leq e^{-\mathbf{Pr}_u \cdot m_0} \prod_{t=1}^{T} e^{-\left((1-\lambda)\mathbf{Pr}_u m_t + \lambda\mathbf{Pr}_{h_t} m_t\right)}$$
$$= e^{-\left(\mathbf{Pr}_u \cdot m_0 + (1-\lambda)\sum_{t=1}^{T}\mathbf{Pr}_u m_t + \lambda\sum_{t=1}^{T}\mathbf{Pr}_{h_t} m_t\right)}$$
$$\leq e^{-\left((1-\lambda)\sum_{t=1}^{T}\mathbf{Pr}_u m_t + \lambda\sum_{t=1}^{T}\mathbf{Pr}_{h_t} m_t\right)}$$
$$= e^{-\left((1-\lambda)\mathbf{Pr}_u + \lambda\overline{\mathbf{Pr}}_h\right)\sum_{t=1}^{T} m_t},$$

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

In order that $\mathbf{Pr}(f(x) > \alpha^*) < \delta$, we let $e^{-\left((1-\lambda)\mathbf{Pr}_u + \lambda\overline{\mathbf{Pr}}_h\right)\sum_{t=1}^{T} m_t} < \delta$, which solves that $\sum_{t=1}^{T} m_t = O\left(\frac{1}{(1-\lambda)\mathbf{Pr}_u + \lambda\overline{\mathbf{Pr}}_h}\ln\frac{1}{\delta}\right)$. ∎

B. The Sampling-and-Classification Algorithms

To further unfold the unknown term $\overline{\mathbf{Pr}}_h$ in Theorem 1, we focus on 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, and then invokes the classification algorithm to learn from the binary data set. sign$[\cdot]$ is defined as sign$[v] = +1$ if $v \geq 0$ and -1 if $v < 0$. Note that SAC algorithms use the current

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 threshold parameters
    $\mathcal{C}$: Classification algorithm
Procedure:
  1: Construct $B = \{(x_1, z_1), \ldots, (x_{|T|}, z_{|T|})\}$ from that, for all i and all $(x_i, y_i) \in T$, $z_i = \text{sign}[\alpha_t - y_i]$
  2: $h = \mathcal{C}(B)$
  3: **return** h

---

data set T in the learning algorithm, but not the last data set T' and the last hypothesis h'. Putting Algorithm 2 into the framework of Algorithm 1, we always set $S_t = \emptyset$ for SAC, and $\mathcal{T}_h$ will be some distribution over the positive area of h.

By these specifications, we can have a general PAA performance for SAC algorithms. According to Theorem 1, we need to estimate a lower bound of $\overline{\mathbf{Pr}}_h$, i.e., how likely the distribution $\mathcal{T}_{h_t}$ will lead to a good solution. Recall $D_\alpha = \{x \in X | f(x) \leq \alpha\}$ for any scaler $0 < \alpha < 1$. Denote $D_h = \{x \in X | h(x) = +1\}$ for any hypothesis h, $\mathcal{U}_{D_h}$ as the uniform distribution over D_h, and D_{KL} as the Kullback-Leibler (KL) divergence. KL-divergence measures how difference one distribution departs from another one. For probability distributions P and Q of two continuous random variables, $D_{KL}(P \| Q) = \int_{-\infty}^{+\infty} \ln\left(\frac{p(x)}{q(x)}\right) p(x) \mathrm{d}x$, where $p(x)$ and $q(x)$ are the probability densities of P and Q. Let Δ denote the symmetric difference operator of two sets. We have a lower bound of the success probability as in Lemma 1.

Lemma 1 *For any minimization problem f, any approximation level $\alpha^* > 0$, any hypothesis h, the probability that a solution sampled from an arbitrary distribution $\mathcal{T}_h$ defined on D_h will lead to a solution in D_{α^*} is lower bounded as*

$$\mathbf{Pr}_h \geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - |D_{\alpha^*} \cap D_h| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_h \| \mathcal{U}_{D_h})}$$

Proof. Let $I[\cdot]$ denote the indicator function, namely, $I[\texttt{true}] = 1$ and $I[\texttt{false}] = 0$. The proof starts from the definition of the probability,

$$\mathbf{Pr}_h = \int_{D_h} \mathcal{T}_h(x) \cdot I[x \in D_{\alpha^*}] \mathrm{d}x$$
$$= \int_{D_h} (\mathcal{T}_h(x) - \mathcal{U}_{D_h}(x) + \mathcal{U}_{D_h}(x)) \cdot I[x \in D_{\alpha^*}] \mathrm{d}x$$
$$= \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} + \int_{D_h} (\mathcal{T}_h(x) - \mathcal{U}_{D_h}(x)) \cdot I[x \in D_{\alpha^*}] \mathrm{d}x$$
$$\geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - \int_{D_h} \sup_{x'} |\mathcal{T}_h(x') - \mathcal{U}_{D_h}(x')| \cdot I[x \in D_{\alpha^*}] \mathrm{d}x$$
$$\geq \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_h \| \mathcal{U}_{D_h})} \int_{D_h} I[x \in D_{\alpha^*}] \mathrm{d}x$$
$$= \frac{|D_{\alpha^*} \cap D_h|}{|D_h|} - |D_{\alpha^*} \cap D_h| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_h \| \mathcal{U}_{D_h})},$$

where the last inequality is by Pinsker's inequality. ∎

We cannot pre-determine D_h, but we know that h is derived by a binary classification algorithm from a data set which is labeled according to the threshold parameter α. 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 [24], as in Lemma 2. The $VC(\mathcal{H})$ is the VC-dimension measuring the complexity of $\mathcal{H}$.

Lemma 2 ([24]) *Let $\mathcal{H} = \{h : X \to \{-1, +1\}\}$ be the hypothesis space containing a family of binary classification functions and $VC(\mathcal{H}) = d$, 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 $1 - \eta$:*

$$\epsilon_{\mathcal{D}} \leq \hat{\epsilon}_{\mathcal{D}} + \sqrt{8m^{-1}\left(d \log\left(2emd^{-1}\right) + \log\left(4\eta^{-1}\right)\right)}$$

where $\epsilon_{\mathcal{D}}$ is the expected error rate of h over $\mathcal{D}$ and $\hat{\epsilon}_{\mathcal{D}}$ is the error rate in the sampled examples from $\mathcal{D}$, and when $\hat{\epsilon}_{\mathcal{D}} = 0$,

$$\epsilon_{\mathcal{D}} \leq 2m^{-1}\left(d \log\left(2emd^{-1}\right) + \log\left(2\eta^{-1}\right)\right).$$

Again by Pinsker's inequality, we know that the error $\epsilon_{\mathcal{D}}$ under the distribution $\mathcal{D}$ can be converted to the error $\epsilon_{\mathcal{U}}$ under the uniform distribution, as

$$\epsilon_{\mathcal{U}} \leq \frac{\epsilon_{\mathcal{D}}}{1 - \sqrt{\frac{1}{2} D_{KL}(\mathcal{D} \| \mathcal{U})}}$$
$$\leq \frac{\hat{\epsilon}_{\mathcal{D}} + \sqrt{8m^{-1}\left(d \log\left(2emd^{-1}\right) + \log\left(4\eta^{-1}\right)\right)}}{1 - \sqrt{\frac{1}{2} D_{KL}(\mathcal{D} \| \mathcal{U})}},$$

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

We can use this inequality to eliminate the D_h in Lemma 1. In every iteration of SAC algorithms, there are m_t samples collected, which make the error of h_t bounded.

Theorem 2 *For any minimization problem f, any constant $0 < \eta < 1$, and any approximation level $\alpha^* > 0$, the average success probability of sampling from the learnt hypothesis of any SAC algorithm is lower bounded as*

$$\overline{\mathbf{Pr}}_h \geq \frac{1 - \eta}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \left(\frac{|D_{\alpha^*}| - 2\Psi^{m_t, \eta}_{\hat{\epsilon}_{\mathcal{D}_t}, d, D_{KL}(\mathcal{D}_t \| \mathcal{U}_X)}}{|D_{\alpha_t}| + \Psi^{m_t, \eta}_{\hat{\epsilon}_{\mathcal{D}_t}, d, D_{KL}(\mathcal{D}_t \| \mathcal{U}_X)}} \right.$$
$$\left. - |D_{\alpha^*}| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})} \right),$$

where $\mathcal{D}_t = \lambda \mathcal{T}_{h_t} + (1 - \lambda)\mathcal{U}_X$ is the sampling distribution at iteration t, $\hat{\epsilon}_{\mathcal{D}_t}$ is the training error rate of h_t, d is the VC-dimension of the learning algorithm.

Proof. By set operators,

$$|D_{\alpha^*} \cap D_{h_t}| = |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{h_t}|$$
$$\geq |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{\alpha_t}| - |D_{\alpha_t} \Delta D_{h_t}|$$
$$= |D_{\alpha^*} \cup D_{h_t}| - |D_{\alpha^*} \Delta D_{\alpha_t}| - \epsilon_{\mathcal{U}_X, t}$$

$$= |D_{\alpha^*} \cup D_{h_t}| + |D_{\alpha^*}| - |D_{\alpha_t}| - \epsilon_{\mathcal{U}_X, t},$$

where Δ is the symmetric difference operator of two sets and $\epsilon_{\mathcal{U}_X, t}$ is the expected error rate of h_t under $\mathcal{U}_X$. The first inequality is by the triangle inequality, and the last equation is by that D_{α^*} is contained in D_{α_t}.

Since $\left| |D_{h_t}| - |D_{\alpha_t}| \right| \leq |D_{h_t} \Delta D_{\alpha_t}| = \epsilon_{\mathcal{U}_X, t}$, we can bound $|D_{h_t}|$ as $|D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t} \geq |D_{h_t}| \geq |D_{\alpha_t}| - \epsilon_{\mathcal{U}_X, t}$.

Now, we can apply Lemma 1, and the success probability of sampling from D_{h_t} is lower bounded as

$$
\begin{aligned}
\mathbf{Pr}_{h_t} &\geq \frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} - |D_{\alpha^*} \cap D_{h_t}| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})} \\
&\geq \frac{1}{|D_{h_t}|} \cdot (|D_{\alpha^*} \cup D_{h_t}| + |D_{\alpha^*}| - |D_{\alpha_t}| - \epsilon_{\mathcal{U}_X, t}) \\
&\quad - |D_{\alpha^*}| \cdot \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})} \\
&\geq \frac{1}{|D_{h_t}|} \cdot (|D_{h_t}| + |D_{\alpha^*}| - |D_{\alpha_t}| - \epsilon_{\mathcal{U}_X, t}) \\
&\quad - |D_{\alpha^*}| \cdot \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})} \\
&\geq \frac{|D_{\alpha^*}| - 2\epsilon_{\mathcal{U}_X, t}}{|D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t}} - |D_{\alpha^*}| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})}.
\end{aligned}
$$

Substituting this lower bound and the probability $1 - \eta$ of the generalization bound into $\overline{\mathbf{Pr}}_h$ obtains the theorem. ∎

Combining Theorem 1 and Theorem 2 results an upper bound on the sampling complexity of SAC algorithms. Although the expression looks sophisticated, it can still reveal relative variables that generally effect the complexity. One could design various distributions for $\mathcal{T}_h$ to sample potential solutions, however, without any a priori knowledge, the uniform sampling is the best in terms of the 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 can also improve the performance.

III. 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 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 the 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

$$\Theta\left(\frac{1}{\mathbf{Pr}_u} \cdot \ln \frac{1}{\delta} \right).$$

Contrasting this with Theorem 1, we can find that how much a SAC algorithm improves from the uniform search depends on the average success probability $\overline{\mathbf{Pr}}_h$ that relies on the learnt hypothesis. A SAC algorithm is not always better than the uniform search. Without any restriction, $\overline{\mathbf{Pr}}_h$ can be zero and thus the SAC algorithm is worse. We are then interested in investigating the conditions under which SAC algorithms can accelerate from the uniform search.

A. 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. From the set perspective, it is equivalent with

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

Under the condition, we can bound from below the probability of sampling a good solution, as stated in Lemma 3.

Lemma 3 *For SAC$_I$ algorithms, it holds for all t that*

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{|D_{\alpha^*}|(1 - \epsilon_{\mathcal{U}_X, t})}{|D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t}},$$

where $\epsilon_{\mathcal{U}_X, t}$ is the expected error rate of h_t under $\mathcal{U}_X$.

Proof. For the numerator,

$$
\begin{aligned}
|D_{\alpha^*} \cap D_{h_t}| &= |D_{\alpha^*}| - |D_{\alpha^*} \cap (D_{\alpha_t} \Delta D_{h_t})| \\
&= |D_{\alpha^*}| - |D_{\alpha^*}| \cdot |D_{\alpha_t} \Delta D_{h_t}| \\
&\geq |D_{\alpha^*}|(1 - \epsilon_{\mathcal{U}_X, t}),
\end{aligned}
$$

where the first equation is by $D_{\alpha^*} \subseteq D_{\alpha_t}$, and 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 $|D_{h_t}| \leq |D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t}$. ∎

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

$$
\begin{aligned}
\overline{\mathbf{Pr}}_h \geq \frac{1 - \eta}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \bigg(&\frac{|D_{\alpha^*}|(1 - \epsilon_{\mathcal{U}_X, t})}{|D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t}} \\
&- |D_{\alpha^*}| \sqrt{\frac{1}{2} D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}})} \bigg).
\end{aligned}
$$

We compare the uniform search with the SAC$_I$ algorithms using uniform sampling within D_{h_t}, i.e., $D_{KL}(\mathcal{T}_{h_t} \| \mathcal{U}_{D_{h_t}}) = 0$, which is an optimistic situation. Then by Lemma 3,

$$\overline{\mathbf{Pr}}_h \geq \frac{1 - \eta}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \bigg(\frac{|D_{\alpha^*}|(1 - \epsilon_{\mathcal{U}_X, t})}{|D_{\alpha_t}| + \epsilon_{\mathcal{U}_X, t}} \bigg).$$

By plugging $\epsilon_{\mathcal{U}_X, t} \leq \frac{\epsilon_{\mathcal{D}_t}}{1 - \sqrt{\frac{1}{2} D_{KL}(\mathcal{D}_t \| \mathcal{U}_X)}} = Q \cdot \epsilon_{\mathcal{D}_t}$, where $\epsilon_{\mathcal{D}_t}$ is the expected error rate of h_t under the distribution $\mathcal{D}_t = \lambda \mathcal{U}_{D_{h_t}} + (1 - \lambda)\mathcal{U}_X$ and $Q = (1 - \sqrt{\frac{1}{2} D_{KL}(\mathcal{D}_t \| \mathcal{U}_X)})^{-1}$,

$$\overline{\mathbf{Pr}}_h \geq \frac{1 - \eta}{\sum_{t=1}^{T} m_t} \sum_{t=1}^{T} m_t \bigg(\frac{|D_{\alpha^*}|(1 - Q \cdot \epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}| + Q \cdot \epsilon_{\mathcal{D}_t}} \bigg). \quad (1)$$

Note from Lemma 2 that, the convergence rate of the error is $\tilde{O}(\frac{1}{m})$ ignoring other variables and logarithmic terms from Lemma 2. We assume that SAC$_I$ uses learning algorithms

with convergence rate $\tilde{\Theta}(\frac{1}{m})$. We then find that such SAC_I algorithms cannot exponentially improve the uniform search in the worst case, as Proposition 1.

Proposition 1 *Using learning algorithms with convergence rate $\tilde{\Theta}(\frac{1}{m})$, $\forall f$, $\alpha^* > 0$ and $0 < \delta < 1$, with probability at least $1 - \delta$, if the query complexity of the uniform search is $\texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})$, the query complexity of SAC_I algorithms is also $\texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})$ in the worst case.*

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

$$\frac{1}{\mathbf{Pr}_u} = \frac{1}{|D_{\alpha^*}|} = \texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta}).$$

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

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

Following the optimistic case of Eq.(1), since $Q \geq 1$, we consider one more optimistic situation that $Q = 1$. Let $\eta = 0.5$. Even though, in the worst case that $|D_{h_t}| = |D_{\alpha_t}| + Q\epsilon_{\mathcal{D}_t}$, we can have that

$$\overline{\mathbf{Pr}}_h = \frac{1}{2\sum_{t=1}^T m_t} \sum_{t=1}^T m_t \left(\frac{|D_{\alpha^*}|(1 - \epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}| + \epsilon_{\mathcal{D}_t}} \right)$$

$$= \frac{1}{\texttt{poly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})} \texttt{poly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta}) \frac{\frac{1}{\texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})}}{\frac{1}{\texttt{poly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})}}$$

$$= \frac{1}{\texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})},$$

where it is noted that as long as $\epsilon_{\mathcal{D}_t} = \texttt{poly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})$ the value of $|D_{\alpha_t}|$ cannot affect the result. Then substituting $\overline{\mathbf{Pr}}_h$ into Theorem 1 obtains the total samples $m_\Sigma = \texttt{superpoly}(\frac{1}{\alpha^*}, n, \frac{1}{\delta})$ that proves the proposition. ∎

The proposition implies that the SAC_I algorithms can face the same barrier as that of the uniform search. Nevertheless, the SAC_I 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 $\mathcal{F}_{sphere}^n = \{f_{sphere}^{x^*, n} | \forall x^* \in X_n\}$ where

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

Obviously, $|X_n| = 1$, $f_{sphere}^{x^*, n} \in [0, 1]$ is convex, and the optimal value is 0. It is important to notice that the volume of a n-dimensional hyper-sphere with radius r is $\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^n$, where $\Gamma(s) = \int_0^\infty t^{s-1} e^{-t} dt$, so that $|D_\alpha| = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} (n\alpha)^{n/2} =$

$C_n(\alpha)^{n/2}$ for any $\alpha > 0$, where $C_n = \Theta\left((2\pi e)^{\frac{n}{2}} / \sqrt{\pi n}\right)$, since the radius leading to $f_{sphere}^{x^*, n}(x) = \frac{1}{n} \|x - x^*\|_2^2 \leq \alpha$ is $\sqrt{n\alpha}$.

Note that $\mathbf{Pr}_u = |D_{\alpha^*}| = C_n(\alpha^*)^{n/2} > (\alpha^*)^{n/2}$. It is straightforward to obtain that, minimizing any function in $\mathcal{F}_{sphere}^n$ using the uniform search, the PAA query complexity with approximation level $\alpha^* > 0$ is, with probability at least $1 - \delta$,

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

We assume $\mathcal{L}_{sphere}$ is a learning algorithm that 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, and meanwhile the sphere satisfies the error-target independence condition. Then a SAC algorithm using $\mathcal{L}_{sphere}$ is a SAC_I algorithm. We simply assume that the search of the consistent sphere is feasible. Note that $VC(\mathcal{H}_n) = n + 1$.

Lemma 4 *For any h_t, denote $\epsilon_{\mathcal{U}_X}$ as the error rate of h_t under the uniform distribution over X and $\epsilon_{\mathcal{D}_t}$ as the error rate of h_t under the distribution $\mathcal{D}_t = \lambda \mathcal{U}_{D_{h_t}} + (1 - \lambda)\mathcal{U}_X$, then it holds that*

$$\epsilon_{\mathcal{U}_X} \leq \frac{1}{1 - \lambda} \epsilon_{\mathcal{D}_t},$$

where $\lambda \in [0, 1]$ and $\mathcal{U}_{D_{h_t}}$ is the uniform distribution over D_{h_t}.

Proof. Let $I[\cdot]$ be the indicator function and $D_{\neq}$ be the area where h_t makes mistakes. We split $D_{\neq}$ into $D_{\neq}^+ = D_{\neq} \cap D_{h_t}$ and $D_{\neq}^- = D_{\neq} \setminus D_{\neq}^+$. We can calculate the probability density that $\mathcal{D}_t(x) = \lambda \frac{1}{|D_{h_t}|} + (1 - \lambda) \frac{|D_{h_t}|}{|X|} \frac{1}{|D_{h_t}|}$ for any $x \in D_{\neq}^+$, and $\mathcal{D}_t(x) = (1 - \lambda) \frac{|X \setminus D_{h_t}|}{|X|} \frac{1}{|X \setminus D_{h_t}|} = (1 - \lambda) \frac{1}{|X|}$ for any $x \in D_{\neq}^-$. Thus,

$$\epsilon_{\mathcal{D}_t} = \int_X \mathcal{D}_t(x) I[h_t \text{ makes mistake on } x] dx$$

$$= \int_{D_{\neq}} \mathcal{D}_t(x) dx = \int_{D_{\neq}^+} \mathcal{D}_t(x) dx + \int_{D_{\neq}^-} \mathcal{D}_t(x) dx$$

$$\geq \int_{D_{\neq}^+} (1 - \lambda) \frac{1}{|X|} dx + \int_{D_{\neq}^-} (1 - \lambda) \frac{1}{|X|} dx$$

$$= (1 - \lambda) \epsilon_{\mathcal{U}_X},$$

which proves the lemma. ∎

We then obtain the PAA complexity as in Proposition 2.

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

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

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 $|D_{\alpha_T}| = \sqrt{|D_{\alpha^*}|}$, for the approximation level α^*. Solving this equation with the sphere volume results in $T = \log \frac{(C_n)^{\frac{1}{n}}}{\sqrt{\alpha^*}}$. We let the SAC_I 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 $\mathcal{L}_{sphere}$, we want the error of the hypothesis h_t, $\epsilon_{\mathcal{D}_t}$, to be $\frac{1}{2^t}$. Since the $\mathcal{L}_{sphere}$ produces a hypothesis with zero training error, from

$$\epsilon_{\mathcal{D}_t} = \frac{1}{2^t} \leq 2m^{-1}\left(d\log\left(2emd^{-1}\right) + \log\left(2\eta^{-1}\right)\right),$$

we can solve the required sample size with η being a constant,

$$m_t \leq m_T = O(nT2^T) = O\left(\frac{n}{\sqrt{\alpha^*}}\log\frac{1}{\sqrt{\alpha^*}}\right)$$

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

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

$$\overline{\mathbf{Pr}}_h \geq \frac{1}{2\log\frac{1}{\sqrt{\alpha^*}}}\sum_{t=1}^{\log\frac{1}{\sqrt{\alpha^*}}}\left(\frac{|D_{\alpha^*}|(1-Q\epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}| + Q\epsilon_{\mathcal{D}_t}}\right)$$

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

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

$$= \frac{C_n(\alpha^*)^{\frac{n}{2}}}{2\log\frac{1}{\sqrt{\alpha^*}}}\frac{(\frac{1}{\sqrt{\alpha^*}} - 2)}{(C_n+2)} = \Omega\left(\frac{(\alpha^*)^{\frac{n-1}{2}}}{\log\frac{1}{\sqrt{\alpha^*}}}\right).$$

So we obtain the query complexity from Theorem 1

$$O\left(m_0 + \max\left\{(\frac{1}{\alpha^*})^{\frac{n-1}{2}}\log\frac{1}{\sqrt{\alpha^*}}\ln\frac{1}{\delta}, \frac{n}{\sqrt{\alpha^*}}(\log\frac{1}{\sqrt{\alpha^*}})^2\right\}\right)$$

which is $O\left((\frac{1}{\alpha^*})^{\frac{n-1}{2}}\log\frac{1}{\sqrt{\alpha^*}}(\ln\frac{1}{\delta} + n\log\frac{1}{\sqrt{\alpha^*}})\right)$ using a constant m_0 and the $\max$ is upper bounded by plus. ∎

We can see that the SAC$_\text{I}$ algorithms can accelerate the uniform search by a factor near $\frac{1}{\sqrt{\alpha^*}}/\log\frac{1}{\sqrt{\alpha^*}}$. The closer the approximation, the more the acceleration.

On Spike Function Class
As modeling EAs, SAL algorithms should be expected to be applied on the complex problems, while the Sphere Function class only consists of convex functions. Inherited from EAs, SAL algorithms can handle problems with some local optima. We show this by comparing SAC$_\text{I}$ with the uniform search on the Spike Function class defined below.

Define regions $A_{1,k} = [\frac{3k}{20}, \frac{3k+2}{20}]$ where $0 \leq k \in \mathbb{N} \leq 6$ and $A_{2,k} = (\frac{3k-1}{20}, \frac{3k}{20})$ where $1 \leq k \in \mathbb{N} \leq 6$, 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 $X_n = [-\frac{1}{2}, \frac{1}{2}]^n$ be the n-dimensional solution space. The Spike Function class is $\mathcal{F}^n_{spike} = \{f^{x^*,n}_{spike} | \forall x^* \in X_n\}$, where, for all $x \in X_n$

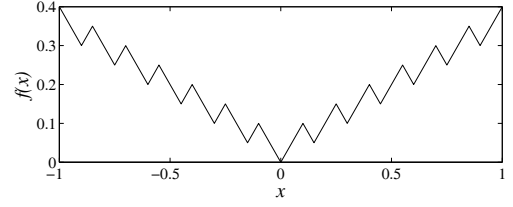$$f^{x^*,n}_{spike}(x) = g(\frac{1}{\sqrt{n}}\|x - x^*\|_2).$$

Fig. 1. The landscape of function $f^{0,1}_{spike}(x)$ in $[-1,1]$.

It is easy to know $\min_{x\in X_n} f(x) = 0$ and $\max_{x\in X_n} f(x) \leq 1$ for any $f \in \mathcal{F}^n_{spike}$. For any $\alpha > 0$, we can bound the area $|D_\alpha| \in [C_n\alpha^n, C_n(3\alpha)^n]$, where $C_n = \Theta\left((2\pi e)^{\frac{n}{2}}/\sqrt{\pi n}\right)$.

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

Minimizing any function in $\mathcal{F}^n_{spike}$ using the uniform search, the PAA query complexity with approximation level $\alpha^* > 0$ is, with probability at least $1 - \delta$,

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

We configure the SAC$_\text{I}$ algorithm to use the learning algorithm $\mathcal{L}_{spike}$ that searches the smallest sphere covering all the samples labeled as positive, of which the VC-dimension is $n+1$. Note that since the function is non-convex, the $\mathcal{L}_{spike}$ may output a sphere that also covers some negative examples, and thus with some training error. Using this SAC$_\text{I}$ algorithm to minimize any member in the function class $\mathcal{F}^n_{spike}$, we obtain the PAA query complexity as in Proposition 3.

Proposition 3 *For any function in $\mathcal{F}^n_{spike}$ and any approximation level $\alpha^* > 0$, SAC$_\text{I}$ algorithms can achieve the PAA query complexity*

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

with probability at least $1 - \delta$.

Proof. For any function in $\mathcal{F}^n_{spike}$, we note that the function is convex in D_α when α is smaller than 0.05. We set $\alpha_t = \frac{1}{2^t}$, so that when $t \geq 5$, the SAC$_\text{I}$ algorithm with $\mathcal{L}_{spike}$ 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 $|D_\alpha| \in [C_n\alpha^n, C_n(3\alpha)^n]$, we can obtain $T \geq \log\frac{3(C_n)^{\frac{1}{2n}}}{\sqrt{\alpha^*}}$. We let the SAC$_\text{I}$ algorithm run $T = \log\frac{1}{\sqrt{\alpha^*}}$ number of iterations and assume that $\log\frac{1}{\sqrt{\alpha^*}}$ is an integer.

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

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

$$\overline{\mathbf{Pr}}_h \geq \frac{1}{2\log\frac{1}{\sqrt{\alpha^*}}}\sum_{t=5}^{\log\frac{1}{\sqrt{\alpha^*}}}\left(\frac{|D_{\alpha^*}|(1-Q\epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}| + Q\epsilon_{\mathcal{D}_t}}\right)$$

$$\geq \frac{C_n(\alpha^*)^n}{2\log\frac{1}{\sqrt{\alpha^*}}} \sum_{t=5}^{\log\frac{1}{\sqrt{\alpha^*}}} \frac{1-2\frac{1}{2^t}}{C_n(\frac{3}{2^t})^n + 2\frac{1}{2^t}}$$

$$\geq \frac{C_n(\alpha^*)^n}{2\log\frac{1}{\sqrt{\alpha^*}}} \frac{\frac{15}{16}}{3C_n+2} \sum_{t=5}^{\log\frac{1}{\sqrt{\alpha^*}}} 2^t$$

$$= \frac{15}{32} \frac{C_n(\alpha^*)^n}{\log\frac{1}{\sqrt{\alpha^*}}} \frac{\frac{2}{\sqrt{\alpha^*}}-2^5}{3C_n+2} = \Omega\Big(\frac{(\alpha^*)^{n-\frac{1}{2}}}{\log\frac{1}{\sqrt{\alpha^*}}}\Big).$$

So we obtain the query complexity from Theorem 1

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

as the max is upper bounded by plus. ∎

We observe from the proof that the non-convexity can result in non-zero training error for the learning algorithms in SAC$_I$ algorithms, and thus the search process is interfered. But as long as the non-convexity is not quite severe, like the Spike Functions, SAC$_I$ algorithms are not significantly affected, and can still be better than the uniform search by a factor near $\frac{1}{\sqrt{\alpha^*}}/\log\frac{1}{\sqrt{\alpha^*}}$.

B. A Super-Polynomial Acceleration Condition

We have shown in Proposition 1 that SAC$_I$ algorithms using common classification algorithms cannot super-polynomially improve from the uniform search in the worst case. An interesting question is therefore raised that when the super-polynomial improvement is possible.

Learned from the proof of Proposition 1, a straightforward way is to use a powerful classification algorithm with exponentially improved sample complexity, i.e., $\widetilde{O}(\ln\frac{1}{\epsilon})$, so that only a polynomial number of samples is required to achieve a super-polynomially small error. Several *active learning* algorithms can do this in some circumstances (e.g. [7], [39]). Applying active learning algorithms needs a small modification of SAC$_I$. In iteration t, instead of sampling from the uniform distribution in D_{h_t}, the sampling is guided by the classifier. Nevertheless, the achieved error is still evaluated under the original (uniform) distribution. Using such learning algorithms denoted as $\mathcal{L}^{\ln}_{sphere}$, we achieve Proposition 4 showing a super-polynomial acceleration from the uniform search on Sphere Functions.

Proposition 4 *For any function in $\mathcal{F}^n_{sphere}$ and any approximation level $\alpha^* > 0$, SAC$_I$ algorithms using $\mathcal{L}^{\ln}_{sphere}$ can achieve the PAA query complexity, for any $n \geq 2$,*

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

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 α^*. Solving this equation with the sphere volume results in $T = \log\frac{(C_n)^{\frac{1}{n}}}{\alpha^*}$. We let the SAC$_I$ 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 $\mathcal{L}^{\ln}_{sphere}$, we want the error of the hypothesis h_t, $\epsilon_{\mathcal{D}_t}$, to be $\frac{1}{2^{tn/2}}$. Since the $\mathcal{L}^{\ln}_{sphere}$ has the sample complexity $O(\ln\frac{1}{\epsilon})$, we ask for a hypothesis with zero

training error, which requires the sample size $m_t = O(tn) = O(n\log\frac{1}{\alpha^*})$ with η being a constant.

We thus obtain $\sum_{t=1}^{T} m_t = O\left(n\big(\log\frac{1}{\alpha^*}\big)^2\right)$.

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

$$\overline{\mathbf{Pr}}_h \geq \frac{1}{2\log\frac{1}{\alpha^*}} \sum_{t=1}^{\log\frac{1}{\alpha^*}} \left(\frac{|D_{\alpha^*}|(1 - Q\epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}| + Q\epsilon_{\mathcal{D}_t}}\right)$$

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

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

$$\geq \frac{C_n(\alpha^*)^{\frac{n}{2}}}{2\log\frac{1}{\alpha^*}} \frac{(\frac{1}{\alpha^*})^{\frac{n}{2}}\big(1 - (2\alpha^*)^{\frac{n}{2}}\big)}{2(C_n+2)} = \Omega\Big(\frac{1}{\log\frac{1}{\alpha^*}}\Big).$$

So we obtain the query complexity from Theorem 1, letting m_0 be a constant,

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

which is $O\big(\log\frac{1}{\alpha^*}(\ln\frac{1}{\delta} + n\log\frac{1}{\alpha^*})\big)$. ∎

Meanwhile, we are more interested in exploring conditions under which the super-polynomial improvement is possible without requiring such powerful learning algorithms. For this purpose, we find the *one-side-error* condition.

Condition 2 (One-Side-Error) *In SAC algorithms, for any t and any $x \in X$, if $x \in D_{h_t}\Delta D_{\alpha_t}$, it must hold that $x \in D_{\alpha_t}$.*

The condition implies that h_t can only make false-negative errors, i.e., wrongly classifies positive samples (inside D_{α_t}) as negative, but no false-positive errors. One practical way to approach this condition is through the cost-sensitive classifiers [14], [46] with a very large mis-classification cost for negative samples. We call SAC$_I$ algorithms that are further under this condition as SAC$_{II}$ algorithms.

Lemma 5 *For SAC$_{II}$ algorithms, it holds for all t that*

$$|D_{h_t}| \leq |D_{\alpha_t}|.$$

Proof. Note that for training h_t we label the samples from D_{α_t} as positive and label the rest as negative. Since h_t only makes false-negative errors, i.e., every error is in D_{α_t}, we have $D_{h_t} \subseteq D_{\alpha_t}$, which implies the lemma. ∎

Lemma 5 shows that the one-side-error condition controls the size $|D_{h_t}|$ to be bounded by $|D_{\alpha_t}|$. Thus we can refine Lemma 3 as Lemma 6.

Lemma 6 *For SAC$_{II}$ algorithms, it holds for all t that*

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{|D_{\alpha^*}|(1 - \epsilon_{\mathcal{U}_X,t})}{|D_{\alpha_t}|},$$

where $\epsilon_{\mathcal{U}_X,t}$ is the expected error rate of h_t under $\mathcal{U}_X$.

Proof. Since the SAC$_{II}$ algorithm is also a SAC$_I$ algorithm, incorporating Lemma 5 into Lemma 3 proves the lemma. ∎

We assume that $\mathcal{L}^+_{sphere}$ is a learning algorithm that not only behaviors like $\mathcal{L}_{sphere}$ but also results a hypothesis

satisfying the one-side-error condition. Then a SAC$_\text{I}$ algorithm using $\mathcal{L}^+_{sphere}$ is a SAC$_\text{II}$ algorithm. We again assume that $\mathcal{L}^+_{sphere}$ is feasible, of which $VC(\mathcal{H}_n) = n+1$. We then use this SAC$_\text{II}$ algorithm on the Sphere Function class, on which SAC$_\text{I}$ algorithms bear a super-polynomial PAA complexity, and obtain Proposition 5.

Proposition 5 *For any function in $\mathcal{F}^n_{sphere}$ and any approximation level $\alpha^* > 0$, SAC$_\text{II}$ algorithms can achieve the PAA query complexity*

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

with probability at least $1 - \delta$.

Proof. By Lemma 6,

$$\frac{|D_{\alpha^*} \cap D_{h_t}|}{|D_{h_t}|} \geq \frac{|D_{\alpha^*}|(1 - Q\epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}|},$$

where $\epsilon_{\mathcal{D}_t}$ is the error of h_t under its original distribution $\mathcal{D}_t$, and Q is the resulting factor of changing the distribution.

Let $\alpha_t = \frac{1}{2^t}$ for all t, and use the number of iterations T to achieve $|D_{\alpha_T}| = |D_{\alpha^*}|$, for the approximation level α^*. Solving this equation with the sphere volume results in $T = \log \frac{(C_n)^{\frac{1}{n}}}{\alpha^*}$. We let the SAC$_\text{II}$ 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 $\mathcal{L}^+_{sphere}$, we want the error of the hypothesis h_t, $\epsilon_{\mathcal{D}_t}$, to be a constant $\frac{1}{2}$. Since $\mathcal{L}^+_{sphere}$ produces a hypothesis with zero training error, to achieve $\epsilon_{\mathcal{D}_t} \leq \frac{1}{2}$ it requires the number of samples in $O(n)$. We thus obtain $\sum_{t=1}^{T} m_t = O\left(n \log \frac{1}{\alpha^*}\right)$.

We then follow Eq.(1). We use uniform sampling within D_{h_t}, then $Q = \frac{1}{1-\lambda}$. Letting the SAC$_\text{II}$ algorithm use m_T number of samples in every iteration, $\lambda = \frac{1}{3}$ and $\eta = 0.5$, we have

$$\overline{\mathbf{Pr}}_h \geq \frac{1}{2 \log \frac{1}{\alpha^*}} \sum_{t=1}^{\log \frac{1}{\alpha^*}} \left(\frac{|D_{\alpha^*}|(1 - Q\epsilon_{\mathcal{D}_t})}{|D_{\alpha_t}|}\right)$$

$$\geq \frac{1}{2 \log \frac{1}{\alpha^*}} \sum_{t=1}^{\log \frac{1}{\alpha^*}} \left(\frac{\frac{1}{4}|D_{\alpha^*}|}{|D_{\alpha_t}|}\right)$$

$$= \frac{C_n(\alpha^*)^{\frac{n}{2}}}{8 \log \frac{1}{\alpha^*}} \sum_{t=1}^{\log \frac{1}{\alpha^*}} \frac{1}{C_n(\frac{1}{2^t})^{\frac{n}{2}}}$$

$$\geq \frac{C_n(\alpha^*)^{\frac{n}{2}}}{8 \log \frac{1}{\alpha^*}} \frac{\left((\frac{1}{\alpha^*})^{\frac{n}{2}} - 1\right)}{C_n} = \Omega\left(\frac{1}{\log \frac{1}{\alpha^*}}\right).$$

So we obtain from Theorem 1 the query complexity of the SAC$_\text{II}$ algorithm, letting m_0 be a constant,

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

which is $O\left(\log \frac{1}{\alpha^*}(\ln \frac{1}{\delta} + n)\right)$. ∎

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 random guess classification (i.e., error rate $\frac{1}{2}$) in the proof of Proposition 5.

IV. DISCUSSIONS AND CONCLUSIONS

This paper describes the sampling-and-learning (SAL) framework which is an abstract summary of a range of EAs. The SAL framework allows us to investigate the general performance of EAs from a statistical view. We show that the SAL framework directly admits a general upper bound on the PAA 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 conditions that drastically effect the performance of SAC algorithms. Under the *error-target independence* condition, which assumes that 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. Further incorporating the *one-side-error* condition, which assumes that the classification only makes false-negative errors, the SAC algorithms can obtain a super-polynomial improvement over the uniform search.

On the one hand, our results show that the property of classification error in SAC algorithms greatly impacts the performance, which was never touched in previous studies, as far as we know. We expect the work could guide the design of novel search algorithms. On the other hand, how to satisfy the conditions is a non-trivial practical issue.

In the case study on the Sphere Function class, we find that a learning error rate no more than the random guess is sufficient to achieve a super-polynomial improvement under the conditions. This implies that an accurate learning algorithm may not be necessary for a good SAC algorithm. It is interesting that a recent work [4] also noticed that a learnable concept is not necessary for the trial-and-error search with a computation oracle.

In this paper, the SAC algorithms are analyzed in continuous domains, while the main body of theoretical studies of evolutionary algorithms focuses on the discrete domains. Thus understanding the performance of SAC algorithms in discrete domains is our future work. 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.

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 (FOGA'09)*, 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 Theory of Computing (STOC'13)*, pages 31–40, Palo Alto, CA, 2013.

[5] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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 (FOGA'02)*, pages 253–270, Torremolinos, Spain, 2002.

[13] 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.

[14] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 973–978, Seattle, WA, 2001.

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

[21] 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.

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

[23] 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 (GECCO'12)*, pages 1325–1332, Philadelphia, PA, 2012.

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

[25] 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.

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

[27] S. Kratsch and F. Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. *Algorithmica*, 65(4):754–771, 2013.

[28] 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*, 2014.

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

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

[31] 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.

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

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

[34] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, New York, NY, 2004.

[35] 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.

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

[37] 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.

[38] 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 (AAAI'12)*, Toronto, Canada, 2012.

[39] 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.

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

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

[42] 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.

[43] 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.

[44] 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.

[45] D. Zhou, D. Luo, R. Lu, and Z. Han. The use of tail inequalities on the probable computational time of randomized search heuristics. *Theoretical Computer Science*, 436:106–117.

[46] Z.-H. Zhou and X.-Y. Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):63–77, 2006.

[47] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.