Chapter 9
Theory of Estimation-of-Distribution Algorithms

Martin S. Krejca and Carsten Witt

Abstract Estimation-of-distribution algorithms (EDAs) are general metaheuristics used in optimization that represent a more recent alternative to classical approaches such as evolutionary algorithms. In a nutshell, EDAs typically do not directly evolve populations of search points but build probabilistic models of promising solutions by repeatedly sampling and selecting points from the underlying search space. Recently, significant progress has been made in the theoretical understanding of EDAs. This chapter provides an up-to-date overview of the most commonly analyzed EDAs and the most recent theoretical results in this area. In particular, emphasis is put on the runtime analysis of simple univariate EDAs, including a description of typical benchmark functions and tools for the analysis. Along the way, open problems and directions for future research are described.

9.1 Introduction

Optimization is one of the most important fields in computer science, with many problems being NP-hard and thus not necessarily easy to solve. Hence, *heuristics* play a major role, i.e., optimization algorithms that try to yield solutions of good quality in a reasonable amount of time. Research over the past decades has resulted in many good heuristics being developed for classical NP-hard problems. Unfortunately, these heuristics are tailored with specific problems in mind and exploit certain problem-specific properties in order to

Martin S. Krejca
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
e-mail: martin.krejca@hpi.de

Carsten Witt
DTU Compute, Technical University of Denmark, Kgs. Lyngby, Denmark
e-mail: cawi@dtu.dk

B. Doerr, F. Neumann (eds.), *Theory of Evolutionary Computation*,
Natural Computing Series, https://doi.org/10.1007/978-3-030-29414-4_9

save computation time. Thus, they cannot be used for problems that do not feature these specific properties.

One alternative to problem-specific heuristics is *general-purpose* heuristics. The information about the problem to be optimized that these algorithms have access to is fairly limited, up to the point that they are only able to compare the quality of different solutions relatively. This has the advantage that the problem itself does not have to be formalized but only the quality of a solution, as the problem formalization is communicated implicitly via the quality measure to the algorithm. In turn, this results in great reusability of these algorithms for different problems.

One such class of general-purpose heuristics is *evolutionary algorithms* (EAs) [34]. EAs are characterized by creating new solutions from already generated solutions. Oftentimes, many solutions are stored and only changed (*evolved*) locally, preferably discarding bad solutions and saving good ones. Such algorithms are EAs in the classical sense [60].

The concept of EAs can be broadened if we are less restrictive about what is being evolved. A similar approach to changing solutions directly is to instead change the procedure that generates the solutions in the first place. Thus, a solution-generating mechanism is evolved. Algorithms following this approach are called *estimation-of-distribution algorithms* (EDAs) [29, 37, 54, 55]. They are not EAs in the classical sense but can be considered EAs in the broad sense, as just described.

EDAs have been used very successfully in real-world applications [29, 37, 54, 55] and have recently gathered momentum in the theory community analyzing EAs [10, 20, 22, 36, 38, 63, 66]. The aim of theoretically analyzing EAs is to provide guarantees for the algorithms and to gain insights into their behavior in order to optimize the algorithms themselves. Common guarantees include the expected time until an algorithm finds a solution of sufficient quality, the probability of doing so after a certain time, and the fact that the algorithm is even able to find desired solutions.

In this chapter, we provide a state-of-the-art overview of the theoretical results on EDAs for *discrete* domains, as that is their main field of application. To the best of our knowledge, while continuous EDAs exist, no detailed theoretical analyses have been conducted so far. We present the most commonly investigated EDAs and give an outline of the history of their analyses, providing deep insights into some of the latest results. After reading this, the reader should be familiar with EDAs in general, the current state of theoretical research, and common tools used for the analyses.

In Section 9.2, we go more into detail about how EDAs work, we introduce the scenario used in most theoretical papers, and we provide different ways of classifying EDAs, stating the most commonly analyzed algorithms. Further, we mention some tools that are often used when deriving results for EDAs. Then, in Section 9.3, we give a short overview of the most commonly considered objective functions. In Section 9.4, we discuss the historically older results of convergence analyses on EDAs. After that, in Section 9.5, we present

more recent results on EDAs, which consider the actual runtime of an algorithm. We end this article in Section 9.6 with some conclusions and open problems.

9.2 Estimation-of-Distribution Algorithms

In general, EDAs are problem-agnostic optimization algorithms that store a probabilistic model over the solution space. This model is the core part of these algorithms. It implies a probability distribution over the solution space and is iteratively refined, using samples. Ideally, the model converges to a state that produces only optimal solutions.

Since EDAs make use of sample sets – called *populations* – they are quite similar in this respect to EAs. However, the main difference is that EAs exclusively store a population and progress using solely this information, by varying samples – called *individuals* – from the population. Thus, they have quite a local view of the solution space and advance locally. In contrast, the probabilistic model of an EDA models most of the time the entire solution space. Updates to the model are done using the old model as well as a population. Hence, EDAs employ a more general view of the solution space than do classical EAs.

The probabilistic model of an EDA is used as an *implicit* probability distribution over the solution space, instead of an explicit distribution. This is usually done by constraining the distributions that can be modeled and by factorizing them, i.e., by writing the distribution as a product of marginal probabilities. Hauschild and Pelikan [29] distinguish between many different classes of EDAs with respect to how strongly constrained the models are. An advantage of factorizing a distribution is that it saves a lot of memory, since an explicit distribution would make it necessary to store a probability for each solution, which is not feasible. With a factorization, only the factors have to be stored in memory. However, even then it is possible for the model to grow to sizes exponential in the input [25].

As mentioned above, EDAs also use populations, like EAs, sampled from their probabilistic model, in order to update that model. It is up to the EDA to decide what to do with its population. However, all EDAs theoretically analyzed so far have in common the fact that they always discard their population after every iteration, valuing the model higher than the population.

In the following, we first state the optimization domain for the EDAs that we consider in this chapter. Then we discuss different classifications of EDAs and name various algorithms that fall into the various classes. Last, we mention the tools that are commonly used in the current theoretical research on EDAs.

9.2.1 Scenario

As in the theory of EAs, theoretical analyses of EDAs consider mainly pseudo-Boolean optimization, i.e., optimization of a function $f\colon \{0,1\}^n \to \mathbb{R}$, often referred to as the *fitness function*. Conventionally, the function value of a bit string $\boldsymbol{x}$ is called the *fitness of* $\boldsymbol{x}$.

The aspect of an EDA being a general-purpose solver is modeled as a classical black-box setting, where the algorithm gains problem-specific information only from querying the fitness function by inputting bit strings and receiving their respective fitness. In this setting, mostly two different scenarios have been of major interest.

Convergence analyses. In this historically older topic, EDAs have been analyzed with respect to the convergence of their probabilistic model, i.e., if they succeed at all in optimizing certain fitness functions. We discuss this scenario in more depth in Section 9.4.

Runtime analyses. A more recent trend is the analysis of an EDA's runtime on certain functions. In this scenario, the focus is on the number of queries needed until an optimum or a solution of sufficient quality is sampled, i.e., the first hitting time of an algorithm sampling such a solution. Although sampling a desired solution can happen by chance, the analyses usually entail that the probabilistic model of an EDA makes it very likely for such a solution to be sampled again. Section 9.5 goes into detail about this topic.

9.2.2 Classifications of EDAs

Arguably, the most straightforward way of classifying EDAs is with respect to the power of their underlying probabilistic model. *Univariate* algorithms use only a single variable in their model per problem variable.[1] In contrast, *multivariate* algorithms use more than a single variable to model a problem variable. Thus, univariate EDAs are not able to capture dependencies between problem variables, whereas multivariate EDAs are explicitly constructed to do so.

Pelikan et al [55] give a more fine-grained classification of EDAs, differentiating multivariate EDAs even further with respect to how many dependencies can be captured by the underlying probabilistic model.

Note that the classification into univariate and multivariate EDAs does not constrain the populations at all.

[1] In our setting of pseudo-Boolean optimization, a *problem variable* is a position in a bit string, i.e., one dimension of a hypercube.

9.2.2.1 Univariate Algorithms

When optimizing a pseudo-Boolean function, univariate EDAs assume independence of all of the n different bit positions to be optimized. Under this assumption, every probability distribution can be factorized into a product of n different probabilities $\boldsymbol{p}_i$, collected together in a vector $\boldsymbol{p}$ of length n. A bit string $\boldsymbol{x}$ is then sampled by choosing each bit $\boldsymbol{x}_i$ to be 1 with probability $\boldsymbol{p}_i$ and 0 otherwise. Since each $\boldsymbol{p}_i$ determines how frequently, in expectation, a 1 is sampled at position i, we call these probabilities *frequencies*, following the common naming convention [22]. The vector $\boldsymbol{p}$ is then consequently called the *frequency vector*.

n-Bernoulli-λ-EDA

Although the class of univariate EDAs does not limit the populations of the algorithms in any way, the most commonly considered univariate EDAs discard their entire population after every iteration. Thus, from a theoretical point of view, a run of a univariate EDA can be modeled as a series $(\boldsymbol{p}^{(t)})_{t\in\mathbb{N}_0}$ of frequency vectors over the number of iterations t. Usually, $\boldsymbol{p}^{(0)}$ models the uniform distribution by satisfying the condition that $\boldsymbol{p}_i^{(0)} = 1/2$ for each i. Friedrich et al [22] capture this class of univariate EDAs in a framework called the *n-Bernoulli-λ-EDA* (Algorithm 9.1).

The n-Bernoulli-λ-EDA samples λ individuals in each iteration and performs an update to its frequency vector, using the current frequency vector as well as all of the just-sampled individuals and their respective fitnesses. The function performing this update is called the *update scheme* and fully characterizes the algorithm.

Note that we do not specify a termination criterion. In fact, determining what a good criterion is may vary between different use cases of the algorithm. When considering the expected runtime of these algorithms (Section 9.5), we are interested in the number of fitness function evaluations until an optimal solution is sampled for the first time.

In many EDAs, if a frequency is either 0 or 1, all bits sampled at the respective position will be 0 or 1, respectively, and the update scheme will not change the frequency anymore. To prevent this, the algorithm is usually modified such that each frequency is only allowed to take values in an interval $[m, 1-m] \subset [0,1]$, where $m \in (0, 1/2]$ is called a *margin*; the values m and $1-m$ are called *borders*. Usually, a margin of $1/n$ is chosen [7, 10, 50]. In a scenario with a margin, line 8 of Algorithm 9.1 can be modified as follows:

foreach $i \in \{1,\dots,n\}$ **do**

$$\boldsymbol{p}_i^{(t+1)} \leftarrow \max\Big\{m,\ \min\big\{1-m,\ \varphi\big(\boldsymbol{p}^{(t)}, \big(\boldsymbol{x}, f(\boldsymbol{x})\big)_{\boldsymbol{x}\in D}\big)_i\big\}\Big\};$$

[2] Note that D is a multiset, that is, we allow duplicates.

Algorithm 9.1: n-Bernoulli-λ-EDA with a given update scheme φ, optimizing f

$t \leftarrow 0$;
foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t)} \leftarrow \frac{1}{2}$;
repeat
  $D \leftarrow \emptyset$;
  foreach $j \in \{1,\dots,\lambda\}$ **do**
    $\boldsymbol{x} \leftarrow$ offspring sampled with respect to $\boldsymbol{p}^{(t)}$;
    $D \leftarrow D \cup \{\boldsymbol{x}\}$; [2]
  $\boldsymbol{p}^{(t+1)} \leftarrow \varphi\big(\boldsymbol{p}^{(t)}, \big(\boldsymbol{x}, f(\boldsymbol{x})\big)_{\boldsymbol{x}\in D}\big)$;
  $t \leftarrow t+1$;
until termination criterion met;

We will continue to give an overview of the most commonly theoretically analyzed univariate EDAs and show how they fit into the n-Bernoulli-λ-EDA framework. We present the algorithms without a margin although they are commonly analyzed with a margin of $1/n$.

Since many of the following examples do not make use of the entire population of size λ (the *population size*) but select a certain number μ (the *effective population size*) of individuals according to their fitness values, we denote the k-th-best individual as $\boldsymbol{x}^{(k)}$, where $1 \leq k \leq \mu$; ties are broken *uniformly at random*. Thus, $\boldsymbol{x}^{(1)}$ denotes an individual with the best fitness.

UMDA

The arguably easiest update scheme is given by the *univariate marginal distribution algorithm* (UMDA; Algorithm 9.2) [49]. It samples λ individuals in each iteration, of which μ of the best are chosen. Then, each frequency $\boldsymbol{p}_i$ is set to the relative frequency of 1s at position i in the set of the μ best individuals, regardless of the current frequency.

The update scheme of UMDA allows it to go from any valid frequency to any other in a single step if not stuck. Thus, the difference of two consecutive frequencies $\boldsymbol{p}_i^{(t)}$ and $\boldsymbol{p}_i^{(t+1)}$ can only be trivially bounded by roughly 1. We call such a difference the *step size* of the algorithm.

PBIL

A variant of UMDA that has an adjustable step size is the *population-based incremental learning* algorithm (PBIL; Algorithm 9.3) [4]. A frequency is updated in a way similar to UMDA, but the new frequency is a convex combination with parameter ρ of the current frequency and the relative frequencies

Algorithm 9.2: UMDA with population size λ, effective population size μ, optimizing f

$t \leftarrow 0$;
foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t)} \leftarrow \frac{1}{2}$;
repeat
  $D \leftarrow \emptyset$;
  foreach $j \in \{1,\dots,\lambda\}$ **do**
    $\boldsymbol{x} \leftarrow$ offspring sampled with respect to $\boldsymbol{p}^{(t)}$;
    $D \leftarrow D \cup \{\boldsymbol{x}\}$;
  foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t+1)} \leftarrow \frac{1}{\mu}\sum_{k=1}^{\mu} \boldsymbol{x}_i^{(k)}$;
  $t \leftarrow t+1$;
until termination criterion met;

of 1s at that position. Thus, the step size is now bounded by ρ, and UMDA is a special case of PBIL with $\rho = 1$.

Algorithm 9.3: PBIL with population size λ, effective population size μ, and learning rate ρ, optimizing f

$t \leftarrow 0$;
foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t)} \leftarrow \frac{1}{2}$;
repeat
  $D \leftarrow \emptyset$;
  foreach $j \in \{1,\dots,\lambda\}$ **do**
    $\boldsymbol{x} \leftarrow$ offspring sampled with respect to $\boldsymbol{p}^{(t)}$;
    $D \leftarrow D \cup \{\boldsymbol{x}\}$;
  foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t+1)} \leftarrow (1-\rho)\boldsymbol{p}_i^{(t)} + \frac{\rho}{\mu}\sum_{k=1}^{\mu} \boldsymbol{x}_i^{(k)}$;
  $t \leftarrow t+1$;
until termination criterion met;

MMAS_{ib}

Another important univariate EDA is the *max-min ant system with iteration-best update* (MMAS_{ib}; Algorithm 9.4) [50], which is a special case of PBIL where we set $\mu = 1$, i.e., where we consider only the best individual in each iteration. MMAS_{ib} also falls into the general class of *ant colony optimization* (ACO) algorithms [16]. Although ACO spans an entire research topic independent of EDAs and is typically not considered to be an EDA, the process of how it produces solutions iteratively can be viewed as refining a probabilistic model. Thus, we view ACO as an EDA here.

ACO considers graphs whose edges are weighted with probabilities, called *pheromones*. Additionally, the algorithm uses agents – called *ants* – that traverse the graph and thus construct paths. At each vertex v, if a path needs to be extended, an ant chooses an edge with a certain probability with respect to the pheromones on all of the outgoing edges of v. After the data of all ants has been collected, all pheromones decrease (they *evaporate*) and then some are increased afterward, usually the ones that are part of the best solutions constructed.

When pseudo-Boolean optimization is considered, a graph for ACO can be modeled as a multigraph with $n+1$ vertices from 0 to n, each vertex having exactly two outgoing edges to its direct successor (except for vertex n; see Fig. 9.1). One of these edges is interpreted as a 0, and the other one as a 1. Each solution is constructed by letting an ant traverse the graph starting at 0 and ending at n. The corresponding edges are then interpreted as a bit string of length n. Note how the probability of choosing an edge corresponding to a 1 is equal to an n-Bernoulli-λ-EDA's frequency for that respective position.

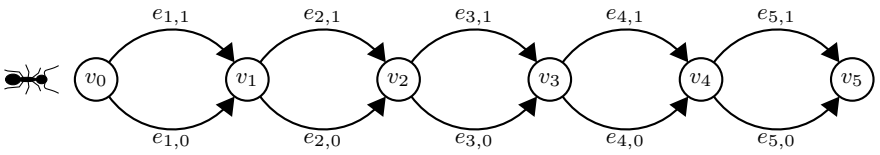

Fig. 9.1 The ACO graph for pseudo-Boolean optimization with $n = 5$ bits.

MMAS_{ib} is a variant of the max-min ant system algorithm [61] that only makes an update with respect to the path of the best ant in each iteration, using a classical update rule in ACO.

Algorithm 9.4: MMAS_{ib} with population size λ and evaporation factor ρ, optimizing f

$t \leftarrow 0$;
foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t)} \leftarrow \frac{1}{2}$;
repeat
  $D \leftarrow \emptyset$;
  foreach $j \in \{1,\dots,\lambda\}$ **do**
    $\boldsymbol{x} \leftarrow$ offspring sampled with respect to $\boldsymbol{p}^{(t)}$;
    $D \leftarrow D \cup \{\boldsymbol{x}\}$;
  foreach $i \in \{1,\dots,n\}$ **do** $\boldsymbol{p}_i^{(t+1)} \leftarrow (1-\rho)\boldsymbol{p}_i^{(t)} + \rho \boldsymbol{x}_i^{(1)}$;
  $t \leftarrow t+1$;
until termination criterion met;

cGA

An algorithm with a different approach is the *compact genetic algorithm* (cGA; Algorithm 9.5) [27]. It samples exactly two individuals in each iteration and compares their bit values componentwise. If the bits at position i are the same, the frequency $\boldsymbol{p}_i$ is left unchanged. Otherwise, the frequency is adjusted by $\pm 1/K$, where K is an algorithm-specific parameter, often referred to as the *population size*, such that the probability of sampling the bit value of the fitter individual is higher in the next iteration.

Algorithm 9.5: cGA with population size K, optimizing f

$t \leftarrow 0$;
foreach $i \in \{1, \dots, n\}$ **do** $\boldsymbol{p}_i^{(t)} \leftarrow \frac{1}{2}$;
repeat
  $D \leftarrow \emptyset$;
  foreach $j \in \{1, 2\}$ **do**
    $\boldsymbol{x} \leftarrow$ offspring sampled with respect to $\boldsymbol{p}^{(t)}$;
    $D \leftarrow D \cup \{\boldsymbol{x}\}$;
  foreach $i \in \{1, \dots, n\}$ **do** $\boldsymbol{p}_i^{(t+1)} \leftarrow \max\left\{0, \min\left\{1, \boldsymbol{p}_i^{(t)} + \frac{1}{K}\left(\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)}\right)\right\}\right\}$;
  $t \leftarrow t + 1$;
until termination criterion met;

9.2.2.2 Multivariate Algorithms

The class of multivariate EDAs consists of all algorithms that can use multiple variables to model one problem variable and thus express dependencies. A compact representation of such dependencies can be modeled as a directed graph whose vertices are the variables and whose edges denote dependencies among the variables. For each vertex, the probability distribution conditional on all its adjacent vertices with an incoming edge (its *parents*) is stored. This results in a factorization of the problem space that respects the given dependencies. Multivariate EDAs can assume a certain dependency model and learn only the respective (conditional) probabilities of the factorization, or they can additionally try to learn a model that fits well to the samples.

The *factorized distribution algorithm* (FDA) [48] falls into the former category. It assumes a factorization according to a so-called *additively decomposable function* (ADF), i.e., a function that is a sum of multivariate subfunctions. For each set of variables per subfunction, FDA creates a *metavariable*, and it expresses the objective function (the ADF) with respect to those metavariables. In each iteration, it samples solutions with respect to the factorization, selects a subset of them, and estimates the conditional probabilities based

on these samples. Note that FDA is a generalization of the update of UMDA and coincides with it if no dependencies between the problem variables exist.

Another approach that also uses metavariables is the *extended compact genetic algorithm* (ECGA) [28]. Differently from FDA, a metavariable of ECGA represents multiple variables at once (i.e., it is assumed that such variables are strongly correlated). In each iteration, the algorithm starts by placing each problem variable into its own class. Then, it greedily merges two classes such that a certain metric (the so-called *Bayesian information criterion*) is maximized, using samples from the current model. If no further improvement can be made, the merging process stops and the algorithm uses the newly created model.

The easiest of the multivariate cases is the one where each variable can be at most dependent on one other variable, i.e., a bivariate setting, and the arguably easiest probabilistic model in such a setting is a path. This model is used in the *mutual-information-maximization input clustering* (MIMIC) algorithm introduced by De Bonet et al [11]. The idea of the underlying model is to construct a path that minimizes the Kullback–Leibler divergence with respect to the bivariate setting, i.e., to find a permutation that can explain the sample data best. However, since there are $n!$ possible permutations for n variables, the authors of [11] suggest a greedy approach that makes use of the empirical entropies, i.e., the entropies of the sample data. First, a variable with minimum entropy is chosen as the start vertex of the path. Then, the path is continued by choosing a node that has minimum conditional entropy with respect to the currently last vertex in the path.

The *bivariate marginal distribution algorithm* (BMDA) [52] uses a somewhat similar approach. However, it does not consider paths as its model for dependency graphs but rather a forest of rooted trees. In order to determine which variables are dependent on which other variables, the Pearson's chi-squared statistic is used as an indicator. If the indicator is too low, the corresponding variables are considered independent. The forest is then created greedily very similarly to regular algorithms for maximum spanning trees: iteratively, a vertex is added to one of the trees that has maximum Pearson chi-squared value.

The *Bayesian optimization algorithm* (BOA) [53] is a very general multivariate EDA and constructs an arbitrary dependency graph with respect to a metric of choice. If wanted, the degree of incoming edges, i.e., the number of dependent variables, can be limited. Pelikan et al [53] proposed the Bayesian Dirichlet metric as one possibility to determine the quality of a dependency graph, and they stated that the general problem of finding an optimal graph is NP-hard. Thus, they suggested greedy algorithms or heuristics for efficiently creating good graphs.

9.2.2.3 Other Classifications

Another approach to classifying EDAs is to differentiate them not by how many dependencies they can model but by certain invariances that their probabilistic models may have.

One such classification stems from the theory of EAs and was introduced by Lehre and Witt [39]. These authors considered a new black-box complexity known as *unbiased black-box complexity* in order to prove tighter lower bounds for commonly analyzed EAs. This definition is so general that it applies to any black-box algorithm optimizing pseudo-Boolean functions, thus including EDAs.

Unbiased black-box complexity considers black-box algorithms optimizing perturbations of the hypercube, where a perturbation is any isometric automorphism of the hypercube.[3] For example, cyclically shifting a bit string by one position to the right and changing the value of the first bit in the result is an isometric automorphism.

Given a fitness function and a perturbed variant of it, a black-box algorithm is said to be *unbiased* if the queries to the black box in the perturbed setting are the same as the queries in the unperturbed setting when inverted with respect to the perturbation. Thus, an unbiased algorithm does not favor certain positions over other positions or 1s over 0s, or vice versa, i.e., it has no bias in this respect.

When considering general-purpose algorithms, unbiasedness is a nice property to have, as it certifies that the algorithm has no bias with respect to the encoding of the search space. However, when considering certain problems, different values may have a strict, different meaning, such that unbiasedness with respect to those values does not make sense.

All of the EDAs presented in Section 9.2.2.1 are unbiased when uniform tie-breaking is used.

A seemingly similar but unrelated property that many EDAs feature is that their probabilistic model does not change, in expectation, if all samples have the same fitness, i.e., there is no signal from the fitness function. Friedrich et al [22] called this property *balanced*, with respect to the n-Bernoulli-λ-EDA. However, this property had already been considered before by Shapiro [58], albeit with different terminology.

Although balancedness seems beneficial at first glance, it actually leads to the probabilistic model converging to one of the corners of the hypercube [22, 58]. This is a general problem of martingales, i.e., random processes that do not change in expectation, with a bounded range, which will eventually end up at the bounds of their range. This means that balancedness implies a bias toward outer regions of the hypercube, also called *genetic drift* [3], as this is an inherent drift due to the genotypes of the sampled population. In order

[3] The isometric automorphisms of the hypercube are all isomorphisms that permute any positions and may change a value of x to $1 - x$ at any position.

to overcome this bias and optimize successfully, the drift due to selection introduced by the fitness function has to be larger than the genetic drift.

Different approaches have been suggested in order to prevent an EDA's probabilistic model from quickly converging to a corner of the hypercube. Shapiro [58] proposed to reject updates made to the probabilistic model with a probability equal to the ratio of going from one model to the other. This has the advantage that the resulting implicit distribution is the uniform distribution over the hypercube. However, the transition probabilities have to be known and computed in order to get the correct rejection probabilities. Another approach proposed by Shapiro [58] and also by Friedrich et al [22] is to introduce an artificial bias that counteracts the one introduced by the balancedness.

In the context of balancedness, Friedrich et al [22] introduced another concept, which they called *stable*. An n-Bernoulli-λ-EDA is stable if the limit distribution of each frequency, when no fitness signal is received, is unimodal with its maximum at $1/2$. This means that a stable n-Bernoulli-λ-EDA has a bias toward the center of the hypercube. These authors showed that this concept is mutually exclusive with an n-Bernoulli-λ-EDA being balanced, as such an EDA has a bias toward the corners of the hypercube. The *stable* property is similar to the concept of an EDA's limit distribution being the uniform distribution, as considered by Shapiro [58].

9.2.3 Tools for Analyzing EDAs Theoretically

Most of the theoretical results on EDAs consider univariate algorithms, as we explain in Section 9.5. Thus, tools that make use of independent events are commonly used. However, that does not limit the use of these tools to the univariate case. Especially, *drift analysis*, which we present later in this section, can be applied in any setting.

Many proofs make use of classical probabilistic concentration bounds, such as Markov's inequality, Chebyshev's inequality, or, most importantly, Chernoff bounds [44]. The latter are used very frequently, since the sampling process of a univariate EDA is usually done independently of the other samples. Thus, such a bound can be applied.

Since the theory of EDAs usually considers first hitting times, more specialized tools suited for that purpose are used as well. One such tool is the coupon collector problem [45], which gives highly concentrated first-hitting-time results if a certain number of events with low probability have to occur to reach the target. For EDAs, this can be thought of as a certain number of factors of the probabilistic model being at the wrong end of their spectrum, thus slowing down optimization, since they need to be changed for the optimization process to succeed.

Another tool for determining first hitting times, and the most prominent one when looking at the theory of EAs and EDAs in general, is *drift theory*. It is loosely akin to the potential method in complexity theory. To apply drift theory, one needs to define a potential that maps the stochastic process into the reals. Then, the expected difference between two consecutive steps of the process is considered: the *drift*. This can be thought of as the expected velocity of the process. If the drift can be bounded, the expected hitting time of the process reaching a target is easily deducible, i.e., if there is a known bias in the process toward a certain direction, the first hitting time can easily be bounded.

We now state the three most commonly used drift theorems. The most general theorem with respect to the prerequisites of the process – the additive drift theorem (Theorem 9.2.1) – was stated by He and Yao [30]. However, the ideas used date back to Wald's equation [65].

Theorem 9.2.1 (additive drift [30, 31]). *Let $(X_t)_{t\in\mathbb{N}_0}$ be random variables over a bounded space $S \subseteq \mathbb{R}_{\geq 0}$ containing 0, and let $T = \min\{t \mid X_t = 0\}$.*

If there is a constant $\delta > 0$ such that, for all $t < T$, $E[X_t - X_{t+1} \mid X_t] \geq \delta$, then

$$E[T \mid X_0] \leq \frac{X_0}{\delta}.$$

And if there is a $\delta > 0$ such that, for all $t < T$, $E[X_t - X_{t+1} \mid X_t] \leq \delta$, then

$$E[T \mid X_0] \geq \frac{X_0}{\delta}.$$

The additive drift theorem can be applied when the expected difference between two potentials is known. However, oftentimes it is easier to determine the expected difference conditional on the current potential, i.e., $E[X_t - X_{t+1} \mid X_t]$. Owing to the law of total expectation, a lower bound on the conditional expected value is also a lower bound on the unconditional one.

A theorem more suited to processes whose potential changes at least linearly with respect to the current potential is the following multiplicative drift theorem.

Theorem 9.2.2 (multiplicative drift [14]). *Let $(X_t)_{T\in\mathbb{N}_0}$ be nonnegative random variables over $\mathbb{R}$, each with finite expectation, and let $T = \min\{t \mid X_t < 1\}$.*

If there is a constant $\delta > 0$ such that, for all $t < T$, $E[X_t - X_{t+1} \mid X_t] \geq \delta X_t$, then

$$E[T \mid X_0] \leq \frac{1 + \ln X_0}{\delta}.$$

The multiplicative drift theorem is not well suited if the difference in potential is dependent on the current potential but not in a linear fashion. Such cases are covered by the following variable drift theorem. However, note that

all these theorems assume that the difference in potential does not increase when one gets closer to the goal.

Theorem 9.2.3 (variable drift [35, 43]). *Let $(X_t)_{t\in\mathbb{N}_0}$ be nonnegative random variables over a bounded space $S \subseteq \mathbb{R}_{\geq 0}$ containing 1, each with finite expectation, and let $T = \min\{t \mid X_t < 1\}$.*

If there exists a monotonically increasing function $h\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that $1/h$ is integrable and, for all $t < T$, $E[X_t - X_{t+1} \mid X_t] \geq h(X_t)$, then

$$E[T \mid X_0] \leq \frac{1}{h(1)} + \int_1^{X_0} \frac{1}{h(x)}\,\mathrm{d}x.$$

The drift theorems above have been formulated in a simple, easy-to-read form that covers the most typical scenarios in which they are applied. However, more general drift theorems can be obtained [40, 41]; for example, to apply Theorem 9.2.1 in unbounded state spaces, to apply Theorems 9.2.2 and 9.2.3 with respect to arbitrary minimum states $s_{\min} > 0$ in the definition of T instead of state 1, and to allow processes adapted to arbitrary stochastic filtrations instead of the natural one implicit in the formulations above. These generalizations come partly at the cost of more complicated theorem statements, and sometimes require some additional technical assumptions about the underlying stochastic process.

9.3 Common Fitness Functions

The most commonly analyzed pseudo-Boolean functions for EDAs are OneMax [46] and LeadingOnes [56]. However, other functions have also been analyzed [6, 7], with BinVal being the most prominent one of them [17, 48].

OneMax counts the number of 1s in a bit string. Thus, the unique optimum is the all-1s bit string:

$$\textsc{OneMax}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} \boldsymbol{x}_i. \tag{9.1}$$

This function can be generalized to a class of functions, each having a target bit string $\boldsymbol{a}$ – which denotes the unique global optimum – and yielding the number of incorrectly set bits. Note that any unbiased algorithm, as introduced in Section 9.2.2.3, behaves on OneMax exactly as on the generalized version.

The OneMax function class is used to analyze how well an EDA performs as a hill climber. The usual expected runtime of an EDA on this function is $\Theta(n \log n)$ [36, 38, 63, 66].

Whereas OneMax is oftentimes considered to be the easiest pseudo-Boolean function, BinVal is said to be the hardest [17]. In contrast to OneMax, where all bits are equally weighted, BinVal uses exponentially scaled weights on its bit positions:

$$\textsc{BinVal}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} 2^{n-i} \boldsymbol{x}_i. \tag{9.2}$$

That means that BinVal represents value of a bit string interpreted as a binary unsigned integer.

Since the sum of all powers of 2 up to an exponent j is less than 2^j, BinVal can be interpreted as a lexicographic order on the hypercube, where lexicographically greater bit strings have a better fitness.

As with OneMax, in its general form, the global optimum of BinVal is any bit string $\boldsymbol{a}$, and the fitness of any bit string is the weight of the respective index if the bit value is the same as that of $\boldsymbol{a}$, and it is 0 otherwise.

LeadingOnes yields the number of consecutive 1s in a bit string, starting from the left:

$$\textsc{LeadingOnes}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} \prod_{j=1}^{i} \boldsymbol{x}_j. \tag{9.3}$$

As with OneMax, the unique global optimum is the all-1s bit string. In its general version, the function yields the number of consecutively correctly chosen bits with respect to a fixed permutation π and a target bit string $\boldsymbol{a}$.

LeadingOnes is used to analyze how an EDA copes with dependencies between the bits. The known expected runtime of certain EDAs on this function is $O(n^2)$ [10], which is compliant with the usual upper bound for EAs on this function [1].

9.4 Convergence Analyses

The earliest theoretical studies of EDAs focused mostly on their convergence, and were similar in style to the research that had been done for evolutionary algorithms in the 1990s [56, 64]. More precisely, it was studied how an algorithm behaves in the limit $t \to \infty$, i.e., if the algorithm is allowed to run for an arbitrary amount of time. If optimal solutions will be found in this limit, the algorithm is considered effective.

Almost all convergence analyses of EDAs consider univariate models. An early publication by Höhfeld and Rudolph [32] studied the vector of frequencies $\boldsymbol{p}^{(t)}$ in PBIL using a Markov chain model and rigorously proved that if $\mu = 1 < \lambda$ and $\rho > 0$, it will converge in expectation to some solution $\boldsymbol{x}^* = (\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^*)$; more precisely, $E[\boldsymbol{p}_i^{(t)}] \to \boldsymbol{x}_i^*$ as $t \to \infty$. This solution need not be an optimal one but may correspond to a local optimum to which

the search process is led in the very first steps. If the fitness function f is a linear pseudo-Boolean function, then in fact $E[\boldsymbol{p}_i^{(t)}] \to \boldsymbol{x}_i^*$ with respect to the optimal solution $\boldsymbol{x}^*$. This includes classical benchmark functions such as OneMax. However, as pointed out by Shapiro [58], convergence in expectation does not imply that PBIL eventually will sample the optimum of such functions. In fact, genetic drift may lock frequencies to values that make it impossible to sample the optimum.

PBIL was also theoretically analyzed by González et al [26] using a dynamical systems model. Convergence of the model to local optima of the fitness functions was proven for $\mu = 1$, and it was argued that the actual PBIL will resemble the model if ρ is chosen sufficiently close to 0. Hence, the approach does not make predictions for high learning rates ρ, in particular, it excludes the special case of $\rho = 1$ as used in UMDA.

Several subsequent publications have considered UMDA and its generalization FDA. Mühlenbein and Mahnig [47] also used an approach similar to dynamical systems theory to derive a quantitative statement about the behavior of the frequencies in FDA and UMDA over time. In fact, both fitness-proportionate and the usual truncation selection (take the best μ out of λ individuals) were considered. Specifically, for the classical UMDA on OneMax, they derived the result that, roughly,

$$\boldsymbol{p}_i^{(t+1)} \approx \boldsymbol{p}_i^{(t)} + \frac{I}{\sqrt{n}} \sqrt{\boldsymbol{p}_i^{(t)} \left(1 - \boldsymbol{p}_i^{(t)}\right)}\,, \tag{9.1}$$

where I is the so-called selection intensity, which is determined from the ratio μ/λ and can be thought of as being constant. By solving a differential equation, the formula can be turned into an approximation to the expected frequency at time t. Interestingly, (9.1) resembles a rigorous statement about the drift of the frequencies that was recently proven in [66] and is crucial for upper bounds on the runtime; see a more detailed discussion in Section 9.5.2.1.

A more comprehensive convergence study of FDA was done by Mühlenbein and Mahnig [48]. As a general assumption, the FDA is instantiated with the correct decomposition of an additively decomposable function $f(x) = \sum_{i=1}^{k} f(X_j)$, where $X_j \subset \{1, \dots, n\}$, into its subfunctions. Then the algorithm will compute a probabilistic model, comprising unconditional and conditional frequencies from the sampled search points. Strong results are obtained if a fitness-proportionate selection scheme called Boltzmann selection is used. Under some assumptions about the initial population, the algorithm will converge to a distribution that is uniform on the set of optimal solutions. The drawback of this result is that Boltzmann selection is computationally very expensive. For the usual truncation selection, results building on simplifying assumptions were obtained. Moreover, using infinite-population models, the paper derived quantitative statements similar to (9.1) about the time for a frequency of UMDA to converge to its optimum value, regarding OneMax and BinVal.

In the 2000s, rigorous convergence proofs of FDA (including UMDA) with fitness-proportionate [72] and truncation selection [71] followed. To study the regions of convergence and their stability, this research was supplemented by a fixed-point analysis for UMDA and FDA with 2-tournament selection in [70]. It turns out that FDA, given an appropriate decomposition of a non-linear function, converges under milder assumptions about the starting population than UMDA. Roughly, this indicates that a multivariate model, as used in FDA, can be superior to a univariate model, to which UMDA is restricted. However, the analyses in [70–72] also make the assumption of an infinite population size, which was very common in early convergence analyses of nature-inspired algorithms [64]. Infinite populations simplify the analysis, since certain stochastic effects leading to a deviation from the expected behavior, so-called fluctuations such as genetic drift, vanish under this assumption. Often this type of analysis has been accompanied by experiments, which support the validity of the statements for finite population sizes also. Theoretically motivated research often demands rigorous statements that also hold for finite populations, see the following sections on runtime analysis.

A more recent publication by Wu and Kolonko [68] presented a convergence analysis of a so-called generalized cross-entropy optimization algorithm. The algorithm generalizes PBIL by adding so-called feasibility information to elements of the search space. This information corresponds to the *heuristic information* used in ACO [15]. It was shown, for constant ρ and under different assumptions about the feasibility information, that the algorithm may stagnate in suboptimal points owing to genetic drift. However, for a time-dependent update scheme, almost sure convergence to a set of solutions that may include optimal points was proven. Finally, an initial runtime analysis on LeadingOnes was presented. However, this specific result has been superseded by more detailed analyses in a follow-up paper [69], discussed below.

To conclude this overview of convergence analyses, we mention a very recent publication by Ollivier et al [51]. They introduced the *information-geometric optimization* (IGO) algorithm, which is a very general EDA framework derived from three invariance properties: invariance under the parameterization of the search space, invariance under the parameterization of the probabilistic model, and invariance under monotone transformations of the fitness function. This means that IGO does not care about the encoding of the search space, the probabilistic model, or absolute fitness values. These authors showed that IGO results in a general EDA that encompasses PBIL and cGA when it is used on the discrete hypercube, considering Poisson binomial distributions. Further, they considered a time-continuous infinite-population version of IGO, which they called IGO flow, in the setting of linear pseudo-Boolean optimization and proved that it always converges to the optimum if the probabilistic model is not ill-initialized, i.e., none of the probabilities are initialized such that sampling the optimum is impossible.

9.5 Runtime Analyses

In contrast to convergence results as described in Section 9.4, the focus of runtime analyses is the number of iterations until an algorithm samples a solution of sufficient quality for the first time, usually an optimum. Normally, the analyses consider both the expected number of iterations and concentration results.

In this section, we first give an in-depth overview of the history of runtime analyses on EDAs, ending with a very detailed discussion of the most recent results. These results are summarized in Table 9.1. Then, we consider noisy scenarios, i.e., scenarios where the fitness function is perturbed by some kind of noise, usually as an additive term to the original fitness. In this setting, every time a solution is evaluated, the noise is drawn again and independently of any prior noise, and the goal is to optimize the underlying unperturbed function despite the noise.

9.5.1 Early Results

We start with a discussion of the first publications addressing runtime aspects of EDAs, which date back to the early 2000s. Although some of the runtime bounds proven in these publications can now be improved with state-of-the-art methods, the analyses already point out typical scenarios and challenges in the runtime behavior of EDAs, in particular regarding genetic drift. Also they give insights into fundamental properties of EDAs that distinguish them from other nature-inspired algorithms such as EAs.

9.5.1.1 First Steps Towards Runtime Analyses

As pointed out above, rigorous runtime analyes must avoid the infinite-population model and derive statements for populations of finite size. However, the finiteness comes at a cost: if very small populations are used, there is a high risk of genetic drift and premature convergence in suboptimal regions of the search space. In a series of publications, Shapiro [57–59] addressed sources of genetic drift in EDAs, quantified its impact, and proposed measures to avoid it. In [58], he pointed out that the probability distribution evolved by an EDA may converge to suboptimal points and, using a dynamical systems approach, determined $\sqrt{n}$ as the minimum population size for UMDA to avoid genetic drift on the OneMax problem, and even exponential sizes for Needle. Later, Sudholt and Witt [63] and Krejca and Witt [36] gave rigorous proofs of the fact that genetic drift can happen up to population sizes of $O(\sqrt{n}\log n)$ in cGA and UMDA. Alternatively, for PBIL, the learning rate ρ may be reduced to counteract genetic drift. Using a dynami-

cal systems approach, Shapiro [57] derived the result that the learning rate should be $O(1/\sqrt{n})$ and $O(2^{-n})$ to avoid genetic drift on the OneMax and Needle functions, respectively.

In [59], Shapiro also gave a rigorous theorem on the speed at which genetic drift moves the probabilistic model belonging to a specific class of EDAs called SML-EDA (including UMDA) into suboptimal regions. Also, a rigorous bound $\Omega(2^{n/2}/\sqrt{n})$ was determined for the population size required to make genetic drift on Needle unlikely.

Finally, in [58, p. 115], early conjectures about the runtime of UMDA appeared. More precisely, the paper reported an experimental determination of a runtime of $\Theta(\lambda\sqrt{n})$ for UMDA on OneMax (given that λ is asymptotically larger than $\sqrt{n}$). This bound was rigorously proven in [66]. However, it should be noted that Shapiro's UMDA slightly differs from the standard.

9.5.1.2 First Runtime Analyses

The first rigorous runtime analysis of an EDA was given by Droste [17]. He considered cGA without borders and proved the general lower bound $\Omega(K\sqrt{n})$ for its expected runtime on all linear functions. Using classical drift analysis and Chernoff bounds, Droste also proved the bound $O(K\sqrt{n})$ for OneMax, using $K = \Omega(n^{1/2+\varepsilon})$, i.e., slightly above the threshold stated by Shapiro [58]. This bound becomes $O(n^{1+\varepsilon})$ for the smallest K covered by his analysis. Finally, Droste argued that BinVal is more difficult to optimize than OneMax and asymptotically most difficult within the class of linear functions by proving that cGA without borders takes time $O(Kn)$ with at least constant probability on this function if $K = \Omega(n^{1+\varepsilon})$, and expected time at least $\Omega(Kn)$. The upper bound is $O(n^{2+\varepsilon})$ for the smallest possible K allowed. However, the lower bound $\Omega(Kn)$ does not come with a minimum value for K.

The results for OneMax were recently refined by Sudholt and Witt [63], using more advanced tools. In particular, all of Droste's upper bounds apply Chernoff bounds to show that genetic drift is unlikely; more precisely, he showed that the probability of a frequency dropping below 1/3 during the optimization is superpolynomially small. Using a negative drift theorem, the upper bound was improved from $O(n^{1+\varepsilon})$ to $O(n \log n)$ in [63]. See Section 9.5.2.1 for more details. Regarding BinVal, a very recent analysis by Witt [67] proved Droste's conjecture that the function is harder to optimize than OneMax, since the expected optimization time of cGA on BinVal is $\Omega(n^2)$ no matter how K is chosen. The idea of the analysis is to show, for all $K = o(n)$, that genetic drift will lock many frequencies to 0 before the optimum can be found.

The results discussed in the previous two paragraphs are summarized in the following theorem.

Theorem 9.5.1 ([17, 67]). *Choosing $K = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$, the runtime of cGA without borders on* BinVal *is bounded by $O(Kn)$ with probability $\Omega(1)$. Moreover, the expected runtime of cGA with and without borders on* BinVal *is bounded from below by $\Omega(\min\{n^2, Kn\})$.*

In the years following Droste's seminal work, runtime analysis focused more on UMDA and variants thereof. The first runtime analysis of a UMDA variant was given by Chen et al [5], who studied the LeadingOnes function and a modification called TrapLeadingOnes. An expected optimization time of $O(\lambda n)$ of UMDA on LeadingOnes was derived under the so-called *no-random-error* assumption, which is similar to an infinite-population model and basically eliminates genetic drift. These authors also showed that TrapLeadingOnes, which starts out in the same way as LeadingOnes but requires an almost complete change of the probabilistic model for the EDA to reach the global optimum, yields expected exponential optimization time for UMDA, using 2-tournament selection instead of the usual truncation selection. Moreover, a generalization of UMDA similar to PBIL was considered, but it turned out that the strongest bounds apply for UMDA.

Strictly speaking, Chen et al [5] only derived runtime bounds for a model of UMDA. In subsequent work [7], they therefore supplemented these with a rigorous proof of the fact that UMDA, when appropriate borders for the frequencies are used, with high probability requires superpolynomial time to optimize TrapLeadingOnes. Similarly to Droste's early work, Chernoff bounds were applied to show that the frequencies do not deviate much from their expected behavior, i.e., do not exhibit strong genetic drift. For the Chernoff bounds to be sufficiently strong, unusually large population sizes such as $\lambda = \Omega(n^{2+\varepsilon})$ are required.

This approach was successfully picked up and extended in a more comprehensive journal publication [8]. Using $\lambda = \Omega(n^{2+\varepsilon})$ again, the authors of [8] showed that UMDA without borders optimizes LeadingOnes in time $O(\lambda n)$ with overwhelming probability. Furthermore, the utility of appropriately set frequency borders was shown on a modification called BVLO, where the fitness landscape requires the frequency of the last bit to be changed from one extremal value to the other one. Here, UMDA with borders has expected polynomial runtime, whereas UMDA without borders will with overwhelming probability be stuck at nonoptimal solutions.

Finally, using similar proof techniques, in particular Chernoff bounds, Chen et al [6] presented a constructed example function called Substring, on which simple EDAs and simple evolutionary algorithms behave fundamentally differently. More precisely, it was proven that the $(1+1)$ EA with any mutation probability c/n, where $c > 0$ is constant, with overwhelming probability needs exponential time to find the optimum of the function, while UMDA using $\lambda = \Omega(n^{2+\varepsilon})$ and $\lambda/\mu = O(1)$ finds with very high probability the optimum in time $O(\lambda n)$. Specifically, it is beneficial for the optimization that UMDA can sample search points with high variances as long as all frequencies are close to $1/2$. The $(1+1)$ EA always samples with low variance in

the vicinity of the best-so-far solution, which is detrimental with the specific example function.

9.5.2 Recent Advances

Only very few runtime analyses of EDAs were published in the years 2010–2014, most notably [50, 68]. Starting from 2015, this research area gained significant momentum again (see, e.g., [10, 20, 21]). We now discuss the latest results in runtime analysis of EDAs. They mostly consider the standard benchmark function for EAs: OneMax. Using and advancing the toolbox for the analysis, matching upper and lower bounds have been proven, giving a tight runtime result that allows a direct comparison of the performance of EDAs with other nature-inspired algorithms.

9.5.2.1 Upper Bounds for OneMax

Interestingly, early runtime analyses of EDAs focused more on variants of LeadingOnes instead of OneMax, which is the most commonly considered example function in evolutionary computation. In fact, the first runtime analysis of UMDA on OneMax was not published until 2015 [10]. A possible explanation is that the hierarchical structure of LeadingOnes makes it more accessible to a runtime analysis than OneMax: if the best-so-far LeadingOnes value is k and the frequencies of the first k bits all have attained their maximum value, it is likely to sample only 1s there, which is typically needed for an improvement of the best function value seen. In contrast, there is no direct relationship between the OneMax value and frequencies at specific bits. Also, modern runtime analyses of UMDA [10, 38] reveal that a proof of runtime bounds for LeadingOnes can be relatively short and simple once the case of OneMax has been understood.

Results for cGA and MMAS_{ib}

Before we describe the advances made in the runtime analysis of UMDA in more detail, we discuss the state of the art for the simpler EDAs MMAS_{ib} and cGA. As mentioned above, Droste [17] showed that cGA typically optimizes OneMax in time $O(n^{1+\varepsilon})$, using $K = n^{1/2+\varepsilon}$. His variant of cGA does not use any borders on the frequencies, which is why he used a comparatively large K to make convergence of a frequency to 0 by genetic drift sufficiently unlikely. More recent analyses of cGA and also other EDAs such as UMDA mostly impose borders $\{1/n, 1-1/n\}$ on the frequencies, as mentioned in Section 9.2.2.1. Using a more careful analysis of the stochastic behavior of

frequencies, the classical $O(n \log n)$ runtime can be obtained, as shown in the following summary of theorems.

Theorem 9.5.2 ([50, 63]). *If $\rho \leq 1/(cn^{1/2} \log n)$ for a sufficiently large constant $c > 0$ and $\rho \geq 1/\text{poly}(n)$, then MMAS$_{ib}$ (with borders) optimizes* OneMax *in expected time $O(\sqrt{n}/\rho)$. For $\rho = 1/(cn^{1/2} \log n)$, the runtime bound is $O(n \log n)$.*

The expected optimization time of cGA (with borders) on OneMax *with $K \geq c\sqrt{n} \log n$ for a sufficiently large $c > 0$ and $K = \text{poly}(n)$ is $O(\sqrt{n}K)$. This is $O(n \log n)$ for $K = c\sqrt{n} \log n$.*

Theorem 9.5.2 makes statements for two slightly different EDAs but the proofs of these statements follow roughly the same structure. Crucially, the effect of genetic drift is bounded: in the given time bound, for example, $O(\sqrt{n}K)$ generations, the expected number of frequencies that drop below $1/3$ is proven to be polynomially small, for example, $O(1/n^2)$. Such a statement is typically obtained from a negative drift theorem. Next, the drift of frequencies towards 1 induced by selection (the so-called bias) is analyzed. It turns out that this bias is at least proportional to the sampling variance of the EDA: roughly, each frequency $\boldsymbol{p}_i$ increases by an expected amount $O\big(\boldsymbol{p}_i(1-\boldsymbol{p}_i)/(K\sqrt{\sum_{j=1}^{n} \boldsymbol{p}_j})\big)$ in each generation. An analysis of this variable drift, using the variable drift theorem, then gives the desired runtime bound. (As variable drift analysis was not available to Neumann et al [50], a unified and simpler proof of the statement for MMAS$_{\text{ib}}$ was given in [63].) In the unlikely event that a frequency has reached the wrong border $1/n$ owing to genetic drift, an event of probability $\Omega(1/n)$ is sufficient to lift the frequency again, which is absorbed into the total runtime owing to the low expected number of such bad frequencies.

First Phase Transition Around $\sqrt{n} \log n$

Theorem 9.5.2 requires $K \geq c\sqrt{n} \log n$. Recent research reveals that cGA in fact exhibits a phase transition in the regime $\Theta(\sqrt{n} \log n)$, similarly to MMAS$_{\text{ib}}$. If $K \leq c'\sqrt{n} \log n$ for a sufficiently small constant $c' > 0$, then genetic drift will outweigh the drift due to selection such that a significant number of frequencies will drop to the lower border. In this case, classical arguments about coupon collector processes show that the runtime must be at least $\Omega(n \log n)$; see more arguments below, in Section 9.5.2.2, on lower bounds. There are no upper bounds on the runtime of cGA and MMAS$_{\text{ib}}$ in the regime corresponding to $K \leq c'\sqrt{n} \log n$, but it is conjectured that bounds resembling the existing ones for UMDA (see Theorems 9.5.3 and 9.5.4 below) can be obtained if $K \in [c_1 \log n, c_2\sqrt{n} \log n]$ for appropriate constants $c_1, c_2 > 0$.

Results for UMDA

We complete this discussion of upper bounds with a review of recent advancements for UMDA. As mentioned, Dang and Lehre [10] were the first to prove upper bounds for UMDA on OneMax. If $\lambda \geq c \log n$ for a sufficiently large constant $c > 0$ and $\lambda \geq 13e\mu/(1-c')$ for an arbitrarily small constant $c' > 0$, then the expected runtime of UMDA on OneMax is $O(n\lambda \log \lambda)$. Hence, plugging in the smallest value of λ allowed in the statement, the bound is $O(n \log n \log\log n)$, i.e., slightly above the $O(n \log n)$ bound discussed above with respect to cGA and MMAS_{ib}.

Dang and Lehre used a powerful proof technique to obtain their bound. Interestingly, the so-called level-based theorem [9], which was originally developed for the analysis of population-based evolutionary algorithms, can be applied in this context. It was shown how the truncation selection of the best μ out of λ individuals leads to a reasonable chance of improving the best-so-far OneMax value and allows one to satisfy the other conditions of the level-based theorem with certain parameter settings. As a side-result, using the same proof technique, the bound $O(n\lambda + n^2)$ with respect to the LeadingOnes function was also obtained. Somewhat unusually, these proofs mostly consider populations instead of analyzing the values of single frequencies. For this to work, it is necessary that a frequency vector can be translated more or less unambiguously back into the population from which it was computed. This is possible in UMDA but not even in the slight generalization PBIL, where a frequency vector depends on the history of previous populations.

Obviously, the proof of the above-mentioned $O(n \log n \log\log n)$ bound immediately raised the question of whether this was the best possible runtime of UMDA on OneMax. Recently, two independent improvements of the bound were presented. The first one, due to Lehre and Nguyen [38], builds on a refinement of the level-based analysis, carefully using properties of the Poisson–binomial distribution, and is summarized by the following theorem. We emphasize that UMDA always refers to the algorithm with borders $1/n$ and $1-1/n$ on the frequencies, i.e., Algorithm 9.2 extended by a step that narrows all frequencies down to the interval $[1/n, 1-1/n]$.

Theorem 9.5.3 ([38]). *For some constant $a > 0$ and any constant $c \in (0,1)$, UMDA (with borders) with a parent population size $a \ln n \leq \mu \leq \sqrt{n(1-c)}$ and an offspring population size $\lambda \geq (13e)\mu/(1-c)$ has expected optimization time $O(n\lambda)$ on* OneMax.

Hence, Theorem 9.5.3 proves that the runtime of UMDA is $O(n \log n)$ for an appropriate choice of the parameters. This is tight owing to the recent lower bound $\Omega(n \log n)$ discussed below in Section 9.5.2.2. Interestingly, the set of appropriate choices for the $O(n \log n)$ behavior is confined to $\lambda = \Theta(\log n)$, which corresponds to a parameter choice below the above-mentioned phase transition, i.e., a choice where the algorithm exhibits severe

genetic drift. Also, the theorem includes a limit on μ, which is exactly in the regime of the phase transition. For greater values of μ and λ, Witt [66] independently derived runtime bounds (see the following theorem); this result also includes the regime covered by Lehre and Nguyen [38], albeit with an assumption about the ratio λ/μ.

Theorem 9.5.4 ([66]).

(a) Let $\lambda = (1+\beta)\mu$ for an arbitrary constant $\beta > 0$ and let $\mu \geq c\sqrt{n}\log n$ for some sufficiently large constant $c > 0$. Then the optimization time of UMDA, both with and without borders, on OneMax *is bounded from above by $O(\lambda\sqrt{n})$ with probability $\Omega(1)$. For UMDA with borders, the expected optimization time is also bounded in this way.*

(b) Let $\lambda = (1+\beta)\mu$ for an arbitrary constant $\beta > 0$ and let $\mu \geq c\log n$ for a sufficiently large constant $c > 0$ as well as $\mu = o(n)$. Then the expected optimization time of UMDA with borders on OneMax *is $O(\lambda n)$. For UMDA without borders, it is infinite with high probability if $\mu < c'\sqrt{n}\log n$ for a sufficiently small constant $c' > 0$.*

The two statements of Theorem 9.5.4 reflect the above-mentioned phase transition. For $\mu \geq c\sqrt{n}\log n$, as required in the first statement, the behavior is similar to that underlying Theorem 9.5.2 with respect to cGA and MMAS_{ib}. Frequencies move smoothly towards the upper border, and it is unlikely that frequencies will exhibit genetic drift towards smaller values than $1/3$. Hence, it is unlikely as well that UMDA without borders will get stuck with frequencies at 0. The runtime $O(n\log n)$ is obtained for $\lambda = c\sqrt{n}\log n$ for an appropriately large constant $c > 0$.

The second statement of Theorem 9.5.4 applies to a case where genetic drift is likely, but frequencies that have hit the lower border $1/n$ have a reasonable chance to recover in the given time span, which is $O(n\lambda)$ instead of only $O(\sqrt{n}\lambda)$ now. In fact, the analysis carefully considers the drift of frequencies from the lower towards the upper border and analyzes the probability that a frequency leaves its upper border again. To do so, a very careful analysis of the bias introduced by selecting the best μ individuals is required. Without such selection, a single frequency would correspond to a so-called martingale, but, owing to selection, there is a small drift upwards, similarly to what we described with respect to cGA above. Hence, the proof of Theorem 9.5.4 also gives insights into the stochastic process described by single frequencies. It is more involved than that for cGA since UMDA can change frequencies globally instead of only by $\pm 1/K$. The runtime $O(n\log n)$ can be obtained again, this time for $\lambda = c\sqrt{n}\log n$.

It is worth pointing out that Theorems 9.5.3 and 9.5.4 make nonoverlapping statements. Theorem 9.5.4 also applies to λ above the phase transition and describes a transition of $O(n\lambda)$ to $O(n\sqrt{\lambda})$ in the runtime. However, it crucially assumes $\lambda = (1+\Theta(1))\mu$ in both statements, an assumption that was also useful in earlier analyses of EDAs [59] but restricts the generality of

the statements. In contrast, Theorem 9.5.3 applies to settings such as $\mu = 1$, $\lambda = c \log n$ and shows the $O(n \log n)$ bound also for this somewhat extreme choice of parameters.

We conclude this discussion of upper bounds by summarizing a recent study by Wu et al [69], who presented the first runtime analysis of PBIL (called the cross-entropy (CE) method in their paper). Using $\mu = n^{1+\varepsilon} \log n$ for some constant $\varepsilon > 0$ and $\lambda = \omega(\mu)$, they obtained the result that the runtime of PBIL on OneMax is $O(\lambda n^{1/2+\varepsilon/3}/\rho)$ with overwhelming probability. Hence, if $\rho = \Omega(1)$, including the special case $\rho = 1$, where PBIL collapses to UMDA, a runtime bound of $O(n^{3/2+(4/3)\varepsilon} \log n)$ holds, i.e., slightly above $n^{3/2}$. In light of the detailed analyses of UMDA presented above, one may conjecture that this bound is not tight even if $\rho < 1$ is used, i.e., PBIL actually uses its learning approach to include solutions from several previous generations in the probabilistic model. In addition to that, a bound of the type $O(n^{2+\varepsilon})$ on LeadingOnes is obtained if $\rho = \Omega(1)$, $\mu = n^{\varepsilon/2}$ and $\lambda = \Omega(n^{1+\varepsilon})$. Technically, Wu et al [69] used concentration bounds such as Chernoff bounds to bound the effect of genetic drift, as well as anti-concentration results, in particular for the Poisson–binomial distribution, to obtain their statements. All bounds hold with high probability only, since PBIL is formulated without borders. Probably, using a more detailed analysis of genetic drift and applying modern drift theorems, the bound for LeadingOnes can be improved to an expected $O(n^2)$ runtime for all $\rho = \Omega(1)$, provided that the classical borders $\{1/n, 1-1/n\}$ are used.

9.5.2.2 Lower Bounds for OneMax

Deriving lower bounds on the runtime of EDAs is often more challenging than deriving upper bounds. Roughly, most existing approaches show that the probabilistic model is not sufficiently adjusted towards the set of optimal solutions within a given time span. A relatively straightforward approach relates the runtime to the strength of updates in the algorithm. With respect to simple univariate algorithms such as cGA and UMDA, one can show that frequencies do not increase by more than $1/K$ (with probability 1) or $O(1/\mu)$ (in expectation, assuming $\lambda = (1 + \Theta(1))\mu$) in a step. This naturally leads to a lower bound of $\Omega(K)$ or $\Omega(\mu)$, respectively, on the runtime on OneMax. However, the bound is weak, as it pessimistically assumes that each generation changes frequencies in the right direction. More detailed analyses reveal that cGA, in the early phases of the optimization process, has only a probability of $O(1/\sqrt{n})$ of performing a step where the two offspring differ in fewer than two bits, i.e., the probability that the outcome of a certain bit is relevant for selection is then only $O(1/\sqrt{n})$ [63]. Similar results can be obtained for UMDA [36]. Thus, each bit only moves by up to an expected amount of $O(1/(K\sqrt{n}))$ or $O(1/(\mu\sqrt{n}))$, respectively, per generation. Then a drift analysis translates this into the lower bounds $\Omega(K\sqrt{n})$ and $\Omega(\mu\sqrt{n})$

that appear in the following theorems. The first bound was already known for cGA without borders from Droste's work [17].

Theorem 9.5.5 ([63]). *The optimization time of cGA (with borders) with* $K \leq \text{poly}(n)$ *on* OneMax *is* $\Omega(K\sqrt{n} + n\log n)$ *with high probability and in expectation.*

Theorem 9.5.6 ([36]). *Let* $\lambda = (1+\beta)\mu$ *for some constant* $\beta > 0$ *and* $\lambda \leq \text{poly}(n)$. *Then the expected optimization time of UMDA on* OneMax *is* $\Omega(\mu\sqrt{n} + n\log n)$ *(both with and without borders).*

Sudholt and Witt [63] also stated Theorem 9.5.5 in an analogous fashion for MMAS_{ib}, with the parameter K replaced by $1/\rho$. As its working principle is rather similar to that of cGA, we do not discuss MMAS_{ib} further in this section.

The lower bounds $\Omega(K\sqrt{n})$ and $\Omega(\mu\sqrt{n})$ we have illustrated so far are very weak if K and μ, respectively, are small. In fact, they can be even worse than the bounds $\Omega(n/\log n)$ that follow from black-box complexity [18]. Until 2016, it was not clear whether the runtime of these simple EDAs was also bounded by $\Omega(n\log n)$ or whether they could possibly optimize OneMax in $o(n\log n)$ time and hence be faster than simple evolutionary algorithms. A negative answer was given by the two above theorems, both of which also contain an $\Omega(n\log n)$ term.

The proof of the bound $\Omega(n\log n)$ is technically demanding. It relies on the following strategy:

(a) Show that with high probability several frequencies, for example $\sqrt{n}$ of them, reach the lower border before the optimum is sampled. This requires a detailed analysis of the stochastic behavior of several dependent, single frequencies instead of considering merely the sum $P_t := \sum_{i=1}^{n} \boldsymbol{p}_i^{(t)}$ of the frequencies, whose stochastic behavior is already quite well understood and can relatively easily be analyzed by drift analysis, as sketched in the paragraph following Theorem 9.5.2. In fact, in the detailed analysis of single frequencies, it is even required to show that some frequencies walk to the lower border while most other frequencies do not move up too far to the upper border; otherwise one cannot rule out with sufficiently high probability the possibility that the optimum is sampled in the meantime.

(b) Once polynomially many frequencies have reached the lower border $1/n$, a so-called coupon collector effect arises. A relatively straighforward generalization of the coupon collector theorem [44, 45] to the case where still polynomially many bits have to be corrected, where a correction is made with probability at most $1/n$, yields the following statement: *Assume cGA reaches a situation where at least* $\Omega(n^{\varepsilon})$ *frequencies attain the lower border* $1/n$. *Then, with high probability and in expectation, the remaining optimization time is* $\Omega(n\log n)$. The underlying modification of the coupon collector theorem may be called folklore in probability theory,

but is interesting for its own sake: collecting the last n^{ε} coupons takes asymptotically the same time as collecting them all.

A major effort is required to flesh out the behavior sketched in item (a) above. Roughly speaking, one exploits the fact that frequencies behave similarly to a martingale and can walk to the lower border owing to genetic drift. However, the effect of genetic drift is dependent on many factors. When all frequencies have reached a border, genetic drift is much less pronounced than in situations where many frequencies are close to the median value 1/2 (which is initially the case). To handle this dependency on time, it has to be shown that some frequencies move unusually fast, which means faster than the expected time, to the lower border while the majority of the frequencies is still at a medium value. More precisely, the proofs approximate the hitting time of the lower border by a normally distributed random variable, which is not sharply concentrated around the mean and exhibits exactly the desired reasonable probability of deviating from the mean. Additionally, the drift analysis features a novel use of potential functions that smooth out the variances of the movements of frequencies, which would be place-dependent and not applicable to the approximation by a normal distribution otherwise.

Second Phase Transition Around $\log n$

Not much research has been done on very small values of the population size λ and K in UMDA and cGA, corresponding to very large ρ in MMAS_{ib}. Neumann et al [50] gave an exponential bound on the runtime of MMAS_{ib} if $\rho \geq c/\log n$, indicating a second phase transition in behavior around $\log n$. Roughly speaking, if the set of possible values for a frequency becomes less than $\log n$, then the scale is too coarse for the probabilistic model to adjust slowly towards the set of optimal solutions. For example, even after a frequency has reached its maximum $1 - 1/n$ once, an unlucky step may lead to a drastic decline in frequency which, on average, cannot be recovered in polynomial time. It is conjectured that cGA and UMDA will not optimize OneMax in polynomial time either if $K \leq c \log n$ or if $\lambda \leq c \log n$, respectively, for a small constant $c > 0$.

Major Open Problems

Even if we ignore the values below $\log n$ corresponding to the second phase transition just mentioned, the lower bounds given in Theorems 9.5.5 and 9.5.6 still do not give a complete picture of the runtime of the algorithms on OneMax. For example, for μ in the medium regime between the phase transitions, i.e., when μ is both $\omega(\log n)$ and $o(\sqrt{n} \log n)$, it is not clear whether a lower bound of the kind $\Omega(\mu n)$ (which would match the upper bound given above in Theorem 9.5.4) or any other runtime $\omega(n \log n)$ holds. It is an open

problem to prove tight bounds on the runtime of simple EDAs in this medium regime. As usual, we expect analyses to be harder for UMDA than for cGA, as the former algorithm can change frequencies in a global way, while the latter only changes them locally by $\pm 1/K$.

Some progress on the way to tight bounds has been made very recently by Lengler et al. [42], who proved a lower bound of $\Omega(K^{1/3}n + n\log n)$ on the expected optimization time of the cGA if $K = O(n^{1/2}/(\log n \log\log n))$. Hence, the expected optimization time will be $\Omega(n^{7/6}/(\log n \log\log n))$ for $K = O(n^{1/2}/(\log n \log\log n))$, while it is bounded from above by $O(n\log n)$ for $K = cn^{1/2}$ if c is chosen as a sufficiently large constant. Hence, the runtime seems to depend in a multimodal way on K. Nevertheless, this still remains a conjecture, since there are no upper bounds on the runtime of the cGA for $K = o(n^{1/2})$; there are only upper bounds for the UMDA if $\lambda = o(n^{1/2})$ that support this conjecture.

A summary of proven upper and conjectured bounds on the runtime of UMDA on OneMax is displayed in Fig. 9.2. We believe that similar results hold for cGA and MMAS_{ib}, with λ replaced by K and $1/\rho$, respectively.

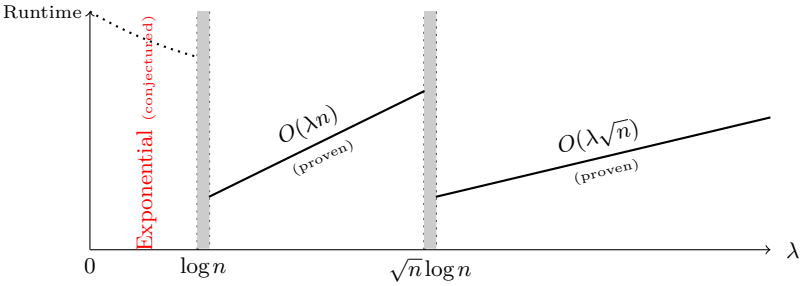

Fig. 9.2 Picture of runtime bounds with UMDA on OneMax, assuming $\lambda = (1 + \Theta(1))\mu$.

We have carried out experiments for UMDA on OneMax to gain some empirical insights into the relationship between λ and the runtime. The algorithm was implemented in the C programming language using the WELL512a random number generator. The problem size was set to $n = 2000$, λ was increased from 14 to 350 in steps of size 2, μ was set to $\lambda/2$, and, owing to the high variance of the runs, especially for small λ, an average was taken over 3000 runs for every setting of λ. The left-hand side of Fig. 9.3 demonstrates that the runtime in fact shows a multimodal dependence on λ. Starting from very high values, it has a minimum at $\lambda \approx 20$ and then increases again up to $\lambda \approx 70$. Thereafter it falls again up to $\lambda \approx 280$, and finally increases rather steeply for the rest of the range. The right-hand side also illustrates that the number of times the lower border is hit seems to decrease exponentially with

λ. The phase transition where the behavior of frequencies turns from chaotic into stable is empirically located somewhere between 250 and 300.

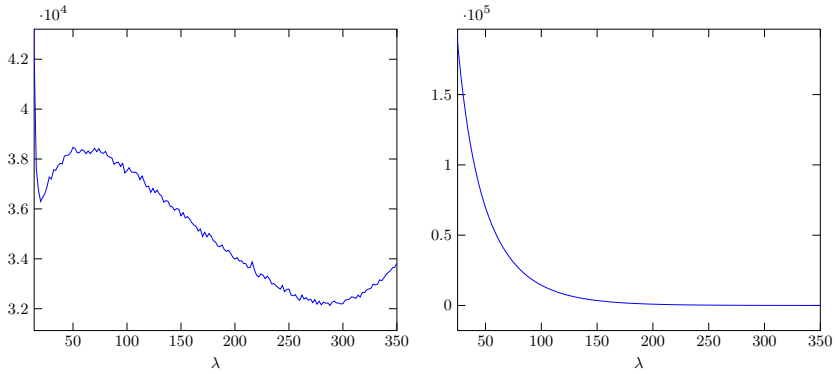

Fig. 9.3 Left-hand side: empirical runtime of UMDA on OneMax, right-hand side: number of hits of lower border; for $n = 2000$, $\lambda \in \{14, 16, \ldots, 350\}$, $\mu = \lambda/2$, and averaged over 3000 runs

9.5.2.3 New Advances in Tackling Genetic Drift

Genetic drift slows down optimization because it basically adds a random signal to the objective function. One reason why this impacts the algorithms is their myopic behavior: they have to perform an update to their frequencies based only on information from the current iteration. Especially if this sample size is small, such as for cGA or MMAS_{ib}, the amount of information gained during a single iteration may be too small to perform a sensible decision with respect to the update.

In order to counteract such ill-informed updates, Doerr and Krejca [12] proposed a new EDA that tries to reduce the number of incorrect frequency updates by relying not only on information from a single iteration but also information from multiple previous iterations. Their *significance-based cGA* (sig-cGA) stores a frequency vector, like an n-Bernoulli-λ-EDA, but additionally also stores a history H_i for each bit position i. In each iteration, only two offspring are sampled, and the bits of the better individual are saved in the respective histories. Then the algorithm checks, for each history, whether a *significance* occurs, that is, whether the number of 1s or 0s saved is drastically more than expected when assuming that each 1 occurs with probability $\boldsymbol{p}_i$. The level of confidence can be regulated by a parameter called ε. If a significance of 1s is detected at a position, the respective frequency is set to $1 - 1/n$; if a significance of 0s is detected, the frequency is set to $1/n$; otherwise, the frequency is left unchanged. Overall, the algorithm uses only three

different frequency values: $1-1/n$, $1/2$, and $1/n$, where $1/2$ is used only as a starting value – if a frequency once takes a value different from $1/2$, it never returns.

This significance-based approach allows sig-cGA, at the beginning of an optimization, to keep frequencies at $1/2$ until there is statistical proof that another value would be more beneficial. Thus, it can be thought of as an algorithm that is both balanced and stable.[4] The usefulness of this approach was shown by proving that this algorithm optimizes OneMax and LeadingOnes both in time $O(n \log n)$ in expectation and with high probability, which has not been proven for any other EA or EDA before [12].

9.5.3 Noisy Settings

In real-world optimization, the evaluation of a solution often involves a degree of uncertainty due to inaccuracies in the evaluation process. We call this uncertainty in the fitness *noise*. Since EDAs, as general-purpose heuristics, build on this inaccurate information, it is interesting to analyze how they perform when faced with noise.

Most EDA scenarios with noise consider ACO variants on single-destination shortest-path problems, mostly not in the context of EDAs at all [13, 19, 33, 62]. However, some results have analyzed pseudo-Boolean optimization [23, 24].

9.5.3.1 Combinatorial Optimization

Horoba and Sudholt [33] considered an acyclic weighted graph and were interested in finding a shortest path from each vertex to a single given destination. The noise was modeled by drawing a random nonnegative value η per edge weight w, possibly dependent on the edge, and its new weight w' was determined by $w' = w(1+\eta)$. Thus, depending on the distribution of η, large weights increase more than small weights. The algorithm of interest was an ACO variant. This constructs paths from each node to the destination, using the perturbed weights and choosing an edge with a probability related to its pheromone value with respect to the pheromones of all competing edges. The algorithm compares each constructed path with the currently best-so-far solution per node without reevaluation. That means that the best-so-far solutions, as well as their possibly perturbed weights, are stored and used for lookup.

These authors provided instances in which the algorithm does not find a desired approximation within polynomial time with high probability. This is

[4] Since sig-cGA is not an n-Bernoulli-λ-EDA (owing to the histories that store data from multiple iterations), the actual definitions of *balanced* and *stable* do not apply.

Table 9.1 Expected runtimes (number of fitness evaluations) of various EDAs until they first find an optimum for the three functions OneMax (9.1), LeadingOnes (9.3), and BinVal (9.2)

Algorithm	OneMax	Constraints	LeadingOnes	Constraints	BinVal	Constraints
UMDA/PBIL[5]	$\Omega(\lambda\sqrt{n} + n\log n)$ (Thm. 9.5.6)	$\mu = \Theta(\lambda), \lambda = O\big(\text{poly}(n)\big)$	$O(n\lambda\log\lambda + n^2)$ [10]	$\lambda = \Omega(\log n)$, $\mu = \Theta(\lambda)$	Unknown	–
	$O(\lambda n)$ (Thm. 9.5.3 and 9.5.4)	$\mu = \Omega(\log n) \cap O(\sqrt{n}), \lambda = \Omega(\mu)$ or $\mu = \Omega(\log n) \cap o(n), \mu = \Theta(\lambda)$				
	$O(\lambda\sqrt{n})$ (Thm. 9.5.4)	$\mu = \Omega(\sqrt{n}\log n), \mu = \Theta(\lambda)$				
cGA/2-MMAS$_{\text{ib}}$	$\Omega\Big(\frac{\sqrt{n}}{\rho} + n\log n\Big)$ (Thm. 9.5.5)	$\frac{1}{\rho} = O\big(\text{poly}(n)\big)$	Unknown	–	$\Omega(\min\{n^2, Kn\})$ (Thm. 9.5.1)[6]	None
	$O\Big(\frac{\sqrt{n}}{\rho}\Big)$ (Thm. 9.5.2)	$\frac{1}{\rho} = \Omega(\sqrt{n}\log n) \cap O\big(\text{poly}(n)\big)$			$O(Kn)$ (Thm. 9.5.1)[6]	$K = n^{1+\varepsilon}$, $\varepsilon = \Theta(1)$
sig-cGA	$O(n\log n)$ [12]	$\varepsilon > 12$	$O(n\log n)$ [12]	$\varepsilon > 12$	Unknown	–

[5] The results shown for PBIL are the results for UMDA, since the latter is a special case of the former. Wu et al [69] also analyzed PBIL but with worse results.

[6] This result was only proven for cGA.

due to the best-so-far solution not being reevaluated. Thus, if a nonoptimal path is evaluated to be very good by chance, it will get reinforced many times, making it more unlikely that the algorithm will sample other paths that, additionally, have to be evaluated even better. However, these authors also proved that optimization will succeed if the noise follows the same distribution for every edge.

Sudholt and Thyssen [62] extended the results of Horoba and Sudholt [33] by considering a larger range of noise distributions, showing how long it takes to approximate optimal solutions or even when optimization succeeds.

Doerr et al [13] considered a similar scenario to the one analyzed by Horoba and Sudholt [33], the difference being that the weights of the graphs were purely random, i.e., there was no groundtruth to rely on. This setting makes it harder to define what an optimal solution actually is.

The authors of [13] first considered a multigraph consisting of two nodes with multiple edges between those nodes. They called an edge *preferred* if its probability of being shorter than any other edge from the same vertex was at least $1/2 + \delta$, where $\delta > 0$ is a constant, and they stated how this scenario relates to armed-bandit settings. Using the same ACO algorithm as Horoba and Sudholt [33] but reevaluating the best-so-far solution each iteration, they gave an upper bound on the expected time until the pheromone on the preferred edge was maximal. They then provided examples of weight distributions that result in an edge being preferred. The paper concludes with a more general graph setting that assumes that there exists an inductively defined set of edges S, starting at a given node, such that each edge that extends paths using edges from S is preferred. The authors of the paper gave an upper bound, in the case where S is a tree, on the expected time until the ACO variant considered maximizes the pheromones on all of the edges in S.

Feldmann and Kötzing [19] analyzed the same setting as Doerr et al [13] but investigated another ACO variant: MMAS-fp. This algorithm does not store best-so-far solutions but always makes an update with respect to the current samples; however, the update is done with respect to each sample's fitness. Thus, good solutions yield larger changes in the update than bad solutions. The authors of [19] explained the difference in this approach with respect to those of Horoba and Sudholt [33] and Doerr et al [13] by saying that MMAS-fp optimizes paths that are shortest in expectation. They proved this claim by providing upper bounds on the expected number of iterations until MMAS-fp finds expected shortest paths in graphs where, for each node, the difference between the expected lengths of different outgoing edges can be lower-bounded by a value $\delta > 0$, which influences the runtime.

9.5.3.2 Pseudo-Boolean Optimization

Friedrich et al [23] (conference version [21]) also considered MMAS-fp, just like Feldmann and Kötzing [19], but in the setting of optimizing linear pseudo-

Boolean functions. The noise was mostly modeled as Gaussian *additive posterior noise*, i.e., when evaluating the fitness of an individual, a normally distributed random variable is added to the fitness, every time anew and independently. Friedrich et al [23] showed that MMAS-fp *scales gracefully* in this scenario. That means that, for every polynomially bounded variance of the noise, there is a configuration of MMAS-fp such that the runtime is polynomially bounded as well. Since the runtime results hold with high probability, by performing an uninformed binary search using restarts, the correct variance of a problem with Gaussian noise can be guessed correctly within polynomial time. Thus, MMAS-fp can be modified such that the runtime is, with high probability, polynomial if the variance of the noise is.

Additionally, the authors of [23] extended their results to posterior noise other than Gaussian. Further, they considered a *prior noise* model where, before evaluating the fitness of an individual, a uniformly randomly chosen bit is flipped. In both of these settings, they proved that the algorithm scales gracefully.

Friedrich et al [24] (conference version [20]) also considered cGA under the Gaussian additive posterior noise model. As for MMAS-fp, they proved that the algorithm scales gracefully. Further, they showed that the $(\mu + 1)$ EA, a commonly analyzed EA, does not scale gracefully. Both of the results of Friedrich et al [23, 24] suggest that EDAs are inherently more tolerant to noise than standard EAs, as the EDAs did not need to be modified to cope with noise, except for choosing correct parameters. These authors also compared the restart version of cGA with an approach that uses resampling in order to basically remove the noise in the fitness, as described by Akimoto et al [2]. Since the number of resamples is closely tied to the noise's variance, the cGA variant using restarts instead of resampling emerges victorious.

9.6 Conclusions and Open Problems

We have given an overview of the state of the art in the theory of discrete EDAs, where the most recent research surpasses convergence analyses and instead deals with the runtime of especially simple univariate EDAs such as cGA, UMDA, and PBIL. In this domain, increasingly precise results have been obtained with respect to well-established benchmark problems such as OneMax, but, as we have emphasized in this chapter, there are several open problems even for this simple problem. In particular, a complete picture of the runtime of the simple EDAs depending on their parameters is still missing. We think that further results for benchmark functions will give insight into the right choice of specific EDAs, including the choice of parameters such as the population size and the borders on the frequencies depending on the problem characteristics. We also expect that this research will lead to runtime results and advice on the choice of algorithms and parameters with respect

to more practically relevant combinatorial optimization problems. Here in particular, noisy settings or, more generally, optimization under uncertainty seem to represent scenarios where EDAs can outperform classical evolutionary algorithms. Also, the combinatorial structure may favor the application of multivariate EDAs, a type of EDA for which almost no theoretical results exist yet.

Acknowledgements Carsten Witt was supported by a grant from the Danish Council for Independent Research (DFF-FNU 4002-00542). Support by the COST Action 15140 "Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice" (ImAppNIO) is also gratefully acknowledged.

References

[1] Afshani P, Agrawal M, Doerr B, Doerr C, Larsen KG, Mehlhorn K (2013) The query complexity of finding a hidden permutation. In: Space-Efficient Data Structures, Streams, and Algorithms – Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday, pp 1–11, DOI 10.1007/978-3-642-40273-9_1

[2] Akimoto Y, Astete-Morales S, Teytaud O (2015) Analysis of runtime of optimization algorithms for noisy functions over discrete codomains. Journal of Theoretical Computer Science 605:42:50, DOI 10.1016/j.tcs.2015.04.008

[3] Asoh H, Mühlenbein H (1994) On the mean convergence time of evolutionary algorithms without selection and mutation. In: Proc. of PPSN '94, pp 88–97

[4] Baluja S (1994) Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA

[5] Chen T, Tang K, Chen G, Yao X (2007) On the analysis of average time complexity of estimation of distribution algorithms. In: Proc. of CEC '07, pp 453–460

[6] Chen T, Lehre PK, Tang K, Yao X (2009) When is an estimation of distribution algorithm better than an evolutionary algorithm? In: Proc. of CEC '09, pp 1470–1477

[7] Chen T, Tang K, Chen G, Yao X (2009) Rigorous time complexity analysis of univariate marginal distribution algorithm with margins. In: Proc. of CEC '09, pp 2157–2164

[8] Chen T, Tang K, Chen G, Yao X (2010) Analysis of computational time of simple estimation of distribution algorithms. IEEE Transactions on Evolutionary Computation 14(1):1–22

[9] Corus D, Dang DC, Eremeev AV, Lehre PK (2017) Level-based analysis of genetic algorithms and other search processes. IEEE Transactions on Evolutionary Computation 22(5):707–719
[10] Dang D, Lehre PK (2015) Simplified runtime analysis of estimation of distribution algorithms. In: Proc. of GECCO '15, pp 513–518
[11] De Bonet JS, Isbell CL Jr, Viola PA (1997) MIMIC: Finding optima by estimating probability densities. In: Proc. of NIPS '96, pp 424–430
[12] Doerr B, Krejca MS (2018) Significance-based estimation-of-distribution algorithms. In: Proc. of GECCO '18, pp 1483-1490
[13] Doerr B, Hota A, Kötzing T (2012) Ants easily solve stochastic shortest path problems. In: Proc. of GECCO '12, pp 17–24, DOI 10.1145/2330163.2330167
[14] Doerr B, Johannsen D, Winzen C (2012) Multiplicative drift analysis. Algorithmica 64(4):673–697, DOI 10.1007/s00453-012-9622-x
[15] Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press
[16] Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Tech. Rep. 91–016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy
[17] Droste S (2006) A rigorous analysis of the compact genetic algorithm for linear functions. Natural Computing 5(3):257–283, preliminary version in GECCO '05
[18] Droste S, Jansen T, Wegener I (2006) Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems 39:525–544
[19] Feldmann M, Kötzing T (2013) Optimizing expected path lengths with ant colony optimization using fitness proportional update. In: Proc. of FOGA '13, pp 65–74, DOI 10.1145/2460239.2460246
[20] Friedrich T, Kötzing T, Krejca MS, Sutton AM (2015) The benefit of recombination in noisy evolutionary search. In: Proc. of ISAAC '15, pp 140–150
[21] Friedrich T, Kötzing T, Krejca MS, Sutton AM (2015) Robustness of ant colony optimization to noise. In: Proc. of GECCO '15, pp 17–24
[22] Friedrich T, Kötzing T, Krejca MS (2016) EDAs cannot be balanced and stable. In: Proc. of GECCO '16, pp 1139–1146, DOI 10.1145/2908812.2908895
[23] Friedrich T, Kötzing T, Krejca MS, Sutton AM (2016) Robustness of ant colony optimization to noise. Evolutionary Computation 24(2):237–254
[24] Friedrich T, Kötzing T, Krejca MS, Sutton AM (2017) The compact genetic algorithm is efficient under extreme gaussian noise. IEEE Transactions on Evolutionary Computation 21(3):477–490
[25] Gao Y, Culberson J (2005) Space complexity of estimation of distribution algorithms. Evolutionary Computation 13(1):125–143, DOI 10.1162/1063656053583423

[26] González C, Lozano J, Larrañaga P (2000) Analyzing the PBIL algorithm by means of discrete dynamical systems. Complex Systems 12(4):465–479
[27] Harik G, Lobo FG, Goldberg DE (1998) The compact genetic algorithm. IEEE Transactions on Evolutionary Computation pp 523–528
[28] Harik GR, Lobo FG, Sastry K (2006) Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In: [54], pp 39–61
[29] Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm and Evolutionary Computation 1(3):111–128
[30] He J, Yao X (2001) Drift analysis and average time complexity of evolutionary algorithms. Artificial Intelligence 127(1):57–85, DOI 10.1016/S0004-3702(01)00058-3
[31] He J, Yao X (2004) A study of drift analysis for estimating computation time of evolutionary algorithms. Natural Computing 3(1):21–35, DOI 10.1023/B:NACO.0000023417.31393.c7
[32] Höhfeld M, Rudolph G (1997) Towards a theory of population-based incremental learning. In: Proc. of ICEC '97, pp 1–5
[33] Horoba C, Sudholt D (2010) Ant colony optimization for stochastic shortest path problems. In: Proc. of GECCO '10, pp 1465–1472, DOI 10.1145/1830483.1830750
[34] Janusz Kacprzyk WP (ed) (2015) Springer Handbook of Computational Intelligence. Springer, DOI 10.1007/978-3-662-43505-2
[35] Johannsen D (2010) Random combinatorial structures and randomized search heuristics. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany and the Max-Planck-Institut für Informatik
[36] Krejca MS, Witt C (2017) Lower bounds on the run time of the univariate marginal distribution algorithm on OneMax. In: Proc. of FOGA '17, pp 65–79, DOI 10.1145/3040718.3040724
[37] Larrañaga P, Lozano JA (2002) Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Genetic Algorithms and Evolutionary Computation, vol 2. Springer
[38] Lehre PK, Nguyen PTH (2017) Improved runtime bounds for the univariate marginal distribution algorithm via anti-concentration. In: Proc. of GECCO '17, pp 1383–1390, DOI 10.1145/3071178.3071317
[39] Lehre PK, Witt C (2010) Black-box search by unbiased variation. In: Proc. of GECCO '10, pp 1441–1448
[40] Lehre PK, Witt C (2014) Concentrated hitting times of randomized search heuristics with variable drift. In: Proc. of ISAAC '14, pp 686–697, DOI 10.1007/978-3-319-13075-0
[41] Lehre PK, Witt C (2017) General drift analysis with tail bounds, arXiv:1307.2559
[42] Lengler J, Sudholt D, Witt C (2018) Medium step sizes are harmful for the compact genetic algorithm. In: Proc. of GECCO '18, pp 1499–1506

[43] Mitavskiy B, Rowe JE, Cannings C (2009) Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. International Journal of Intelligent Computing and Cybernetics 2(2):243–284, DOI 10.1108/17563780910959893
[44] Mitzenmacher M, Upfal E (2005) Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press
[45] Motwani R, Raghavan P (1995) Randomized Algorithms. Cambridge University Press
[46] Mühlenbein H (1992) How genetic algorithms really work: Mutation and hillclimbing. In: Proc. of PPSN '92, pp 15–26
[47] Mühlenbein H, Mahnig T (1999) Convergence theory and applications of the factorized distribution algorithm. Journal of Computing and Information Technology 7:19–32
[48] Mühlenbein H, Mahnig T (1999) FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. Evolutionary Computation 7(4):353–376, DOI 10.1162/evco.1999.7.4.353
[49] Mühlenbein H, Paass G (1996) From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In: Proc. of PPSN '96, pp 178–187
[50] Neumann F, Sudholt D, Witt C (2010) A few ants are enough: ACO with iteration-best update. In: Proc. of GECCO '10, pp 63–70
[51] Ollivier Y, Arnold L, Auger A, Hansen N (2017) Information-geometric optimization algorithms: A unifying picture via invariance principles. Journal of Machine Learning Research 18:1–65
[52] Pelikan M, Mühlenbein H (1999) The bivariate marginal distribution algorithm. In: Advances in Soft Computing, Springer, pp 521–535
[53] Pelikan M, Goldberg DE, Cantú-Paz E (1999) BOA: The bayesian optimization algorithm. In: Proc. of GECCO '99, pp 525–532
[54] Pelikan M, Sastry K, Cantú-Paz E (2006) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Studies in Computational Intelligence, vol 33. Springer
[55] Pelikan M, Hauschild M, Lobo FG (2015) Estimation of distribution algorithms. In: [34], pp 899–928, DOI 10.1007/978-3-662-43505-2_45
[56] Rudolph G (1997) Convergence properties of evolutionary algorithms. Verlag Dr. Kovač
[57] Shapiro JL (2003) The sensitivity of PBIL to its learning rate, and how detailed balance can remove it. In: Proc. of FOGA '02, pp 115–132
[58] Shapiro JL (2005) Drift and scaling in estimation of distribution algorithms. Evolutionary Computation 13(1):99–123, DOI 10.1162/1063656053583414
[59] Shapiro JL (2006) Diversity loss in general estimation of distribution algorithms. In: Proc. of PPSN '06, Springer, pp 92–101
[60] Simon D (2013) Evolutionary Optimization Algorithms. John Wiley & Sons

[61] Stützle T, Hoos HH (2000) MAX–MIN ant system. Future generation computer systems 16(8):889–914
[62] Sudholt D, Thyssen C (2012) A simple ant colony optimizer for stochastic shortest path problems. Algorithmica 64(4):643–672, DOI 10.1007/s00453-011-9606-2
[63] Sudholt D, Witt C (2016) Update strength in EDAs and ACO: How to avoid genetic drift. In: Proc. of GECCO '16, pp 61–68
[64] Vose MD (1999) The Simple Genetic Algorithm: Foundations and Theory. MIT Press
[65] Wald A (1944) On cumulative sums of random variables. The Annals of Mathematical Statistics 15(3):283–296, DOI 10.1214/aoms/1177731235
[66] Witt C (2017) Upper bounds on the runtime of the univariate marginal distribution algorithm on OneMax. In: Proc. of GECCO '17, pp 1415–1422, DOI 10.1145/3071178.3071216
[67] Witt C (2018) Domino convergence: why one should hill-climb on linear functions. In: Proc. of GECCO '18, ACM Press, to appear
[68] Wu Z, Kolonko M (2014) Asymptotic properties of a generalized cross-entropy optimization algorithm. IEEE Transactions on Evolutionary Computation 18(5):658–673, DOI 10.1109/TEVC.2014.2336882
[69] Wu Z, Kolonko M, Möhring RH (2017) Stochastic runtime analysis of the cross-entropy algorithm. IEEE Transactions on Evolutionary Computation 21(4):616–628, DOI 10.1109/TEVC.2017.2667713
[70] Zhang Q (2004) On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. IEEE Transaction on Evolutionary Computation 8(1):80–93, DOI 10.1109/TEVC.2003.819431
[71] Zhang Q (2004) On the convergence of a factorized distribution algorithm with truncation selection. Complexity 9(4):17–23, DOI 10.1002/cplx.20013
[72] Zhang Q, Mühlenbein H (2004) On the convergence of a class of estimation of distribution algorithms. IEEE Transactions on Evolutionary Computation 8(2):127–136