Theory of estimation-of-distribution algorithms

Witt, Carsten

Published in:
Proceedings of 2019 Genetic and Evolutionary Computation Conference

Link to article, DOI:
10.1145/3319619.3323367

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
https://doi.org/10.1145/3319619.3323367
Theory of Estimation-of-Distribution Algorithms

Carsten Witt

DTU Compute
Technical University of Denmark

Tutorial at GECCO 2019
Updated version: https://tinyurl.com/eda2019

Overview

- Introduction to estimation-of-distribution algorithms (EDAs)
- What do we mean by theory?
- Presentation of important EDAs in theory
- Main results: from convergence to runtime analysis
- Conclusions

Instructor

Carsten Witt is an associate professor at the Technical University of Denmark. He received his M.Sc. in 2000 and Ph. D. degree in 2004, both in Computer Science from the Technical University of Dortmund, Germany. His main expertise is in the algorithmic analysis of metaheuristics, including evolutionary algorithms, ant colony optimization and estimation-of-distribution algorithms. Carsten has over 80 peer-reviewed publications and has given tutorials about bio-inspired computation in combinatorial optimization at previous GECCO and PPSN conferences. He is member of the editorial boards of Evolutionary Computation Journal and Theoretical Computer Science.

Evolutionary vs. Estimation-of-Distribution Algorithms

**Evolutionary Algorithms (EAs)**
- Work with populations of search points
- Modify the search points through, e.g., mutation and crossover
- Select promising search points based on fitness

**EDAs**
- Work with probability distributions
- Sample search points based on current distribution
- Adjust distribution based on the most promising search points
A Classical Evolutionary Algorithm

An Estimation-of-Distribution Algorithm

How an EA Approaches the Optimum
How an EA Approaches the Optimum

In the image, there are two sequential graphs showing the approach of an optimization algorithm to the optimum. The graphs illustrate the movement of a point (representing the population of solutions) along a curve, indicating the improvement in the objective function value as the algorithm progresses towards the global optimum. The left graph shows the initial state, and the right graph shows the final state after the algorithm has adjusted its probabilistic model.
How an EDA Adjusts the Probabilistic Model
Important EDAs

Domain: discrete optimization, e.g., find maximum for \( f : \{0,1\}^n \rightarrow \mathbb{R} \).

Important distinction: univariate vs. multivariate EDAs.

For example: \( f(x_1, x_2, x_3) = -x_1 + 2x_1x_2 + x_3 \). Good to learn learn dependency between \( x_1 \) and \( x_2 \).

Univariate

- cGA
- UMDA
- PBIL
- MMAS
- ...

Multivariate

- FDA
- ECGA
- MIMIC, BMDA
- BOA
- ...

Most theoretical results concern univariate EDAs.

Some Benchmark Functions

Theoretical results often consider simple problems, which we have to understand first.

- **OneMax** \((x_1, \ldots, x_n) := \sum_{i=1}^{n} x_i\)

- **BinVal** \((x_1, \ldots, x_n) := \sum_{i=1}^{n} 2^{n-i} x_i\)

- **LeadingOnes** \((x_1, \ldots, x_n) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j\)

Illustrate simple but fundamental properties; re-appear in more complex scenarios.

Write OneMax but mean \( n - \text{Ham}(x, a) \) for unknown string \( a \in \{0,1\}^n \).

Univariate Algorithms

Common concept: the frequency vector (aka. marginal probabilities)

- Probabilities \((p_1, \ldots, p_n)\) for setting the individual bits to 1.
- Usually initialized as \( p_i = 1/2 \) for all \( i \).
- Independently sampled.
- Frequency vector adjusted over time.
Compact GA (cGA) (Harik et al., 1999)

Simulates behavior of a GA with population size \( K \) in a compact way.

\[
\begin{align*}
t &\leftarrow 0; \\
p_{t,1} &\leftarrow p_{t,2} \leftarrow \cdots \leftarrow p_{t,n} \leftarrow 1/2; \\
\text{while termination criterion not met do} \\
\quad &\text{Create } x \text{ (i.e., } x_i = 1 \text{ with prob. } p_{t,i} \text{ and } x_i = 0 \text{ with prob. } 1 - p_{t,i}); \\
\quad &\text{Create } y; \\
\quad &\text{if } f(x) < f(y) \text{ then swap } x \text{ and } y; \\
\quad &\text{for } i \in \{1, \ldots, n\} \text{ do} \\
\quad &\quad \text{if } x_i > y_i \text{ then } p_{t+1,i} \leftarrow p_{t,i} + 1/K; \\
\quad &\quad \text{if } x_i < y_i \text{ then } p_{t+1,i} \leftarrow p_{t,i} - 1/K; \\
\quad &\quad \text{if } x_i = y_i \text{ then } p_{t+1,i} \leftarrow p_{t,i}; \\
\quad &\quad \text{Restrict } p_{t+1,i} \text{ to be within } [1/n, 1-1/n] \text{ ("borders");} \\
\quad &\quad t \leftarrow t + 1; \\
\end{align*}
\]

Parameter \( K \) determines preciseness of model. Big \( K \) = fine model = small update strength.

2-MMAS\(_{ib}\) (Neumann et al., 2010)

Max-Min Ant System (Stützle and Hoos, 2000) with iteration-best update

\[
\begin{align*}
t &\leftarrow 0; \\
p_{t,1} &\leftarrow p_{t,2} \leftarrow \cdots \leftarrow p_{t,n} \leftarrow 1/2; \\
\text{while termination criterion not met do} \\
\quad &\text{Create } x; \\
\quad &\text{Create } y; \\
\quad &\text{if } f(x) < f(y) \text{ then swap } x \text{ and } y; \\
\quad &\text{for } i \in \{1, \ldots, n\} \text{ do} \\
\quad &\quad \text{if } x_i = 1 \text{ then } p_{t+1,i} \leftarrow p_{t,i}(1 - \rho) + \rho; \\
\quad &\quad \text{else } p_{t+1,i} \leftarrow p_{t,i}(1 - \rho); \\
\quad &\quad \text{Restrict } p_{t+1,i} \text{ to be within } [1/n, 1-1/n]; \\
\quad &\quad t \leftarrow t + 1; \\
\end{align*}
\]

Here \( 1/\rho \) reflects preciseness.

UMDA (Mühlenbein and Paass, 1996)

\[
\begin{align*}
t &\leftarrow 0, \ p_{t,1} \leftarrow p_{t,2} \leftarrow \cdots \leftarrow p_{t,n} \leftarrow \frac{1}{\lambda}; \\
\text{while termination criterion not met do} \\
\quad &P_t \leftarrow \emptyset; \\
\quad &\text{for } j \in \{1, \ldots, \lambda\} \text{ do} \\
\quad &\quad \text{for } i \in \{1, \ldots, n\} \text{ do} \\
\quad &\quad &x_{i,j}^{(0)} \leftarrow 1 \text{ with prob. } p_{t,i}, \\
\quad &\quad &x_{i,j}^{(0)} \leftarrow 0 \text{ with prob. } 1 - p_{t,i}; \\
\quad &\quad \mu \left\{ \sum_{j=1}^{\lambda} x_{i,j}^{(0)} \right\} \\
\quad &P_t \leftarrow P_t \cup \{x_{i,j}^{(0)}\}; \\
\quad &\text{Sort individuals in } P \text{ descending by fitness (s. t. } f(x_i^{(1)}) \geq \cdots \geq f(x_i^{(\mu)})), \text{ breaking ties u. a. r.}; \\
\quad &\text{for } i \in \{1, \ldots, n\} \text{ do} \\
\quad &\quad p_{t+1,i} \leftarrow \frac{\sum_{j=1}^{\lambda} x_{i,j}^{(0)}}{\lambda}; \\
\quad &\quad \text{Restrict } p_{t+1,i} \text{ to be within } [1/n, 1-1/n]; \\
\quad &\quad t \leftarrow t + 1; \\
\end{align*}
\]

If, e.g., \( \mu = \lambda/2 \), then \( \lambda \) reflects preciseness of model.

Adjustment of frequencies: a general challenge

Frequencies may walk into the wrong direction

- if the fitness function gives wrong hints w. r. t. single bits,
- if the function gives no hints but algorithm must update,
- even if the fitness function overall gives the right hints.

Think of OneMax:

\[
\begin{align*}
x &\quad 1 & 1 & 1 & 0 & 0 & 1 & 1 & * \\
\uparrow \quad \uparrow & \quad \uparrow & \quad \uparrow & \quad \uparrow & \quad \uparrow & \quad \uparrow & \quad p_8 & \quad \circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\downarrow \\
y &\quad 1 & 0 & 0 & 0 & 1 & 1 & 0 & * \\
\end{align*}
\]

\( p_8 \) can go up, down or stay the same.
**Genetic Drift**

If the fitness function is constant/flat (does not give a signal), frequencies move randomly to a border (DEMO).

\[ 1 - \frac{1}{n} \]

\[ \frac{1}{n} \]

Frequencies that move to the wrong border are problematic – even disastrous if the border is not there.

Even if the expected value of a frequency converges to optimal value (Höhfeld and Rudolph, 1997), this does not say much about runtime.

---

**Contents**

- Introduction
- Preliminaries
- OneMax
- LeadingOnes
- BinaryValue
- Noise
- Jump
- Stable EDAs
- End

---

**Early Results (< 2000)**

Models of cGA, UMDA and others, allowing estimations of the dynamical behavior (e.g., Thierens et al., 1998; Mühlenbein and Mahnig, 1999).

Two effects:

1. Overall progress of probabilistic model (roughly: $\sum p_i$) and time to convergence to good distribution ($\sum p_i \approx n$).
2. time for single frequencies to drift to wrong border by genetic drift $\rightarrow$ should be bigger than convergence time.

Avoiding genetic drift requires precise enough model $\rightarrow$ lower bound on runtime. Models of EDAs estimated progress of frequencies:

Mühlenbein and Mahnig (1999)

\[ p_{t+1,i} \approx p_{t,i} + \frac{1}{n} \sqrt{n} p_{t,i} (1 - p_{t,i}) \]

which was made rigorous recently.
First Steps Towards Runtime Analyses

Genetic drift recurrent issue in analysis of GAs and EDAs, e.g., Asoh and Mühlenbein (1994); Shapiro (2003, 2005). Models proposed to make frequencies “stable” (Friedrich et al., 2016a) → later.

**Lower bound on preciseness of model**

For different EDAs on **OneMax**, same threshold identified multiple times (Thierens et al., 1998; Lobo et al., 2000; Shapiro, 2005): need at least \( K = \Omega(\sqrt{n}) \) different frequency values to prevent genetic drift.

**Upper bound on time to convergence**

Time for the EDA to converge to optimal solution \( \approx K \sqrt{n} \).

⇒ Best possible time complexity \( \Theta(n) \)?

Rigorous vs. Non-Rigorous Analyses

All (?) analyses of convergence speed of EDAs before 2005 made some simplifying assumptions.

**Pros**

- Unimportant details ignored
- Insights possible that rigorous analyses cannot achieve

**Cons**

- No estimation of errors: verification by experiments
- No theorem: may only hold for small problem sizes

Since 2005: rigorous runtime analyses of EDAs, following the same principle as runtime analysis of EAs.

First runtime analysis of EDAs

Droste (2006) considered cGA on **OneMax** (and other functions).

Studied runtime = no. iterations until optimum found.

**Main results**

- Upper bound \( O(K \sqrt{n}) \) on **OneMax**, for \( K = \Omega(n^{1/2+\varepsilon}) \)
  \( \rightarrow \) runtime \( O(n^{1+\varepsilon}) \) (with high probability)
- General lower bound \( \Omega(K \sqrt{n}) \).

**Recent refinement**: Upper bound \( O(n \log n) \) (Sudholt and W., 2016) for \( K = c \sqrt{n} \log n \), big constant \( c \)

⇒ competitive with simple EAs (e.g., (1+1) EA).

**Best lower bound until 2016**: \( \Omega(n) \) from general black-box complexity (Doerr and Lengler, 2015)

Demo and Landscape

Expected runtime of cGA depending on \( n \) and \( K \)

Graph showing \( \Omega(n) \) and \( \Theta(K \sqrt{n}) \) with \( \sqrt{n} \log n \) and \( K \).
**Proof Idea in the “Large $K$” Regime**

- Show limits on genetic drift: with high prob. all frequencies stay above $1/3$ in a phase of $\Theta(K\sqrt{n})$ steps.
- Frequencies move smoothly upwards (DEMO).
- To analyze speed: consider $\Phi_t = \sum_{i=1}^{n} p_{t,i}$ and analyze its drift. Important: how does a single frequency evolve?
- Consider the two offspring $x$ and $y$ and look into bit $i$.

**Dynamics on Bit $i$**

If $x_i = y_i$ then $p_{t,i}$ is unchanged.
If $x_i \neq y_i$ then $p_{t,i}$ is changed depending on $f(x)$ vs. $f(y)$:

$$f(x) < f(y) \quad \text{reinforce } y_i$$
$$f(x) > f(y) \quad \text{reinforce } x_i$$

**Red area**: bit $i$ is irrelevant in this step
⇒ genetic drift moves $p_{t,i}$ in a random direction $\pm 1/K$ (rw-step)

**Blue area**: bit $i$ decides the outcome of $f(x)$ vs. $f(y)$
⇒ increase $p_{t,i}$ (b-step, learning that 1s are better than 0s)

---

**Probability of a Biased Step at Bit $i$**

- Biased step occurs with probability at least $\Omega(1/\sqrt{n})$. If offspring differ in the bit (prob. $2p_i(1-p_i)$) then raised by $1/K$.
- Otherwise, frequency is **expected** to stay put.
- Altogether: $p_{t+1,i} = p_{t,i} + \Omega(p_{t,i}(1-p_{t,i})/\sqrt{n})$

Note similarity to $p_{t+1,i} \approx p_{t,i} + \frac{1}{\sqrt{n}} \sqrt{p_{t,i}(1-p_{t,i})}$ by Mühlenbein and Mahnig (1999)
What Happens for Small $K$

Now look into $K < \sqrt{n \log n}$:

- Lower bound $\Omega(n)$ until 2015.
- Improved to $\Omega(n \log n)$ (Sudholt and W., 2016).
- Heavy genetic drift occurs.

DEMO.

Whether optimum can be found at all, depends very much on the borders on the frequencies.

- If no borders, with high probability frequencies locked to 0 $\Rightarrow$ infinite runtime.
- If borders $\{1/n, 1 - 1/n\}$ used, optimum can still be found in polynomial time! No proof for cGA, but for UMDA (Lehre and Nguyen, 2017; W., 2017).

$\Rightarrow$ Borders may make the algorithm efficient despite genetic drift.

Idea for the Lower Bound

Coupon collector

You have to collect $n$ different coupons. In each round, you are given one coupon chosen uniformly at random with replacement. In expectation, it takes $\Omega(n \log n)$ rounds to collect all of them.

Here the coupons are the frequencies at the lower border. Each of them has probability $1/n$ of being raised.

If many frequencies move to the lower border before optimum is found, we cannot be faster than $n \log n$.

Medium and Large $K$: Overview

What happens if $K = o(\log n)$?

Known for 2-MMAS$_{ib}$ (Neumann et al., 2010), similar to cGA: landslides of frequencies occur.

DEMO.

Frequency that have attained their maximum $1 - 1/n$ are nevertheless likely to drop down to minimum.

Very unstable behavior, exponential optimization time.

Small $K$

What happens if $K = o(\log n)$?

Upper bound $O(K)$ only conjectured here.
### Runtime of cGA on OneMax: Complete Picture

![Graph showing runtime of cGA on OneMax]

- **runtime** exponential \( O(n \log n) \) and \( O(Kn) \)
- \( \Theta(K\sqrt{n}) \)

### Analysis of UMDA

How does UMDA perform on OneMax?

Surprisingly, in terms of runtimes (no. of iterations \( \cdot \lambda \)), not very differently from cGA.

Obstacles in analysis:
- frequencies can change drastically, even from minimum to maximum in one generation,
- two parameters: \( \mu \) and \( \lambda \).

The first runtime analysis of UMDA on OneMax stems from 2015!

### Results for UMDA: Upper bounds

First runtime result (Dang and Lehre, 2015): expected runtime of UMDA on OneMax is \( O(n \lambda \log \lambda) \) for \( \lambda > 13e\mu \) and \( \lambda = \Omega(\log n) \). Bound is \( O(n \log n \log \log n) \) for best possible parameter setting.

Bound independently improved to \( O(n \log n) \) by Lehre and Nguyen (2017) and W. (2017):

**Theorem (Expected runtime of UMDA on OneMax)**

1. For constant \( a > 0 \) and constant \( c \in (0, 1) \), assume \( a \ln n \leq \mu \leq \sqrt{n(1-c)} \), \( \lambda \geq (13e)\mu/(1-c) \Rightarrow \) runtime \( O(\lambda n) \).
2. Assume \( \lambda = (1 + \beta)\mu \) for constant \( \beta > 0 \), \( \mu \geq c \log n \) for large constant \( c > 0 \), as well as \( \mu = o(n) \Rightarrow \) runtime \( O(\lambda n) \).
3. Assume \( \lambda = (1 + \beta)\mu \) for constant \( \beta > 0 \), \( \mu \geq c\sqrt{n} \log n \) for large constant \( c > 0 \Rightarrow \) runtime \( O(\lambda \sqrt{n}) \).

### Results for UMDA: Lower Bounds

We also obtain similar bounds as with cGA:

**Theorem (Krejca and W., 2017)**

Let \( \lambda = (1 + \beta)\mu \) for some constant \( \beta > 0 \) and \( \lambda = n^{O(1)} \). Then the expected optimization time of UMDA on OneMax is \( \Omega(\lambda \sqrt{n} + n \log n) \).

**Proof idea again:** at a very high level, we estimate

1. progress (drift) of sum of frequencies per iteration
2. time for genetic drift to move frequencies to wrong border both for upper and lower bounds. Especially 2. is challenging.
Overview of Runtime Bound for UMDA on OneMax

If $\lambda = \omega(\log n)$ and $\lambda = o(\sqrt{n})$
$\Rightarrow$ expected runtime $O(\lambda n)$ and $\Omega(n \log n)$.
Take, e.g., $\lambda = n^{1/3} \Rightarrow$ upper bound $O(n^{1/3})$, lower bound $O(n \log n)$.
Again upper bound $O(n \log n)$ for $\lambda = c\sqrt{n \log n}$. Where is the truth?

Experiments:

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Average Runtime</th>
<th>Frequency Hits Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$50$</td>
<td>$3.2 \times 10^4$</td>
<td>$0.5$</td>
</tr>
<tr>
<td>$100$</td>
<td>$3.4 \times 10^4$</td>
<td>$1.5$</td>
</tr>
<tr>
<td>$150$</td>
<td>$3.6 \times 10^4$</td>
<td>$2.0$</td>
</tr>
<tr>
<td>$200$</td>
<td>$3.8 \times 10^4$</td>
<td>$2.5$</td>
</tr>
<tr>
<td>$250$</td>
<td>$4.0 \times 10^4$</td>
<td>$3.0$</td>
</tr>
<tr>
<td>$300$</td>
<td>$4.2 \times 10^4$</td>
<td>$3.5$</td>
</tr>
<tr>
<td>$350$</td>
<td>$4.4 \times 10^4$</td>
<td>$4.0$</td>
</tr>
</tbody>
</table>

Left: average runtime; right: no. times frequency hits minimum.
In fact a multimodal behavior: is $O(\lambda n)$ also $\Omega(\lambda n)$ in medium regime?

New Result: Runtime of cGA is Multimodal

The multimodal behavior has been made rigorous for the simpler cGA, not yet for UMDA.

Theorem (Lengler et al., 2018)

Expected runtime of cGA on OneMax is $\Omega(K^{1/3} n + n \log n)$ for $K = O(\sqrt{n} / \log^2 n)$.

$\Rightarrow$ Setting $K = \Theta(\log n)$ gives us runtime $\Theta(n \log n)$, so does $K = \Theta(\sqrt{n} \log n)$, but values in between make runtime worse.

OneMax: Summary

- Simple univariate EDAs like UMDA and cGA (and 2-MMAS$_{ib}$) have similar runtime behavior on OneMax.
- Very sensitive to settings of parameters.
- Two phase transitions.
- Multimodal behavior.
- If no borders are used, model must have model of preciseness $\Omega(\sqrt{n} \log n)$ to prevent genetic drift.
EDAs on LeadingOnes

LeadingOnes\( (x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \) has not been considered much in the theory of EDAs (except for UMDA).

Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.

Typical: frequencies are optimized from left to right.
(DEMO)

If best-so-far solution has \( i \) leading ones, then last \( n - i - 1 \) frequencies are drifting randomly.
EDAs on LeadingOnes

LeadingOnes \((x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_{j}\) has not been considered much in the theory of EDAs (except for UMDA).

Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.

Typical: frequencies are optimized from left to right.

(DEMO)

If best-so-far solution has \(i\) leading ones, then last \(n - i - 1\) frequencies are drifting randomly.

---

EDAs on LeadingOnes

LeadingOnes \((x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_{j}\) has not been considered much in the theory of EDAs (except for UMDA).

Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.

Typical: frequencies are optimized from left to right.

(DEMO)

If best-so-far solution has \(i\) leading ones, then last \(n - i - 1\) frequencies are drifting randomly.
EDAs on LeadingOnes

\[
\text{LeadingOnes}(x) = \sum_{i=1}^n \prod_{j=1}^i x_j
\]

has not been considered much in the theory of EDAs (except for UMDA).

Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.

Typical: frequencies are optimized from left to right.

(DEMO)

If best-so-far solution has \(i\) leading ones, then last \(n - i - 1\) frequencies are drifting randomly.
EDAs on LeadingOnes

LeadingOnes\( (x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \) has not been considered much in the theory of EDAs (except for UMDA).
Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.
Typical: frequencies are optimized from left to right.
(DEMO)

If best-so-far solution has \( i \) leading ones, then last \( n - i - 1 \) frequencies are drifting randomly.
EDAs on LeadingOnes

LeadingOnes \(x = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j\) has not been considered much in the theory of EDAs (except for UMDA).

Possible reason: behavior is more obvious than on OneMax and not very different from classical EAs.

Typical: frequencies are optimized from left to right.

(DEMO)

If best-so-far solution has \(i\) leading ones, then last \(n - i - 1\) frequencies are drifting randomly.

EDAs on LeadingOnes

Early Results for LeadingOnes

UMDA without borders (Chen et al., 2007, 2009b, 2010):

Using \(\lambda = \Omega(n^{2+\varepsilon})\), UMDA (without borders) optimizes LeadingOnes in time \(O(\lambda n)\) w.h.p. \(\Rightarrow\) runtime \(O(n^{2+\varepsilon})\) for optimal \(\lambda\).

For comparison: (1+1) EA expected runtime \(\Theta(n^2)\).

Approach in the analysis again:
1. Determine total time for frequency vector to converge to optimality.
2. Determine lower bound on preciseness of model to prevent genetic drift before convergence.

Large \(\lambda\) used to prevent genetic drift also for the last optimized bit (possibly too large).
Recent results for LeadingOnes

Use the borders.

**Theorem (Dang and Lehre, 2015)**

If $\lambda \geq c \ln n$ then expected runtime of UMDA with borders on LeadingOnes is $O(n \lambda \log \lambda + n^2)$.

Holds also for PBIL (generalized UMDA) if learning rate not too small (Lehre and Nguyen, 2018).

When EAs and EDAs Differ on LeadingOnes

Overall analysis was similar to OneMax. No new insights through the study of LeadingOnes?

Consider following example from Chen et al. (2009a).

$$\text{Substring}(x) = \begin{cases} 2n & \text{if } x = (1, \ldots, 1) \\ \max_{i=1}^{n} i \cdot \prod_{j=\max\{i-n/4,1\}}^i x_j & \text{otherwise} \end{cases}$$

Substring equals LeadingOnes if no block of at least $n/4$ consecutive ones. E.g.: $11110 \ast \ast \ast \ast \ast \ast \ast \ast \ast \ast \ast \ast \ast \ast \rightarrow 4$.

Otherwise, it describes the starting position of the rightmost block of $n/4$ ones – except if everything is one. E.g.: $010101011 \ldots 111 \rightarrow 3n/4$.

EDAs Beat EAs on Substring

**Theorem (simplified from Chen et al., 2009a)**

$(1+1)$ EA needs with overwhelming probability time $2^cn$ to optimize Substring. UMDA with $\lambda = \Omega(n^{2+\epsilon})$, $\mu = \lambda/2$ optimizes Substring in polynomial time w. o. p.

**Proof idea for $(1+1)$ EA**

Typically, $(1+1)$ EA starts out by gaining more and more leading ones. $10\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast\ast$.

When $> n/4$ leading ones, the first bit(s) no longer contributes to fitness. These bits slowly become random again. Optimum missed, NIAH.
**EDAs Beat EAs on Substring**

**Theorem (simplified from Chen et al., 2009a)**

(1+1) EA needs with overwhelming probability time \(2^{cn}\) to optimize \(\text{Substring}\). UMDA with \(\lambda = \Omega(n^{2+\varepsilon})\), \(\mu = \lambda/2\) optimizes \(\text{Substring}\) in polynomial time w. o. p.

**Proof idea for (1+1) EA**

Typically, (1+1) EA starts out by gaining more and more leading ones.

110*************************

When \(> n/4\) leading ones, the first bit(s) no longer contributes to fitness. These bits slowly become random again. **Optimum missed, NIAH**
EDAs Beat EAs on Substring

Theorem (simplified from Chen et al., 2009a)

(1+1) EA needs with overwhelming probability time $2^{cn}$ to optimize Substring. UMDA with $\lambda = \Omega(n^2 + \epsilon)$, $\mu = \lambda/2$ optimizes Substring in polynomial time w. o. p.

Proof idea for (1+1) EA

Typically, (1+1) EA starts out by gaining more and more leading ones.

111111111111111110**********

When $> n/4$ leading ones, the first bit(s) no longer contributes to fitness. These bits slowly become random again. Optimum missed, NIAH

EDAs Beat EAs on Substring

Theorem (simplified from Chen et al., 2009a)

(1+1) EA needs with overwhelming probability time $2^{cn}$ to optimize Substring. UMDA with $\lambda = \Omega(n^2 + \epsilon)$, $\mu = \lambda/2$ optimizes Substring in polynomial time w. o. p.

Proof idea for (1+1) EA

Typically, (1+1) EA starts out by gaining more and more leading ones.

110111111111111110**********

When $> n/4$ leading ones, the first bit(s) no longer contributes to fitness. These bits slowly become random again. Optimum missed, NIAH
EDAs Beat EAs on Substring

Theorem (simplified from Chen et al., 2009a)

\((1+1)\) EA needs with overwhelming probability time \(2^{cn}\) to optimize Substring. UMDA with \(\lambda = \Omega(n^2 + \varepsilon)\), \(\mu = \lambda/2\) optimizes Substring in polynomial time w. o. p.

Proof idea for \((1+1)\) EA

Typically, \((1+1)\) EA starts out by gaining more and more leading ones.

\(1100101111111111111110******\)

When \(> \frac{n}{4}\) leading ones, the first bit(s) no longer contributes to fitness. These bits slowly become random again. Optimum missed, NIAH

Proof idea for UMDA

Also UMDA starts out by gaining more and more leading ones. This is reflected in the frequencies.

\[1 - \frac{1}{n}\]

\[\frac{1}{n}\]

\(101011110101011101010\)

Even after the first bits no longer contribute to fitness, their frequencies are expected to remain the same. All-ones string can be sampled.

Need large enough \(\lambda\) to prevent genetic drift.

Proof idea for UMDA

Also UMDA starts out by gaining more and more leading ones. This is reflected in the frequencies.

\[1 - \frac{1}{n}\]

\[\frac{1}{n}\]

\(11110**************\)

Even after the first bits no longer contribute to fitness, their frequencies are expected to remain the same. All-ones string can be sampled.

Need large enough \(\lambda\) to prevent genetic drift.
Proof idea for UMDA

Also UMDA starts out by gaining more and more leading ones. This is reflected in the frequencies.

\[
1 - \frac{1}{n}
\]

\[
\frac{1}{n}
\]

Even after the first bits no longer contribute to fitness, their frequencies are expected to remain the same. All-ones string can be sampled.

Need large enough \( \lambda \) to prevent genetic drift.

LeadingOnes: Summary

Typical behavior of EDAs:
- On OneMax they optimize all bits roughly at the same time.
- On LeadingOnes they optimize bits from left to the right.
- Runtime \( \Theta(n \log n) \) vs. \( \Theta(n^2) \) for opt. parameters settings.
- 

However, there are other EDAs that do not behave like this (see later).
The final example function: BinVal

Recall

\[ \text{BinVal}(x_1, \ldots, x_n) := \sum_{i=1}^{n} 2^{n-i} x_i, \]

being somewhere between LeadingOnes and OneMax. A bit outweighs all less significant bits together, but every bit contributes to fitness.

Often, a runtime analysis of LeadingOnes also gives a runtime bound for BinVal (e.g., for UMDA/PBIL, Lehre and Nguyen, 2018).

**Theorem (Droste, 2006)**

The runtime of cGA on BinVal is \( O(Kn) \) for \( K = \Omega(n^{1+\varepsilon}) \) w.o.p. It is \( \Omega(Kn) \) w.o.p.

Note: upper bound \( O(n^{2+\varepsilon}) \). Lower bound does not restrict \( K \).

Not all Linear Functions Are Equally Difficult for EDAs

Known for (1+1) EA (W., 2013): all linear functions optimized in expected time \((1 \pm o(1))en\ln n\).

\[ \Rightarrow \text{Runtimes on OneMax and BinVal differ by a lower-order term.} \]

Droste conjectured that BinVal cannot be optimized in time \( O(n\log n) \).

**Theorem (W., 2018)**

The runtime of cGA (without borders) on BinVal is \( \Omega(n^2) \) with prob. \( \Omega(1) \). Choosing \( K = o(n) \) leads to infinite runtime w.h.p.

Idea: if \( K \) too small, genetic drift likely to occur at light bits before all heavy bits optimized.

Optimization under Uncertainty

Assume that an evaluation of the objective function is subject to random noise.

**Example: OneMax with additive Gaussian noise**

\[ f_{\text{noise}}(x) = \text{OneMax}(x) + N(0, \sigma). \]

Typical measures to handle noise in evolutionary computation:

- large populations
- resampling
- ...

EDAs have a built-in noise handling mechanism.
EDAs Beat Population-based EAs in Noisy Settings

Definition (Friedrich et al., 2017): an algorithm *scales gracefully* if its expected runtime depends polynomially on the noise strength. For example: runtime is proportionally to $\sigma^2$.

Theorem

$(\mu+1)$ EA does not scale gracefully on OneMax with additive Gaussian noise. However, cGA does.

Proof Ideas

$\mu+1$ EA uses mutation, which drifts away from the optimum when there are many correct bits and noise disturbs fitness signal.

$x = 11111111011111111111011111$

$y = 11110111011111111111011111$

$\Pr(f(x) > f(y)) \approx \Pr(f(x) < f(y))$

$cGA$ does not use mutation and is *balanced*: in expectation, frequency vector and thus fitness does not decrease over time.

$cGA$ does not receive negative signal from noise, but signal (drift) towards increasing OneMax becomes smaller with increasing $\sigma$.

Have to ensure that effect of genetic drift is smaller than the signal $\rightarrow$ choose big enough $K$. 

Further Noisy Settings

Also simple ACO algorithms can be considered EDAs and are superior to EAs in some noisy settings (also from combinatorial optimization):

- Sudholt and Thyssen (2012) for ACO for noisy shortest paths with a ground truth
- Doerr et al. (2012) where the noise is intrinsic
- Feldmann and Kötzing (2013) for ACO with fitness-proportional updates: leads to convergence to expected best solution

Many of the results can probably be transferred to more classical EDAs such as cGA and UMDA.
How EDAs Overcome Gaps

Jump challenging for hillclimbers: \( \Theta(n^k) \) to cross gap of size \( k > 1 \).

EDAs can search more globally in Jump region thanks to higher sampling variance \( \sim \) time \( \Theta(c^k) \) instead of \( \Theta(n^k) \) (first observed in Hasenöhrl and Sutton, 2018).

Theorem (Doerr, 2019)

If \( k \leq \frac{1}{20} \ln n \) and \( \mu = \Omega(\sqrt{n} \log n) \) then cGA optimizes Jump in \( O(\mu \sqrt{n}) \) iterations w. h. p. Time \( O(n \log n) \) if \( \mu = \Theta(\sqrt{n} \log n) \).

See talk at this GECCO.

Balancedness, Stability and Genetic Drift

Have seen: without fitness signal, stochastic model of EDAs is expected to be the same. Term: balanced (Friedrich et al., 2016a).

However, this is only the expected value. Genetic drift plays major role in classical EDAs.

Term: EDA is stable if a frequency in absence of fitness signal stays close to its initial value.

cGA, UMDA, ... are not stable. Frequencies quickly converge to either maximum or minimum (each with probability 1/2) due to genetic drift.

A New Way to Overcome Genetic Drift: Significance-Based EDAs

Idea (Doerr and Krejca, 2018): Move frequency away from its initial value only when there is evidence that 0 or 1 is the better bit value.

sig-cGA: algorithmic ideas

- Framework like cGA.
- For each bit, history \( H_t \in \{0,1\}^* \) of values in better individual
- Investigate last \( m \) history bits. If a value significantly dominates, move frequency to corresponding border (1/n if 0 dominates, 1 − 1/n if 1).
- Otherwise, leave frequency at 1/2.
- Example of significance: \( \|H_t\|_1 \geq C \sqrt{m \ln n} \)
- Different values for \( m \) are tested by the algorithm.
Significance-Based EDAs are Fast

Theorem

The expected runtime of sig-cGA on both OneMax and LeadingOnes is $O(n \log n)$ (and with high probability).

No other evolutionary algorithm is known that simultaneously optimizes OneMax and LeadingOnes in time $O(n \log n)$.

Proof ideas

- On OneMax, drift $\Omega(p_i(1 - p_i)/\sqrt{n})$ quickly identified as significant. Many ideas of analysis of plain cGA work.
- On LeadingOnes, frequencies are optimized from left to right. Bits that do not contribute to fitness yet: no signal, no significance of deviation, no genetic drift, stay at $1/2$

Significance-based EDAs are promising, theory-driven approach.

Summary and Conclusions

Summary

- Runtime analysis for simple univariate EDAs
- Identified similarities to and differences from simple EAs
- Genetic drift a major obstacle
- Sensitive to parameters (phase transitions)
- Robust to noise
- Significance-based EDAs as novel theory-driven approach

Future work

- Combinatorial problems
- Multivariate EDAs
- Classification of problems w.r.t. appropriateness for EAs/EDAs, ...

References I


References II


References III


References IV


References V


References VI


References VII


References VIII


References IX


References X


