ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

Fast Genetic Algorithms

Benjamin Doerr (École Polytechnique, France)

joint work with Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen

In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO), pages 777-784, ACM, 2017.

Supported by ANR-11-LABEX-0045-DIGICOSME and ANR-11-LABEX-0056-LMH

Topic

- This talk is about how to do mutation in evolutionary computation for discrete search spaces
  - We only regard the search space $\Omega = \{0,1\}^n$, but our findings hold in a similar manner for other representations

- The best EA to study mutation in isolation is the (1+1) EA. Most of our results will hold in a similar way for other mutation-based algorithms.

(1+1) evolutionary algorithm to maximize $f: \{0,1\}^n \to \mathbb{R}$:
1. choose $x \in \{0,1\}^n$ uniformly at random
2. while not $terminate$ do
3. $\quad y := mutate(x)$
4. $\quad$ if $f(y) \geq f(x)$ then $x := y$
5. output x

- Runtime: Number of iterations until the optimum is generated (assuming no termination criterion)

General Belief on Mutation

- A good way of doing mutation is *standard-bit mutation*, that is, flipping each bit independently with some probability p ("mutation rate")

standard-bit mutation($x \in \{0,1\}^n$, $p \in [0,1]$):

1. $y := x$
2. for $i = 1$ to n do
3. if $random([0,1]) \leq p$ then $y_i := 1 - y_i$
4. return y

- Note: from any parent you can generate any offspring
  → algorithms cannot get stuck forever in a local optimum ("convergence")

- **General recommendation:** Use a small mutation rate like $1/n$

Informal Justifications for $1/n$

- Maximal chance for a 1-bit flip: If you want to flip a particular single bit, then
  - a mutation rate of $1/n$ maximizes this probability
  - a rate of c/n, $c < 1$, reduces this probability by a factor of $\Theta(c)$
  - a rate of c/n, $c > 1$, reduces this probability by a factor of $e^{-\Theta(c)}$

- Mutation is destructive: If your current search point x has a Hamming distance $H(x, x^*)$ of at most $n/2$ from the optimum x^*, then the offspring y has (in expectation) a larger Hamming distance and this increase is proportional to p:
  - $E[H(y, x^*)] = H(x, x^*) + p(n - 2H(x, x^*))$

$O(c)$ = at most γc for some constant γ
$\Omega(c)$ = at least δc for some constant $\delta > 0$
$\Theta(c)$ = both $O(c)$ and $\Omega(c)$

Proven Results Supporting a $1/n$ Mut. Rate

- Optimal mutation rates for (1+1) EA:
  - $1/n$ for OneMax [Mühlenbein '92; Bäck '93]: $f(x) = x_1 + \cdots + x_n$
  - $1/n$ for all linear functions [Witt '13]: $f(x) = w_1 x_1 + \cdots + w_n x_n$
  - $1.59/n$ for LeadingOnes [Böttcher, D., Neumann '10]: $f(x) = \max\{j \in [0..n] \mid \forall i \in [1..j]: x_i = 1\}$

- For monotonically increasing functions $f: \{0,1\}^n \to \mathbb{R}$, the (1+1) EA has the following runtimes for different mutation rates p [Jansen '07; D., Jansen, Sudhold, Winzen, Zarges '13; Lengler, Steger '17]:
  - $p = c/n,\ 0 < c < 1$, gives an $O(n \log n)$ runtime on all monotonic functions,
  - $p = 1/n$ gives $O(n^{1.5})$ for all monotonic functions,
  - $p \geq 2/n$ gives an exponential runtime on some monotonic functions.

- The optimal mutation rate for the (1+λ) EA with $\lambda \leq \ln n$ is $1/n$ for OneMax [Giessen, Witt'15].

Proven Results Supporting a $1/n$ Mut. Rate

- Optimal mutation rates for (1+1) EA:
  - $1/n$ for OneMax [Mühlenbein '92; Bäck '93]: $f(x)$
  - $1/n$ for all linear functions [Witt '13]: $f(x) = w_1 x_1 +$
  - $1.59/n$ for LeadingOnes [Böttcher, D., Neumann '10]: $f(x) = \max\{j \in [0..n] | \forall i \in [1..j]: x_i = 1\}$

all this supports using standard-bit mutation with mutation rate around $1/n$

- For monotonically increasing functions $f: \{0,1\}^n \to \mathbb{R}$, the (1+1) EA has the following runtimes for different mutation rates p [Jansen '07; D., Jansen, Sudhold, Winzen, Zarges '13; Lengler, Steger '17]:
  - $p = c/n,\ 0 < c < 1$, gives an $O(n \log n)$ runtime on all monotonic functions,
  - $p = 1/n$ gives $O(n^{1.5})$ for all monotonic functions,
  - $p \geq 2/n$ gives an exponential runtime on some monotonic functions.

- The optimal mutation rate for the (1+λ) EA with $\lambda \leq \ln n$ is $1/n$ for OneMax [Giessen, Witt'15].

Really?

- Can we really say that $1/n$ is good (at least “usually”)?
- More provocative: Can we really say that *standard-bit mutation* the right way of doing mutation?

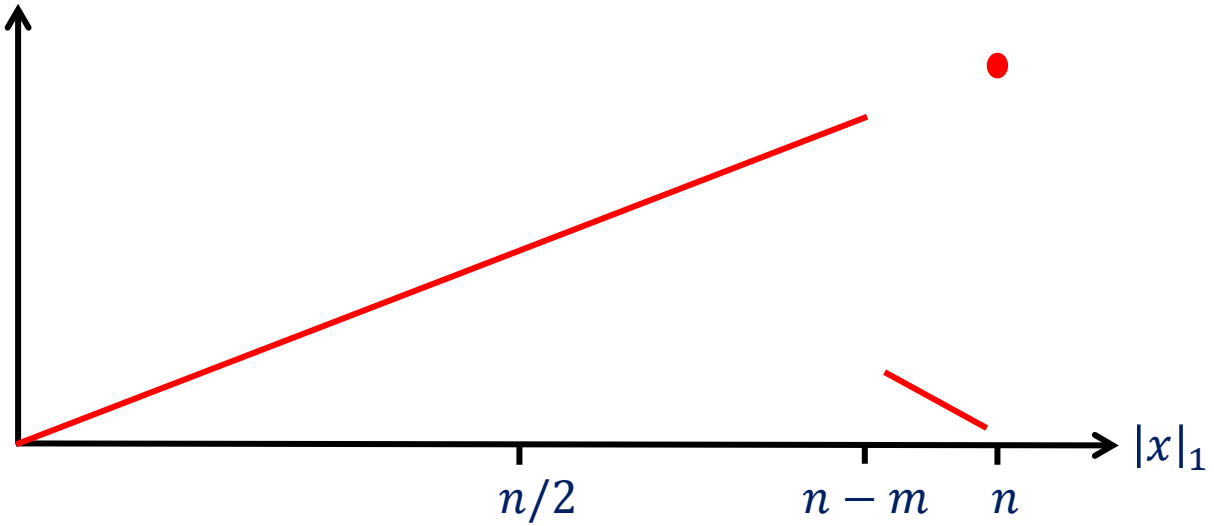
- What made us skeptical: All results supporting standard-bit mutation with rate $1/n$ regard easy unimodal optimization problems (where flipping single bits is a very good way of making progress)
  - OneMax, LeadingOnes, linear functions, monotonic functions

- Plan for this talk:
  - precise runtime analysis for $JUMP_{m,n}$ functions (not unimodal!)
  - observe something very different
  - design a new mutation operator to cope with this
  - show that it is pretty good for many problems

Main Object of This Study: Jump Functions

Jump functions [Droste, Jansen, Wegener '02]:

- $JUMP_{m,n}$: fitness of an n-bit string x is the number $|x|_1$ of ones, except if $|x|_1 \in \{n-m+1, \dots, n-1\}$, then the fitness is the number of zeroes.



- Observations:
  - All x with $|x|_1 = n-m$ form an easy to reach *local optimum*.
  - From there, only flipping (the right) m bits gives an improvement.
  - The unique *global optimum* is $x^* = (1 \dots 1)$.

Runtime Analysis

- Theorem: Let $T_p(m,n)$ denote the expected optimization time of the (1+1) EA optimizing $JUMP_{m,n}$ with mutation rate $p \leq 1/2$. For $2 \leq m \leq n/2$,

$$T_p(m,n) = \Theta(p^{-m}(1-p)^{n-m}).$$

- Corollary (speed-up at least exponential in m): For any $p \in [2/n\,, m/n]$,
$T_p(m,n) \leq 6e^2\, 2^{-m}\, T_{1/n}(m,n)$.

- **→ Clearly, here $1/n$ is not a very good mutation rate!**

- Proof of theorem (not overly difficult):
  - upper bound: classic fitness level method
  - lower bound: argue that the runtime is essentially the time it takes to go from the local to the global optimum

Optimal Mutation Rates

- Theorem: Let $T_{opt}(m,n) := \inf\{T_p(m,n) \mid p \in [0,1/2]\}$.
  - $T_{opt}(m,n) = \Theta\left(T_{m/n}(m,n)\right)$.
  - If $p \geq (1+\varepsilon)(m/n)$ or $p \leq (1-\varepsilon)(m/n)$, then
$$T_p(m,n) \geq \frac{1}{6} \exp\left(\frac{m\,\varepsilon^2}{5}\right) T_{opt}(m,n).$$
- In simple words: m/n is essentially the optimal mutation rate, but a small deviation from it increases the runtime massively.

- Dilemma: The right mutation rate depends strongly on your problem ☹
- Reason: With standard-bit mutation, the Hamming distance $H(x,y)$ of parent and offspring is strongly concentrated around the mean

- **→ Maybe standard-bit mutation is not the right thing to do?**

Solution: Heavy-tailed Mutation

- Recap: We want…
  - no strong concentration of $H(x,y)$
  - large numbers of bits flip with reasonable probability (to leave local opt.)
  - 1-bit flips occur with constant probability (for easy parts of the search)

- Solution: *Heavy-tailed mutation* (with parameter $\beta > 1$, say $\beta = 1.5$)
  - choose $\alpha \in \{1, 2, \dots, n/2\}$ randomly with $\Pr[\alpha] \sim \alpha^{-\beta}$ [power-law distrib.]
  - perform standard-bit mutation with mutation rate α/n

- Some maths:
  - The probability to flip k bits is $\Theta\left(k^{-\beta}\right)$ instead of roughly $\frac{1}{e \cdot k!}$
  - $\Pr[H(x,y) = 1] = \Theta(1)$, e.g., ≈32% for $\beta = 1.5$ instead of ≈37%

Heavy-tailed Mutation: Results

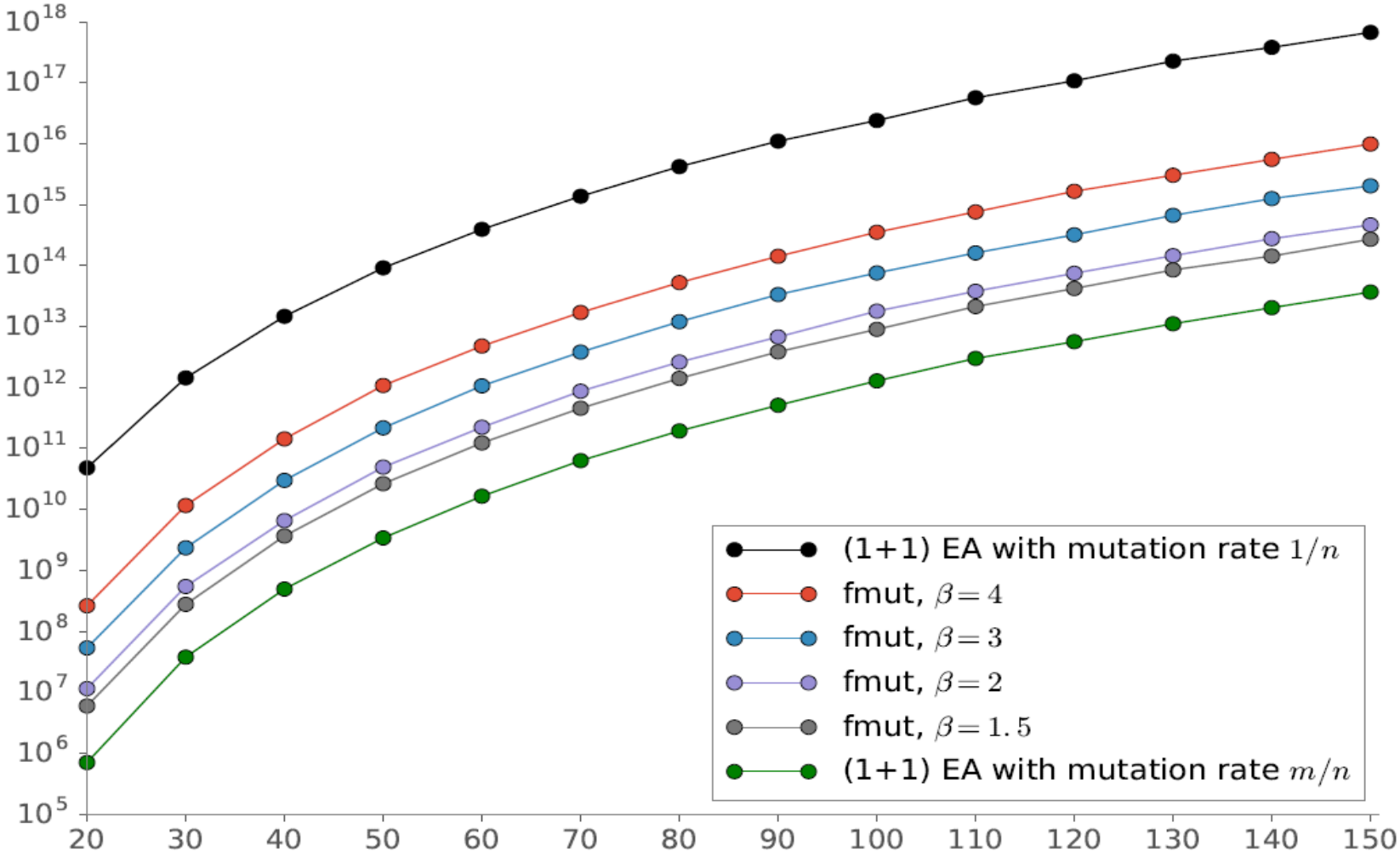
- Theorem: The (1+1) EA with heavy-tailed mutation ($\beta > 1$) has an expected optimization time on $JUMP_{m,n}$ of

$$O\big(m^{\beta-0.5}\, T_{opt}(m,n)\big).$$

- **This one algorithm for all m is only an $O(m^{\beta-0.5})$ factor slower than when using the (for this m) optimal mutation rate!**
- **Compared the rate $1/n$, this is a speed-up by a factor of $m^{\Theta(m)}$.**

- Lower bound (not so important, but mathematically beautiful): The small loss of order $\Theta(m^{0.5+\varepsilon})$ – by taking $\beta = 1 + \varepsilon$ – cannot be avoided:
  - For n sufficiently large, any distribution D_n on the mutation rates in $[0, 1/2]$ has an $m \in [2..n/2]$ such that $T_{D_n}(m,n) \geq \sqrt{m}\, T_{opt}(m,n)$.

Experiments (m=8, n=20..150)



Runtime of the (1+1) EA on $JUMP_{8,n}$ (average over 1000 runs). To allow this number of experiments, the runs where stopped once the local optimum was reached and the remaining runtime was sampled directly from the geometric distribution describing this waiting time.

Beyond Jump Functions

- 1st Example (maximum matching): Let G be an undirected graph having n edges. A matching is a set of non-intersecting edges. Let OPT be the size of a maximum matching. Let $m \in \mathbb{N}$ be constant and $\varepsilon = \frac{2}{2m+1}$.
  - The classic (1+1) EA finds a matching of size $\frac{OPT}{1+\varepsilon}$ in an expected number of at most $T_{n,\varepsilon}$ iterations, where $T_{n,\varepsilon}$ is some number in $\Theta(n^{2m+2})$. [GW03]
  - The (1+1) EA with heavy-tailed mutation does the same in expected time of at most $\boldsymbol{(1+o(1))\, e\, \zeta(\beta) \left(\frac{e}{m}\right)^m m^{\beta-0.5}\, T_{n,\varepsilon}}$.

Riemann zeta function:
$\zeta(\beta) < 2.62$ for $\beta \geq 1.5$

- 2nd example: Vertex cover in bipartite graphs (details omitted)

Performance in Classic Theory Results

- Since the heavy-tailed mutation operator flips any constant number of bits with constant probability, many classic theory results remain valid (apart from constant factor changes):
  - $O(n \log n)$ runtime on OneMax
  - $O(n^2)$ runtime on LeadingOnes
  - $O(m^2 \log(nw_{max}))$ runtime on MinimumSpanningTree [Neumann, Wegener '07]
  - and many others…

- The largest expected runtime that can occur on an $f: \{0,1\}^n \to \mathbb{R}$ is
  - $\Theta(n^n)$ for the classic (1+1) EA [Droste, Jansen, Wegener '02 (Trap); Witt '05 (minimum makespan scheduling)]
  - $O(n^\beta \, 2^n)$ for the heavy-tailed (1+1) EA

Working Principle of Heavy-Tailed Mutation

- Reduce the probability of a 1-bit flip slightly (say from 37% to 32%)
- Distribute the remaining probability mass in a power-law fashion on all other k-bit flips
  → increases the prob. for a k-bit flip from roughly $\frac{1}{e \cdot k!}$ to roughly $k^{-\beta}$
  → reduces the waiting time for a k-bit flip from $e \cdot k!$ to k^{β}

- This redistribution of probability mass is a good deal, because we usually spend much more time on finding a good multi-bit flip
  - $JUMP_{m,n}$: spend $\Theta(n \log n)$ time on all 1-bit flips, but at least $\binom{n}{m}$ time to find the one necessary m-bit flip

- **This simple working principle suggest that heavy-tailed mutation can be advantageous for a wide range of multi-modal problems.**

Nomenclature: Heavy-tailed → *Fast*

- Heavy-tailed mutation has been experimented with in *continuous optimization* (with mixed results as far as I understand)
  - simulated annealing [Szu, Hartley ‘87]
  - evolutionary programming [Yao, Lui, Lin ‘99]
  - evolution strategies [Yao, Lui ’97; Hansen, Gemperle, Auger, Koumoutsakos ’06; Schaul, Glasmachers, Schmidthuber ‘11]
  - estimation of distribution algorithms [Posik ’09, ‘10]

- Algorithms using heavy-tailed mutation were called *fast* by their inventors, e.g., *fast simulated annealing.*
  - → we propose to call our mutation *fast mutation* and the resulting algorithms *fast*, e.g., $(1+1)\ FEA_\beta$

How Does this Work for Real Problems?

- Oliveto, Sudholt (student project): (1+(λ,λ)) GA, P3, and $(1+1)$ FEA_β all do significantly better than many classic GAs

- Mironovich, Buzdalov (GECCO '17 student paper): Solid experiments for the problem of finding worst case instances for given algorithms ("scenario")
  - fast mutation significantly beats the two mutation-based approaches
  - fast mutation worsens the best-so-far crossover-based approach

Evol. Algorithm	Gain Scenario 1	Gain Scenario 2
(1+1) EA	+58%	+33%
GA w/o crossover	+22%	+9%
GA with crossover	-27%	-9%

- Much more experience needed: You can help us a lot by taking your favorite GA application and replacing your mutation with fast mutation!

Summary: Fast Mutation – Suggested by Theory, Successful in Practice?

- In a mathematical analysis of a simple mutation-based EA on the multi-unimodal JUMP fitness landscape, we observe that
  - higher mutation rates can greatly speed-up leaving local optima
  - standard-bit mutation with a fixed rate is sub-optimal on most instances

- Solution: *Fast mutation* ("parameter-less" heavy-tailed mutation). Standard-bit mutation with a rate sampled from a power-law distribution
  - $m^{\Theta(m)}$ factor speed-up for all $JUMP_{m,n}$ functions
  - $m^{\Theta(m)}$ factor improvement of the runtime guarantee for max. matching
  - promising preliminary experimental results

- Big question: Will this work in practice … and will people use it?

Thanks!