Blending Dynamic Programming with Monte Carlo Simulation for Bounding the Running Time of Evolutionary Algorithms

Kirill Antonov (ITMO) Maxim Buzdalov (ITMO)
Arina Buzdalova (ITMO) Carola Doerr (Sorbonne)

ITMO UNIVERSITY
SORBONNE UNIVERSITÉ

IEEE Congress on Evolutionary Computation
28.06 – 01.07.2021

Introduction

- Dynamic parameter settings can greatly improve the efficiency of evolutionary algorithms (EAs)
- Runtime lower bounds give a baseline, which is important for algorithm comparison and development
- Proving precise lower bounds for algorithms with dynamic parameter choices is challenging
- Previously, a dynamic programming approach was proposed to derive lower bounds for simple problems [Buzdalov, Doerr, PPSN 2020]
  - transition probabilities between different states can be expressed by mathematical expressions
  - applied to derive optimal mutation rates for OneMax problem
- We propose a method that combines dynamic programming with Monte Carlo sampling, which is applicable for a broader problem class

Considered Evolutionary Aglorithms

Data: n: problem size; $f : \{0,1\}^n \to \mathbb{R}$: function to maximize; λ: population size; $\mathcal{D}(p)$: a family of parameterized distributions over $[0..n]$

```
Sample parent x ∈ {0,1}^n uniformly at random;
for t ← 1, 2, ... do
    for i ∈ [1..λ] do
        Choose a distribution parameter p_i^t ;
        Sample k_i ~ D(p_i^t), the number of bits to flip;
        Create y_i by flipping k_i different bits in x chosen uniformly at random ;
    Select x ← arg max_{z ∈ {x, y_1, ..., y_λ}} f(z) breaking ties arbitrarily;
```

Considered Evolutionary Aglorithms

Data: n: problem size; $f : \{0,1\}^n \to \mathbb{R}$: function to maximize; λ: population size;
$\mathcal{D}(p)$: a family of parameterized distributions over $[0..n]$

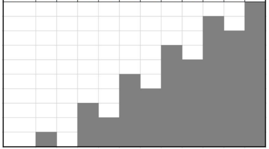
Sample parent $x \in \{0,1\}^n$ uniformly at random;
for $t \leftarrow 1, 2, \ldots$ **do**
  for $i \in [1..\lambda]$ **do**
    Choose a distribution parameter p_i^t ;
    Sample $k_i \sim \mathcal{D}(p_i^t)$, the number of bits to flip;
    Create y_i by flipping k_i different bits in x chosen uniformly at random ;
  Select $x \leftarrow \arg\max_{z \in \{x, y_1, \ldots, y_\lambda\}} f(z)$ breaking ties arbitrarily;

Parameter control in $(1+\lambda)$ EA with mutation rate p:

- 2-rate: try $p/2$ and $2p$ on two halves of population
- Ab rule: multiply p by A or b based on success
- HQEA: multiply p by A or b according to Q-learning

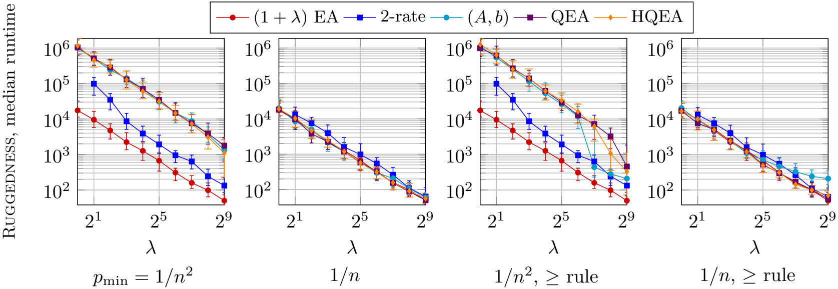
Ruggedness Problem and Benchmarking

i	0	1	2	3	4	5	6	7	8	9	10
$r_2(i)$	1	0	3	2	5	4	7	6	9	8	10



Optimum: $f(z) = n$.
Points at Hamming distance one from z have fitness $n - 2$,
those at distance two have fitness $n - 1$,
those at distance three have fitness $n - 4$,
those at distance four have fitness $n - 3$, and so on

Previous results for parameter control on Ruggedness:



Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;
Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;

Description of the Proposed Method

$f_{\text{min}}, f_{\text{max}} \leftarrow$ minimum and maximum fitness values;

Initialize optimal times: $T^*_{f_{\text{max}}} \leftarrow 0$;

for $f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}}$ **do**

Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;

Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;

for $f \leftarrow f_{\max} - 1, \ldots, f_{\min}$ **do**

  for $p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\}$ **do**

Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;
Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;
for $f \leftarrow f_{\max} - 1, \ldots, f_{\min}$ **do**
    for $p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\}$ **do**
        Compute approximate probabilities $(\tilde{p}_i)_{i=0,1,\ldots}$ of increasing fitness by i with mutation rate p using the Monte Carlo approach;

Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;
Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;
for $f \leftarrow f_{\max} - 1, \dots, f_{\min}$ **do**
    for $p \in \{p_1^{(f)}, p_2^{(f)}, \dots, p_{m_f}^{(f)}\}$ **do**
        Compute approximate probabilities $(\tilde{p}_i)_{i=0,1,\dots}$ of increasing fitness by i with mutation rate p using the Monte Carlo approach;
        $T_{f,p} \leftarrow \dfrac{1}{1 - \tilde{p}_0} \left(1 + \sum_{i>0} \tilde{p}_i \cdot T^*_{f+i}\right)$;

Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;

Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;

for $f \leftarrow f_{\max} - 1, \dots, f_{\min}$ **do**

  for $p \in \{p_1^{(f)}, p_2^{(f)}, \dots, p_{m_f}^{(f)}\}$ **do**

    Compute approximate probabilities $(\tilde{p}_i)_{i=0,1,\dots}$ of increasing fitness by i with mutation rate p using the Monte Carlo approach;

    $T_{f,p} \leftarrow \dfrac{1}{1-\tilde{p}_0}\left(1 + \sum_{i>0} \tilde{p}_i \cdot T^*_{f+i}\right)$;

  Store optimal time: $T^*_f \leftarrow \min_p(T_{f,p})$;

  Store optimal rate: $P^{\mathrm{opt}}_f \leftarrow \arg\min_p(T_{f,p})$;

Description of the Proposed Method

$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;

Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;

for $f \leftarrow f_{\max} - 1, \ldots, f_{\min}$ **do**

  for $p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\}$ **do**

    Compute approximate probabilities $(\tilde{p}_i)_{i=0,1,\ldots}$ of increasing fitness by i with mutation rate p using the Monte Carlo approach;

    $T_{f,p} \leftarrow \frac{1}{1-\tilde{p}_0}\left(1 + \sum_{i>0} \tilde{p}_i \cdot T^*_{f+i}\right)$;

  Store optimal time: $T^*_f \leftarrow \min_p(T_{f,p})$;

  Store optimal rate: $P^{\text{opt}}_f \leftarrow \arg\min_p(T_{f,p})$;

return $\{P^{\text{opt}}, T^*, T\}$

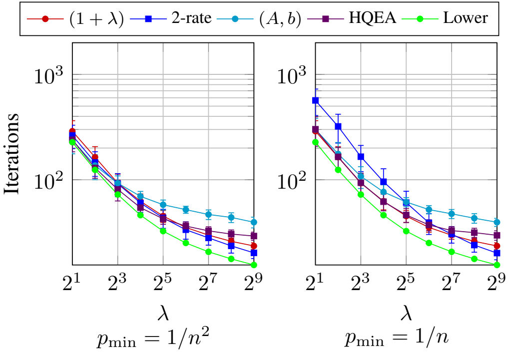
Description of the Proposed Method

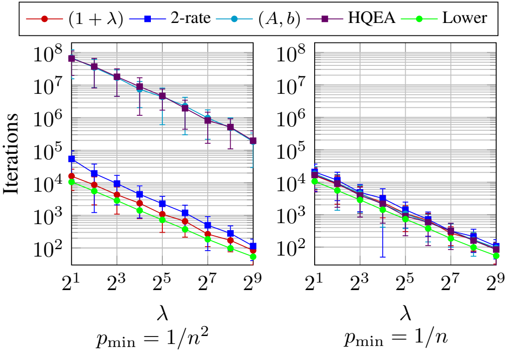
$f_{\min}, f_{\max} \leftarrow$ minimum and maximum fitness values;
Initialize optimal times: $T^*_{f_{\max}} \leftarrow 0$;
for $f \leftarrow f_{\max} - 1, \ldots, f_{\min}$ **do**
    for $p \in \{p^{(f)}_1, p^{(f)}_2, \ldots, p^{(f)}_{m_f}\}$ **do**
        Compute approximate probabilities $(\tilde{p}_i)_{i=0,1,\ldots}$ of increasing fitness by i with mutation rate p using the Monte Carlo approach;
        $T_{f,p} \leftarrow \frac{1}{1-\tilde{p}_0}\left(1 + \sum_{i>0} \tilde{p}_i \cdot T^*_{f+i}\right)$;
    Store optimal time: $T^*_f \leftarrow \min_p(T_{f,p})$;
    Store optimal rate: $P^{\text{opt}}_f \leftarrow \arg\min_p(T_{f,p})$;
return $\{P^{\text{opt}}, T^*, T\}$

Requirement: the optimal choice of p depends on the fitness value exclusively

Lower Runtime Bounds for Parameter Control

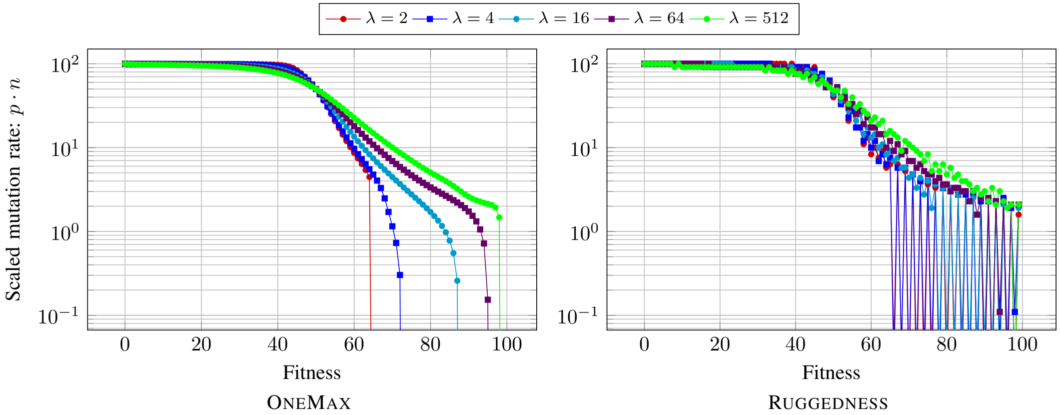
Iterations until the optimum of OneMax (left) and Ruggedness (right)





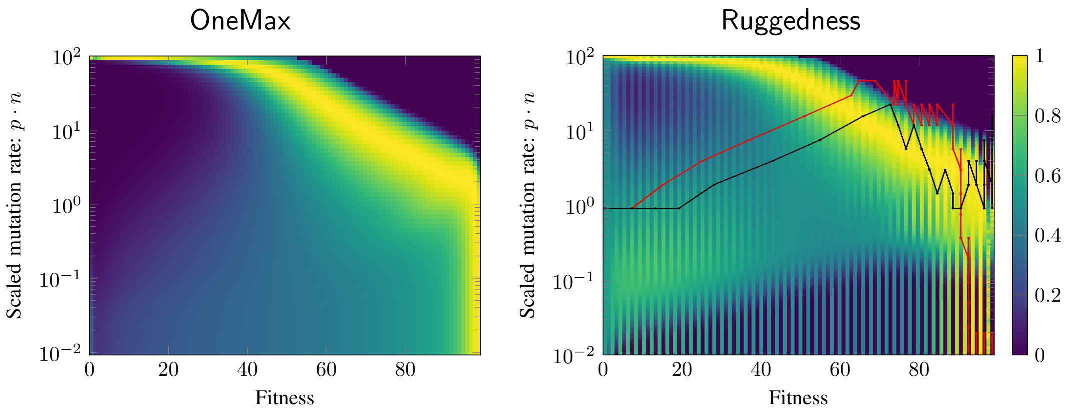
- New insight: on Ruggedness, only a constant-factor improvement is possible
- Why does (A,b) rule performs so much worse than 2-rate when using $p_{min} = 1/n^2$?

Optimal Mutation Rates



- Regular oscillations on Ruggedness with a period of 2
- It may be difficult to track precisely – is this a problem?

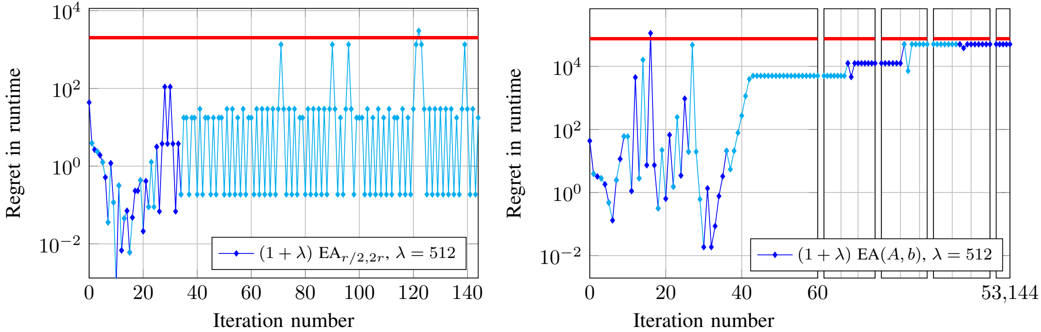
Parameter Efficiency Heatmaps



- Relative efficiency of the corr. p among all mutation rates for the corr. f
- The range of nearly equally good rates is wide enough
- On Ruggedness, for odd fitness values the best rates are higher
- (A, b) rule (red) gets stuck with too small rates near the optimum

Regret Plots

Regret $|T_{f,p} - T^*_f|$ for p chosen by 2-rate (left) and (A, b) rule (right)



- How much of the performance the method loses from acting suboptimally
- (A, b) rule spends most of its time with very large regrets

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds
- Introduced regret plots: not only show deficiencies in parameter control methods, but indicate their impact on the running time

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds
- Introduced regret plots: not only show deficiencies in parameter control methods, but indicate their impact on the running time
- Example application
  - Runtime estimations for the $(1 + \lambda)$ EA on the Ruggedness problem, $n = 100$
  - Analysis of (A, b) and 2-rate parameter control methods

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds
- Introduced regret plots: not only show deficiencies in parameter control methods, but indicate their impact on the running time
- Example application
  - Runtime estimations for the $(1 + \lambda)$ EA on the Ruggedness problem, $n = 100$
  - Analysis of (A, b) and 2-rate parameter control methods
- The method is restricted to settings in which states are not visited more than once
- Possible solution:
  - construction of Markov chains on all states with equal fitness
  - solving the resulting system of equations

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds
- Introduced regret plots: not only show deficiencies in parameter control methods, but indicate their impact on the running time
- Example application
  - Runtime estimations for the $(1 + \lambda)$ EA on the Ruggedness problem, $n = 100$
  - Analysis of (A, b) and 2-rate parameter control methods
- The method is restricted to settings in which states are not visited more than once
- Possible solution:
  - construction of Markov chains on all states with equal fitness
  - solving the resulting system of equations
- Not limited to $(1 + \lambda)$ type algorithms

Conclusion and Generalisation

- We proposed a dynamic programming approach with Monte Carlo simulations
  - Computes running times for different mutation rates at each stage of optimization
  - Useful for deriving optimal rates and runtime lower bounds
- Introduced regret plots: not only show deficiencies in parameter control methods, but indicate their impact on the running time
- Example application
  - Runtime estimations for the $(1 + \lambda)$ EA on the Ruggedness problem, $n = 100$
  - Analysis of (A, b) and 2-rate parameter control methods
- The method is restricted to settings in which states are not visited more than once
- Possible solution:
  - construction of Markov chains on all states with equal fitness
  - solving the resulting system of equations
- Not limited to $(1 + \lambda)$ type algorithms

Thank you!