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)

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 Algorithms

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

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

**Data**: $n$: problem size; $f : \{0, 1\}^n \rightarrow \mathbb{R}$: function to maximize; $\lambda$: population size;

$\mathcal{D}(p)$: a family of parameterized distributions over $[0..n]

1. Sample parent $x \in \{0, 1\}^n$ uniformly at random;
2. for $t \leftarrow 1, 2, \ldots$ do
   3. for $i \in [1..\lambda]$ do
      4. Choose a distribution parameter $p_{i}^{t}$;
      5. Sample $k_i \sim \mathcal{D}(p_{i}^{t})$, the number of bits to flip;
      6. Create $y_i$ by flipping $k_i$ different bits in $x$ chosen uniformly at random;
    7. 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

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

1  \( f_{\text{min}}, f_{\text{max}} \leftarrow \text{minimum and maximum fitness values}; \)
2  Initialize optimal times: \( T_{f_{\text{max}}}^* \leftarrow 0; \)
Description of the Proposed Method

1 $f_{\text{min}}, f_{\text{max}} \leftarrow$ minimum and maximum fitness values;
2 Initialize optimal times: $T^*_{f_{\text{max}}} \leftarrow 0$;
3 for $f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}}$ do
Description of the Proposed Method

1. $f_{\text{min}}, f_{\text{max}} \leftarrow$ minimum and maximum fitness values;
2. Initialize optimal times: $T_{f_{\text{max}}}^* \leftarrow 0$;
3. for $f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}}$ do
4.     for $p \in \{ p_1^{(f)}, p_2^{(f)}, \ldots, p_{mf}^{(f)} \}$ do

Description of the Proposed Method

1. \( f_{\min}, f_{\max} \leftarrow \) minimum and maximum fitness values;
2. Initialize optimal times: \( T_{f_{\max}}^* \leftarrow 0; \)
3. **for** \( f \leftarrow f_{\max} - 1, \ldots, f_{\min} \) **do**
   4. **for** \( p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\} \) **do**
   5. Compute approximate probabilities \((\tilde{p}_i)_{i=0,1,\ldots}\) of increasing fitness by \( i \) with mutation rate \( p \) using the Monte Carlo approach;
   6. \( T_{f,p} \leftarrow \frac{1}{1 - \tilde{p}_0} \left( 1 + \sum_{i>0} \tilde{p}_i \cdot T_{f_{\max}}^* + i \right) \)
7. Store optimal time: \( T_{f_{\min}}^* \leftarrow \min p \left( T_{f,p} \right) \)
8. Store optimal rate: \( P_{\text{opt}}^{f_{\min}} \leftarrow \arg \min p \left( T_{f,p} \right) \)
9. **return** \{\( P_{\text{opt}}^{f_{\min}}, T_{f_{\min}}^*, T_{f_{\max}}^* \}\)
Description of the Proposed Method

1. \( f_{\text{min}}, f_{\text{max}} \leftarrow \) minimum and maximum fitness values;
2. Initialize optimal times: \( T_{f_{\text{max}}}^* \leftarrow 0; \)
3. for \( f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}} \) do
   4. for \( p \in \{ p_1^{(f)}, p_2^{(f)}, \ldots, p_m^{(f)} \} \) do
      5. Compute approximate probabilities \( (\tilde{p}_i)_{i=0,1,...} \) of increasing fitness by \( i \) with mutation rate \( p \) using the Monte Carlo approach;
      6. \( T_{f,p} \leftarrow \frac{1}{1 - \tilde{p}_0} \left( 1 + \sum_{i>0} \tilde{p}_i \cdot T_{f+i}^* \right); \)
9. return \( \{ P_{\text{opt}}, T^*_f, T^{*}_{f_{\text{max}}} \} \)

Requirement: the optimal choice of \( p \) depends on the fitness value exclusively.
Description of the Proposed Method

1. \( f_{\text{min}}, f_{\text{max}} \leftarrow \) minimum and maximum fitness values;
2. Initialize optimal times: \( T_{f_{\text{max}}}^* \leftarrow 0; \)
3. \textbf{for} \( f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}} \) \textbf{do}
   \begin{align*}
   &\text{for} \ p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\} \textbf{ do} \\
   &\quad \text{Compute approximate probabilities} \ (\tilde{p}_i)_{i=0,1,\ldots} \text{ of increasing fitness by } i \text{ with} \ \\
   &\quad \text{mutation rate } p \text{ using the Monte Carlo approach;} \\
   &\quad T_{f,p} \leftarrow \frac{1}{1 - \tilde{p}_0} \left(1 + \sum_{i>0} \tilde{p}_i \cdot T_{f+i}^*\right); \\
   &\text{Store optimal time: } T_f^* \leftarrow \min_p (T_{f,p}); \\
   &\text{Store optimal rate: } P_f^{\text{opt}} \leftarrow \arg \min_p (T_{f,p});
   \end{align*}
4. \textbf{return} \{P_f^{\text{opt}}, T_f^*, T_{f_{\text{max}}}^*\} \\

\textbf{Requirement:} the optimal choice of \( p \) depends on the fitness value exclusively
Description of the Proposed Method

1. \( f_{\text{min}}, f_{\text{max}} \leftarrow \) minimum and maximum fitness values;
2. Initialize optimal times: \( T_{f_{\text{max}}}^* \leftarrow 0; \)
3. for \( f \leftarrow f_{\text{max}} - 1, \ldots, f_{\text{min}} \) do
4.     for \( p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_{m_f}^{(f)}\} \) do
5.         Compute approximate probabilities \((\tilde{p}_i)_{i=0,1,\ldots}\) of increasing fitness by \( i \) with mutation rate \( p \) using the Monte Carlo approach;
6.         \[
       T_{f,p} \leftarrow \frac{1}{1 - \tilde{p}_0} \left( 1 + \sum_{i>0} \tilde{p}_i \cdot T_{f+i}^* \right);
       \]
7.     Store optimal time: \( T_f^* \leftarrow \min_p (T_{f,p}); \)
8. Store optimal rate: \( P_f^{\text{opt}} \leftarrow \arg \min_p (T_{f,p}); \)
9. return \( \{P^{\text{opt}}, T^*, T\} \)
Description of the Proposed Method

1. \( f_{\min}, f_{\max} \leftarrow \) minimum and maximum fitness values;
2. Initialize optimal times: \( T_{f_{\max}}^* \leftarrow 0; \)
3. \textbf{for} \( f \leftarrow f_{\max} - 1, \ldots, f_{\min} \) \textbf{do}
4. \quad \textbf{for} \( p \in \{p_1^{(f)}, p_2^{(f)}, \ldots, p_m^{(f)}\} \) \textbf{do}
5. \quad \text{Compute approximate probabilities} \( (\tilde{p}_i)_{i=0,1,\ldots} \) \text{of increasing fitness by} \( i \) \text{with}
6. \quad \text{mutation rate} \( p \) \text{using the Monte Carlo approach;}
7. \quad \begin{aligned} T_{f,p} & \leftarrow \frac{1}{1 - \tilde{p}_0} \left( 1 + \sum_{i>0} \tilde{p}_i \cdot T_{f+i}^* \right); \end{aligned}
8. \quad \text{Store optimal time:} \quad T_{f}^* \leftarrow \min_p(T_{f,p});
9. \quad \text{Store optimal rate:} \quad P_{f}^{\text{opt}} \leftarrow \arg \min_p(T_{f,p});
10. \textbf{return} \ \{P_{f}^{\text{opt}}, T_{f}^*, T\}

Requirement: the optimal choice of \( p \) depends on the fitness value exclusively
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 $p_{\text{min}} = 1/n^2$?
Regular oscillations on Ruggedness with a period of 2
- It may be difficult to track precisely – is this a problem?
Parameter Efficiency Heatmaps

OneMax

Ruggedness

- 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

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!
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!
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!