A First Step towards the Runtime Analysis of Evolutionary Algorithm Adjusted with Reinforcement Learning

Maxim Buzdalov  Arina Buzdalova

National Research University of IT, Mechanics and Optics
Saint-Petersburg, Russia

13 September 2013
Workshop on Theory of Randomised Search Heuristic (ThRaSH 2013)
Outline

Introduction

Area Overview
EA + RL Method
Application Example

Analyzed Algorithm

Modified OneMax Problem
RMHC Algorithm
Q-learning with Greedy Strategy
RMHC + Q-learning Algorithm
Theorem Formulation

Proof

Learning Lemma
Markov Chain
Linear Markov Chain
Complexity Estimation

Conclusion
Introduction

Area Overview
EA + RL Method
Application Example

Analyzed Algorithm
Modified OneMax Problem
RMHC Algorithm
Q-learning with Greedy Strategy
RMHC + Q-learning Algorithm
Theorem Formulation

Proof
Learning Lemma
Markov Chain
Linear Markov Chain
Complexity Estimation

Conclusion
Adjusting Evolutionary Algorithms with Reinforcement Learning

- EA + RL: select fitness function for each EA population
- Other methods
  - select evolutionary operator (mutation, crossover, ...)
  - adjust real valued parameters (mutation rate, ...)
- No runtime analysis, just empirical results
Reinforcement Learning

- The agent applies some actions \( a \in A \) to the environment
- After each action the agent receives from the environment:
  - some representation of the current state \( s \in S \)
  - some numeric reward \( R(s, a) \), \( R : S \times A \rightarrow \mathbb{R} \)
- Goal: maximize the total amount of reward:
  \[
  E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \rightarrow \max
  \]
Auxiliary Fitness Functions

- The problem: maximize target fitness function \( g \):
  \[
g(x) \rightarrow \max_{x \in X}
\]

- The set of auxiliary fitness functions is given:
  \[
  H = \{h_i(x)\}
  \]

  No prior knowledge about \( h_i \) properties

- The goal: adjust evolutionary algorithm using \( \{h_i\} \), i.e. decrease number of populations needed to find solution
EA + RL Method

Control EA with reinforcement learning:

- agent chooses fitness function from \{h_i\} \cup g
- next generation is created, reward and state are returned
Application Example

- Generation of tests against solutions of programming challenge tasks
- Success: test which makes the solution exceed time limit
- Fitness functions:
  - Target: running time of the solution (T)
  - Aux: counters in the solution code (Q, I, L), that correlate with T

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FFs</th>
<th>Success, %</th>
<th>Populations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>GA</td>
<td>Q</td>
<td>95</td>
<td>3815</td>
</tr>
<tr>
<td>GA + RL</td>
<td>all</td>
<td>80</td>
<td>5817</td>
</tr>
<tr>
<td>GA</td>
<td>I</td>
<td>54</td>
<td>12669</td>
</tr>
<tr>
<td>GA</td>
<td>L</td>
<td>51</td>
<td>13755</td>
</tr>
<tr>
<td>GA</td>
<td>T</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Auxiliary set type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>efficient only</td>
<td>method &gt; EA</td>
</tr>
<tr>
<td>efficient and inefficient</td>
<td>method &gt; EA</td>
</tr>
<tr>
<td>inefficient only</td>
<td>method = EA</td>
</tr>
<tr>
<td>dynamically changes</td>
<td>method ≥ EA</td>
</tr>
</tbody>
</table>

Notation:

- **efficient** fitness function ⇒ target increases more rapidly
- **inefficient** – the rest
- “=” asymptotically equals, “>” outperforms
Introduction
Area Overview
EA + RL Method
Application Example

Analyzed Algorithm
Modified OneMax Problem
RMHC Algorithm
Q-learning with Greedy Strategy
RMHC + Q-learning Algorithm
Theorem Formulation

Proof
Learning Lemma
Markov Chain
Linear Markov Chain
Complexity Estimation

Conclusion
Modified OneMax Problem

- Individual — bit vector of length N
- Target fitness function $f_1$ — number of ones
- Inefficient aux. fitness function $f_0$ — number of zeros
- Reinforcement learning state — number of ones
Random Mutation Hill Climber

- $X$ — bit vector
- $\text{Mutate}(X)$ — inverts one random bit
- $f$ — fitness function

**RMHC algorithm**

1. Initialize $X$: vector of $N$ zeros
2. Repeat until termination condition is not reached
   2.1 $Y := \text{Mutate}(X)$
   2.2 If ($f(Y) \geq f(X)$) $X := Y$
Q-learning with Greedy Strategy

- $Q : S \times A \rightarrow \mathbb{R}$ — quality of applying action in state
- $\alpha$ — the learning rate
- $\gamma$ — the discount factor

Q-learning algorithm

1. Initialize $Q(s, a)$: fill it with zeros
2. Repeat until termination condition is not reached
   2.1 Select an action: $a := \arg \max_a Q(s_t, a)$
   2.2 Apply selected action, get reward $R$
   2.3 Update
      $$Q(s_t, a) := (1 - \alpha)Q(s_t, a) + \alpha(R + \gamma \max_{a'} Q(s_{t+1}, a'))$$
RMHC + Q-learning Algorithm

\[ s(t) = f_1(x_t) \]

\[ r(t) = f_1(x_t) - f_1(x_{t-1}) \]

\[ r(t+1) \]

\[ s(t+1) \]

Q-Learning Agent

RMHC

fitness function \((f_0 \text{ or } f_1)\)
RMHC + Q-learning Algorithm

X ← current individual, vector of N zeros
Q ← transition quality matrix, N × 2, filled with zeros
f₁ ← target fitness function: number of ones in an individual
f₀ ← inefficient fitness function: number of zeros in an individual
Mutate(X) ← mutation operator: inverts random bit
α ∈ (0; 1), γ ∈ (0; 1) — Q-learning parameters
while f₁(X) < N do
    S ← f₁(X)
    Y ← Mutate(X)
    f, I: chosen fitness function and its index
    if Q(S, 0) > Q(S, 1) then
        f ← F₀, I ← 0
    else if Q(S, 0) < Q(S, 1) then
        f ← F₁, I ← 1
    else
        I ← random(0,1), f ← F_I
    end if
    if f(Y) ≥ f(X) then
        X ← Y
    end if
    R ← F₁(X) − S
    Q(S, I) ← (1 − α)Q(S, I) + α(γ · R + max_j Q(S, j))
end while
Theorem

Random Mutation Hill Climber controlled with Q-learning algorithm with greedy exploration strategy solves modified OneMax problem in $\Theta(N \log N)$ fitness function calls.
Introduction
Area Overview
EA + RL Method
Application Example

Analyzed Algorithm
Modified OneMax Problem
RMHC Algorithm
Q-learning with Greedy Strategy
RMHC + Q-learning Algorithm
Theorem Formulation

Proof
Learning Lemma
Markov Chain
Linear Markov Chain
Complexity Estimation

Conclusion
Learning Lemma: Formulation

Assume that the Q-learning agent visits a state $S$ and leaves it. Then the optimal fitness function $f_1$ will be chosen in $S$ in all next visits.
Learning Lemma: Proof

- \( Q(S, I) := (1 - \alpha)Q(S, I) + \alpha(\gamma \cdot r + \max_j Q(S', j)) \)
- In both cases, \( Q(S, 1) > Q(S, 0) \)
- So \( f_1 \) will be chosen in \( S \)
Markov Chain: Overview
Markov Chain: Transition Probabilities
Expectation of Transition Number-1

\[ E(B_{i-1} \rightarrow C_i) = 1 \times \frac{N-i+1}{N} + (1 + E(B_{i-1} \rightarrow C_i)) \times \frac{i-1}{N} \]

\[ E(B_{i-1} \rightarrow C_i) = \frac{N}{N-i+1} \]
**Expectation of Transition Number-2**

\[
\begin{align*}
E(C_i \rightarrow A_{i+1}) &= 1 \times \frac{N-i}{N} + (1 + E(C_i \rightarrow A_{i+1})) \times \frac{i}{N} \\
E(C_i \rightarrow A_{i+1}) &= \frac{N}{N-i}
\end{align*}
\]
Expectation of Transition Number-3

\[ \text{Len}(A_i \rightarrow A_{i+1}) = 1 + E(B_{i-1} \rightarrow C_i) + E(C_i \rightarrow A_{i+1}) \]
Linear Markov Chain: Overview
Linear Markov Chain: Lengths and Probabilities

\[ p = \frac{N - i}{2N}, \]
\[ \text{length} = 1 + \frac{N}{N - i + 1} + \frac{N}{N - 1} \]

\[ p = \frac{1}{2}, \]
\[ \text{length} = 1 \]
\[ Z(i) = (1 + Z(i)) \times \frac{1}{2} + \frac{N-i}{2N} + (1 + \frac{N}{N-i+1} + \frac{N}{N-i}) \times \frac{i}{2N} \]

\[ Z(i) = 2 + \frac{i}{N-i+1} + \frac{i}{N-i} \]

For \( Z(0) \) this also holds

\[ T_R(N) = \sum_{i=0}^{N-1} \left( 2 + \frac{i}{N-i+1} + \frac{i}{N-i} \right) \]
Comparison with RMHC without Inefficient Fitness Function

- **RMHC without inefficient fitness function**
  - \( T_0(N) = \sum_{i=0}^{N-1} (1 + \frac{i}{N-i}) \)
  - \( T_0(N) = \Theta(N \log N) \)
- **RMHC + Q-learning**:
  - \( T_R(N) = \sum_{i=0}^{N-1} (2 + \frac{i}{N-i+1} + \frac{i}{N-i}) \)
  - \( 1 + \frac{i}{N-i} < 2 + \frac{i}{N-i+1} + \frac{i}{N-i} < 2 + 2\frac{i}{N-i} = 2(1 + \frac{i}{N-i}) \)
  - \( T_0(N) < T_R(N) < 2 \cdot T_0(N) \)
  - \( T_R(N) = \Theta(N \log N) \)
Introduction
Area Overview
EA + RL Method
Application Example

Analyzed Algorithm
Modified OneMax Problem
RMHC Algorithm
Q-learning with Greedy Strategy
RMHC + Q-learning Algorithm
Theorem Formulation

Proof
Learning Lemma
Markov Chain
Linear Markov Chain
Complexity Estimation

Conclusion
Conclusion

- EA + RL is proved to ignore an inefficient fitness function in a model problem
- Future work
  - Generalize the obtained result
  - EA + RL with an efficient fitness function outperforms EA
Thank you! Any questions?