Comparing Self-Adjusting $(1 + \lambda)$ EAs under Large Dimensions: A Case Study

Arina Buzdalova (ITMO) Carola Doerr (Sorbonne) Anna Rodionova (ITMO) Kirill Antonov (ITMO)

Working Group Workshop
COST Action CA15140
February 19, 2019

Experiment description

- $(1+\lambda)$ EA with "practice-aware" shift mutation
- 4 self-adjusting $(1+\lambda)$ EAs:
  - 2 different rules for updating mutation rate
    - 2-rate: asymptotically optimal runtime for large enough λ (theoretically proven)
    - Ab: chooses number of bits close to optimal (empirically shown)
  - 2 different mutation lower bounds: $1/n$ and $1/n^2$
- Problem: OneMax
- Dimensions
  - problem size $n = 10000, 20000, \ldots 100000$
  - population size $\lambda = 1, 5, 10, 50, 100, 200, 400, 800, 1600, 3200$
- 100 independent runs of each algorithm

Compared algorithms: $(1+\lambda)$ $\text{EA}_{0\to 1}$

Algorithm 1: The $(1+\lambda)$ $\text{EA}_{0\to 1}$ with mutation rate $p \in (0,1)$ for the maximization of $f : \{0,1\}^n \to \mathbb{R}$

Initialization: Sample $x \in \{0,1\}^n$ u.a.r.;
Optimization: **for** $t = 1, 2, 3, \ldots$ **do**
    for $i = 1, \ldots, \lambda$ **do**
        Sample $\ell^{(i)}$ from $\text{Bin}_{0\to 1}(n,p)$, sample $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ and evaluate $f(y^{(i)})$;
    Sample x^* from $\arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
    if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

Compared algorithms: 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$

Algorithm 2: The 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ with adaptive mutation rates proposed in [DoerrGWY17]

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Initialize $r \leftarrow r^{\text{init}}$; // We use $r^{\text{init}} = 2$;
Optimization: **for** $t = 1, 2, 3, \ldots$ **do**
- **for** $i = 1, \ldots, \lfloor \lambda/2 \rfloor$ **do**
  - Sample $\ell^{(i)} \sim \mathrm{Bin}_{0\to 1}(n, r/(2n))$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
- **for** $i = \lfloor \lambda/2 \rfloor + 1, \ldots, \lambda$ **do**
  - Sample $\ell^{(i)} \sim \mathrm{Bin}_{0\to 1}(n, 2r/n)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
- $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken u.a.r.);
- **if** $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;
- Perform one of the following two actions with prob. $1/2$:
  - replace r with the mutation rate that x^* has been created with;
  - replace r with either $2r$ or $r/2$ equiprobably.
- $r \leftarrow \min\{\max\{2, r\}, n/4\}$;

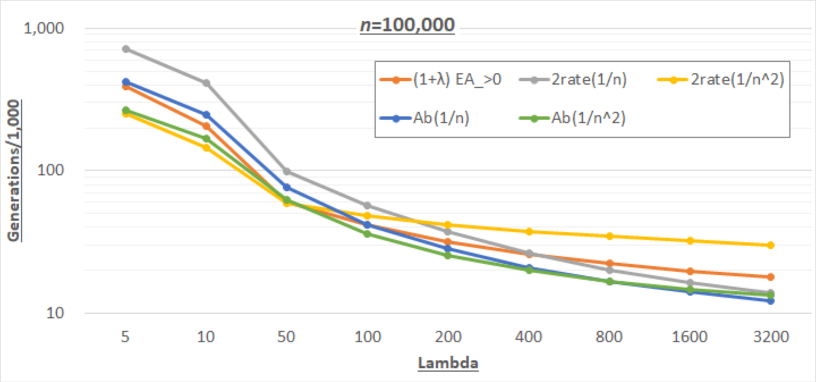
Compared algorithms: $(1+\lambda)$ EA(A, b)

Algorithm 3: The $(1+\lambda)$ EA(A, b) with adaptive mutation rates and update strengths $A > 1$, $0 < b < 1$

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Initialize $p \leftarrow 1/n$; **Optimization:** **for** $t = 1, 2, 3, \dots$ **do**
  for $i = 1, \dots, \lambda$ **do**
    Sample $\ell^{(i)} \sim \mathrm{Bin}_{0\to 1}(n, p)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $N \leftarrow |\{i \in [\lambda] \mid f(x^{(i)}) \geq f(x)\}|$;
  if $N \geq \lceil 0.05\lambda \rceil$ **then** $p \leftarrow \min\{1/2, Ap\}$ **else** $p \leftarrow \max\{1/n, bp\}$;
  $x^* \leftarrow \arg\max\{f(x^{(1)}), \dots, f(x^{(\lambda)})\}$ (ties broken u.a.r.);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

Comparison regarding different population sizes

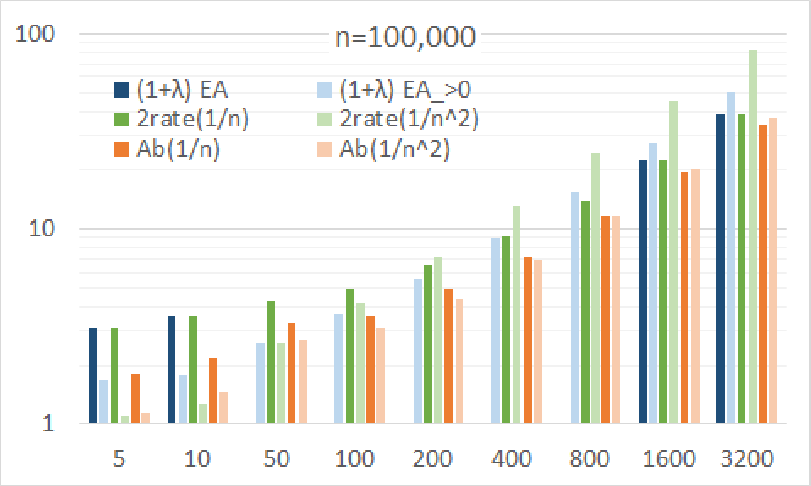
Number of generations:



- Almost opposite ranking for small and large population sizes λ
- For both algorithms, $1/n^2$ lower bound is better for small λ, $1/n$ is better for large λ

Comparison regarding different population sizes

Number of fitness evaluations:

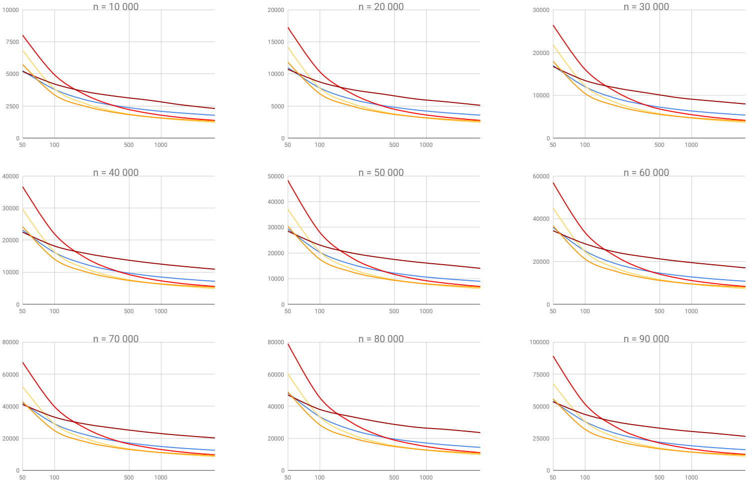


Comparison regarding different population sizes

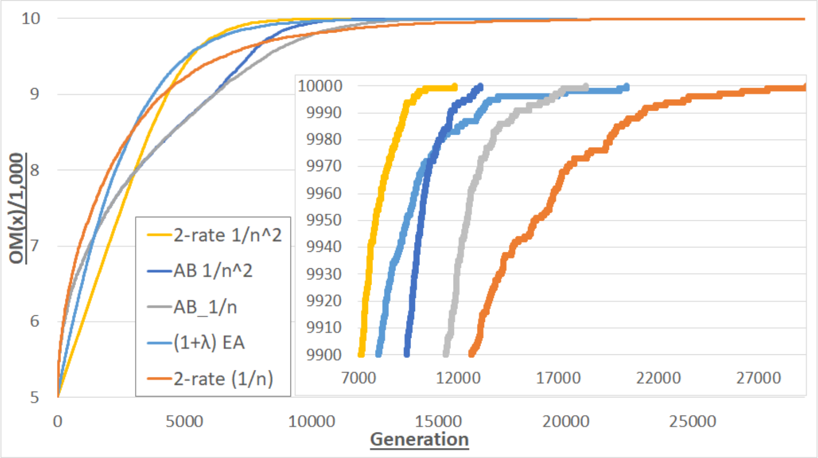
Number of generations averaged by 100 runs and its standard deviation. Shift mutation operator is used in all algorithms, avg.=average, r.dev.=relative standard deviation.

λ	$(1+\lambda)$ $\mathrm{EA}_{0\to 1}$		2-rate $(1/n)$		Ab $(1/n)$		2-rate $(1/n^2)$		Ab $(1/n^2)$	
	avg.	r.dev.	avg.	r.dev.	avg.	r.dev.	avg.	r.dev.	avg.	r.dev.
$n = 100000$										
1	1,873,666	13.4%	2,222,691	10.9%	1,860,718	13.4%	1,143,933	11.7%	1,122,686	10.6%
5	389,110	10.2%	716,372	13.6%	420,165	10.4%	253,747	11.7%	264,171	8.5%
10	205,086	12.0%	414,043	10.4%	248,184	9.7%	146,132	8.2%	167,244	7.0%
50	60,200	6.2%	98,223	8.7%	76,772	5.6%	59,357	5.7%	62,693	3.8%
100	42,151	5.0%	57,325	6.9%	41,520	4.1%	48,659	6.6%	35,814	4.1%
200	31,762	3.4%	37,588	6.8%	28,215	3.7%	41,634	4.6%	25,431	2.2%
400	25,846	2.1%	26,227	5.0%	20,900	2.6%	37,583	8.3%	19,984	1.6%
800	22,229	1.2%	20,055	3.2%	16,681	1.4%	34,965	9.3%	16,691	0.9%
1,600	19,744	0.7%	16,325	2.0%	14,112	1.0%	32,494	11.6%	14,691	0.6%
3,200	17,956	0.4%	13,966	1.1%	12,296	0.6%	29,796	11.1%	13,344	0.5%
$n = 10000$										
1	147,008	14.8%	177,568	14.8%	148,182	14.1%	90,459	13.0%	91,563	15.6%
5	31,738	17.5%	56,940	17.3%	34,514	13.0%	20,657	13.0%	21,999	13.6%
10	16,373	13.1%	32,054	15.3%	20,662	9.2%	12,036	11.1%	14,252	7.9%
50	5,254	7.5%	8,029	12.9%	6,855	6.7%	5,198	8.7%	5,740	3.7%
100	3,790	4.6%	4,922	9.4%	3,824	5.8%	4,211	9.7%	3,360	4.0%
200	2,955	3.6%	3,323	8.2%	2,647	3.7%	3,603	10.5%	2,447	2.8%
400	2,486	2.1%	2,420	5.7%	2,001	2.4%	3,227	11.8%	1,943	1.8%
800	2,164	1.1%	1,903	3.8%	1,633	1.8%	2,932	13.8%	1,639	1.3%
1,600	1,945	0.8%	1,582	2.1%	1,393	1.2%	2,576	11.1%	1,450	0.8%
3,200	1,776	0.6%	1,379	1.8%	1,227	0.8%	2,305	12.3%	1,323	0.8%

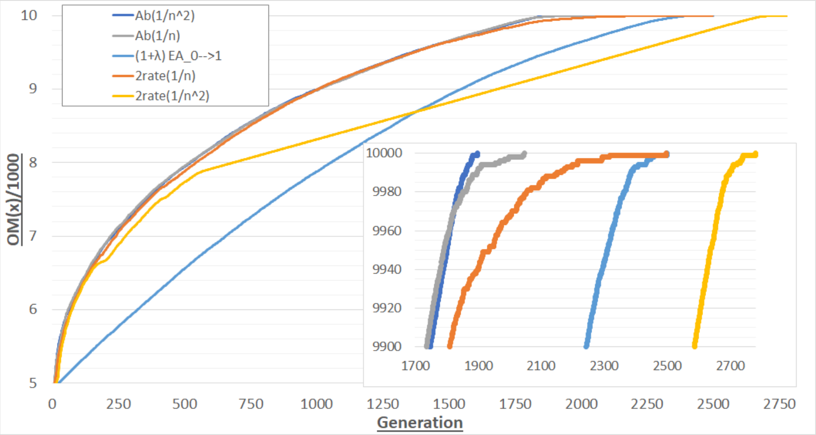
Comparison regarding different problem sizes



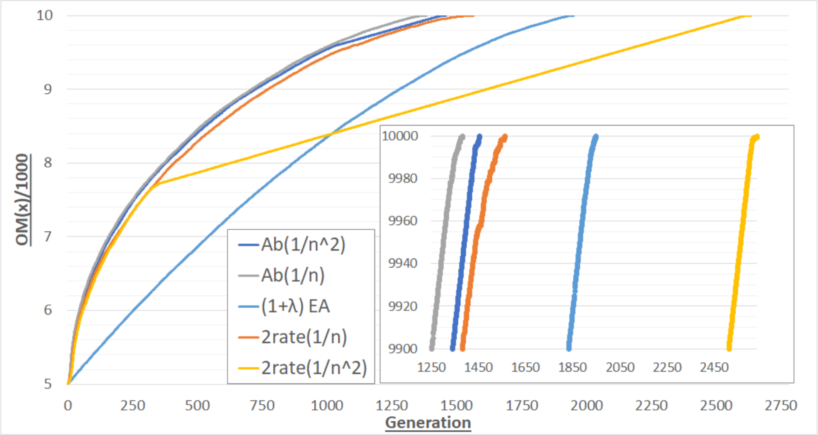
Fixed Budget Results: small population size $\lambda = 10$



Fixed Budget Results: medium population size $\lambda = 400$



Fixed Budget Results: large population size $\lambda = 1600$



Fast implementation of the standard mutation operator

```
calc_fitness (patch, best_individual, best_fitness, mask) {
     patch_fitness := best_fitness
     for (i in patch){
          if (best_individual[i] == mask[i])
               patch_fitness := patch_fitness - 1
          else
               patch_fitness := patch_fitness + 1
     }
     return patch_fitness
}

apply_patch(patch, best_individual) {
     for (i in patch)
          best_individual[i] := 1 - best_individual[i]
}
```

Conclusion

- $(1+\lambda)$ EA and 4 self-adjusting $(1+\lambda)$ EAs were compared on OneMax for $n = 10^4 \ldots 10^5, \lambda = 2 \ldots 3200$
- Results strongly influenced by the population size:
  1. Ranking in terms of runtime
  2. Ranking in terms of fixed budget
  3. Relative standard deviation
- (1), (3) does not depend so much on problem size
- Possible consequences for benchmarking:
  - In a benchmarking framework, plotting against wide parameter range should be available
  - Fast implementation of fitness evaluation and standard operators may be needed