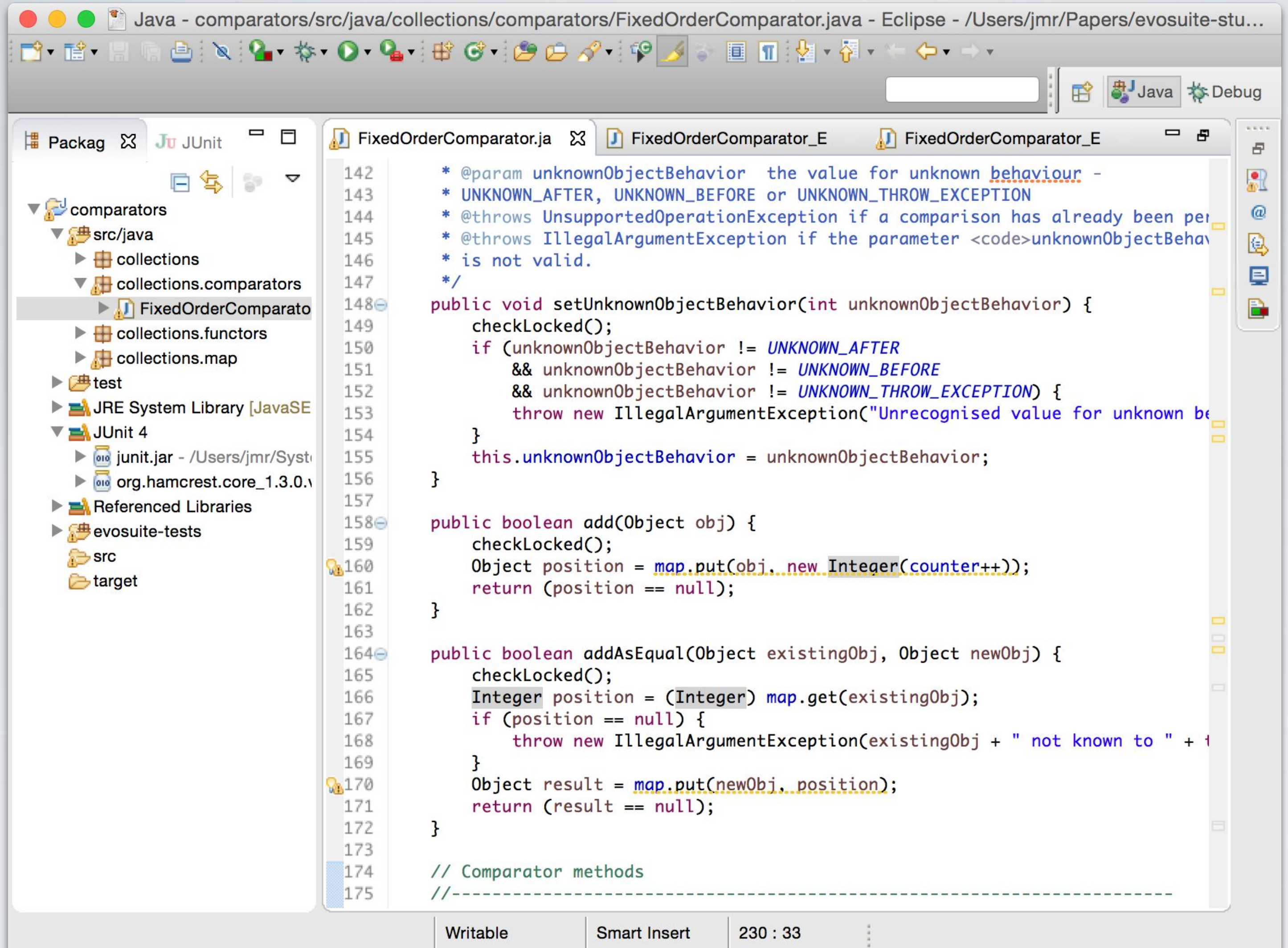
An Empirical Evaluation of Evolutionary Algorithms for Test Suite Generation

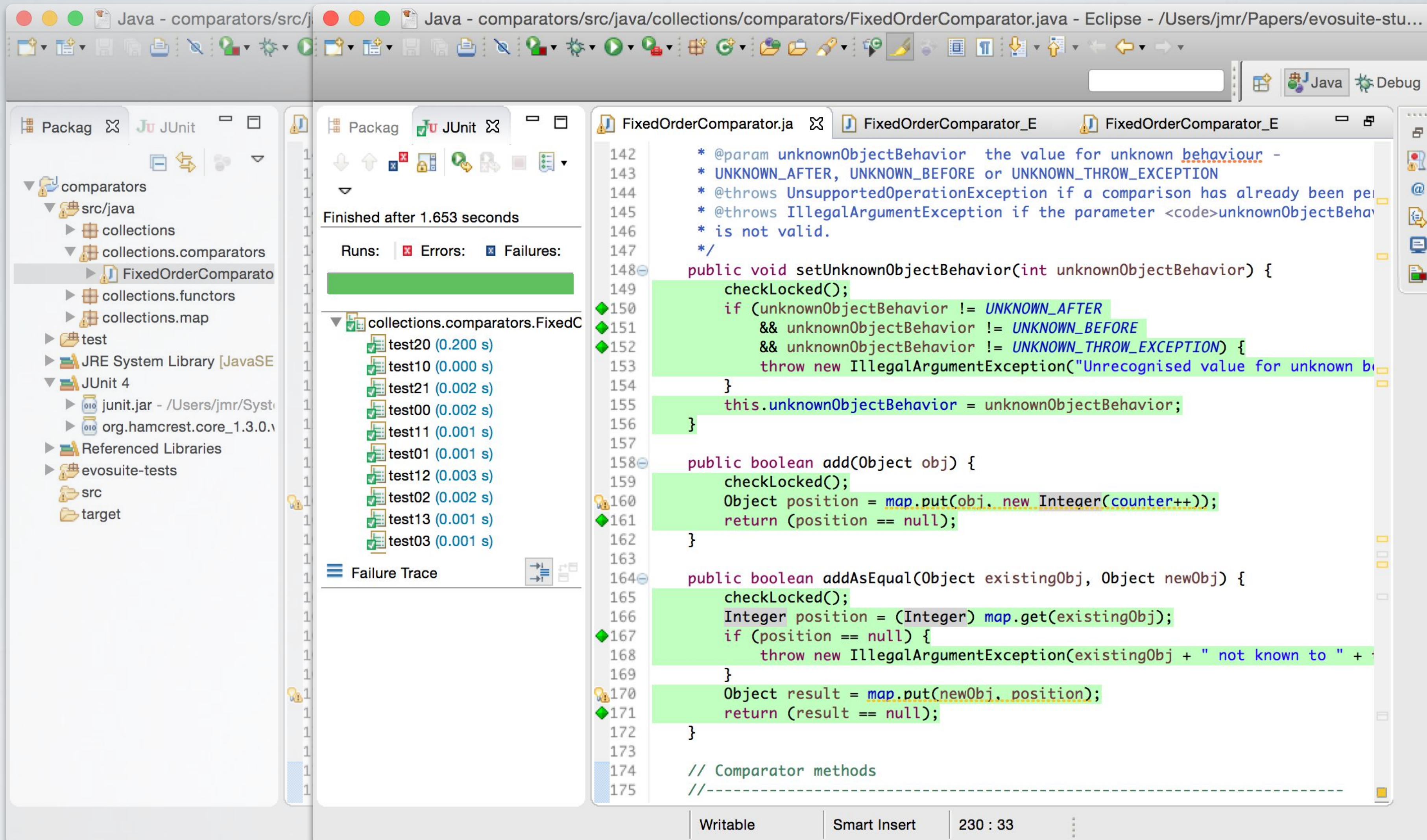
José Campos, Yan Gen, Gordon Fraser, Marcelo Eler, Andrea Arcuri

September 11th, 2017
9th Symposium on Search-Based Software Engineering (SSBSE)
Paderborn, Germany

The University Of Sheffield.
University of São Paulo
Westerdals Oslo School of Arts, Communication and Technology
Université du Luxembourg



Class Under Test



Test Suite optimised for structural coverage

Research Questions

RQ1 - Which evolutionary algorithm works best for test suite optimisation?

RQ2 - How does evolutionary search compare to random search and random testing?

RQ3 - How does evolution of whole test suites compare to many-objective optimisation of test cases?

Standard GA

(1 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

```
P ← GenerateRandomPopulation(p_s)
PerformFitnessEvaluation(δ, P)
while ¬C do
    N_P ← {}
    while |N_P| < p_s do
        p_1, p_2 ← Selection(s_f, P)
        o_1, o_2 ← Crossover(c_f, c_p, p_1, p_2)
        Mutation(m_f, m_p, o_1)
        Mutation(m_f, m_p, o_2)
        N_P ← N_P ∪ {o_1, o_2}
    end while
    P ← N_P
    PerformFitnessEvaluation(δ, P)
end while
return P
```

Standard GA

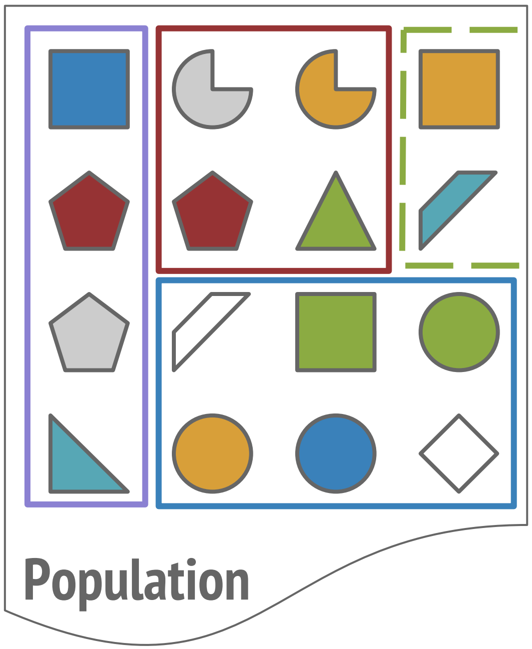
(1 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$\boxed{P \leftarrow \textsc{GenerateRandomPopulation}(p_s)}$
$\textsc{PerformFitnessEvaluation}(\delta, P)$
while $\neg C$ **do**
  $N_P \leftarrow \{\}$
  while $|N_P| < p_s$ **do**
    $p_1, p_2 \leftarrow \textsc{Selection}(s_f, P)$
    $o_1, o_2 \leftarrow \textsc{Crossover}(c_f, c_p, p_1, p_2)$
    $\textsc{Mutation}(m_f, m_p, o_1)$
    $\textsc{Mutation}(m_f, m_p, o_2)$
    $N_P \leftarrow N_P \cup \{o_1, o_2\}$
  end while
  $P \leftarrow N_P$
  $\textsc{PerformFitnessEvaluation}(\delta, P)$
end while
return P

Initial Population




```
@Test
public void test0() {
  int var0 = 10;
  YearMonthDay var1 = new YearMonthDay(var0);
  TimeOfDay var2 = new TimeOfDay();
  DateTime var3 = var1.toDateTime(var2);
  DateTime var4 = var3.minus(var0);
  DateTime var5 = var4.plusSeconds(var0);
}

@Test
public void test1() {
  DateTime var0 = new DateTime("11-09-2017");
  DateTime var1 = new DateTime("25-12-2017");
  int var2 = DateTime.sub(var0, var1);
}
```

© Fraser and Arcuri, Whole Test Suite Generation, TSE 2013

Standard GA

(1 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$P \leftarrow \textsc{GenerateRandomPopulation}(p_s)$

$\boxed{\textsc{PerformFitnessEvaluation}(\delta, P)}$

while $\neg C$ **do**

    $N_P \leftarrow \{\}$

    while $|N_P| < p_s$ **do**

        $p_1, p_2 \leftarrow \textsc{Selection}(s_f, P)$

        $o_1, o_2 \leftarrow \textsc{Crossover}(c_f, c_p, p_1, p_2)$

        $\textsc{Mutation}(m_f, m_p, o_1)$

        $\textsc{Mutation}(m_f, m_p, o_2)$

        $N_P \leftarrow N_P \cup \{o_1, o_2\}$

    end while

    $P \leftarrow N_P$

    $\textsc{PerformFitnessEvaluation}(\delta, P)$

end while

return P

Fitness Evaluation

(line coverage)

T1 T2 T3 T4

```
  public Complex log() {
1   if (isNaN) {
2     return NaN;
    }

3   double r = log(abs());
4   double i = atan2(imaginary, real);
5   return createComplex(r, i);
  }

  public Complex pow(double x) throws NullArgumentException {
6   Complex c = this.log();
7   return c.multiply(x).exp();
  }
```

© Rojas, Campos, Vivantii, Fraser, Arcuri, Combining Multiple Coverage Criteria in Search-Based Unit Test Generation, SSBSE 2015

Fitness Evaluation

(line coverage)

```
  public Complex log() {
1   if (isNaN) {
2     return NaN;
    }

3   double r = log(abs());
4   double i = atan2(imaginary, real);
5   return createComplex(r, i);
  }

  public Complex pow(double x) throws NullArgumentException {
6   Complex c = this.log();
7   return c.multiply(x).exp();
  }
```

Line	T1	T2	T3	T4
1	●	●	●	●
2	●			
3			●	●
4			●	●
5			●	●
6		●	●	
7		●	●	
	2/7	3/7	6/7	4/7

© Rojas, Campos, Vivantii, Fraser, Arcuri, Combining Multiple Coverage Criteria in Search-Based Unit Test Generation, SSBSE 2015

Standard GA

(1 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$P \leftarrow \textsc{GenerateRandomPopulation}(p_s)$
$\textsc{PerformFitnessEvaluation}(\delta, P)$
while $\neg C$ **do**
    $N_P \leftarrow \{\}$
    while $|N_P| < p_s$ **do**
        $p_1, p_2 \leftarrow \textsc{Selection}(s_f, P)$
        $o_1, o_2 \leftarrow \textsc{Crossover}(c_f, c_p, p_1, p_2)$
        $\textsc{Mutation}(m_f, m_p, o_1)$
        $\textsc{Mutation}(m_f, m_p, o_2)$
        $N_P \leftarrow N_P \cup \{o_1, o_2\}$
    end while
    $P \leftarrow N_P$
    $\textsc{PerformFitnessEvaluation}(\delta, P)$
end while
return P

Standard GA

(1 out of 5 EAs)

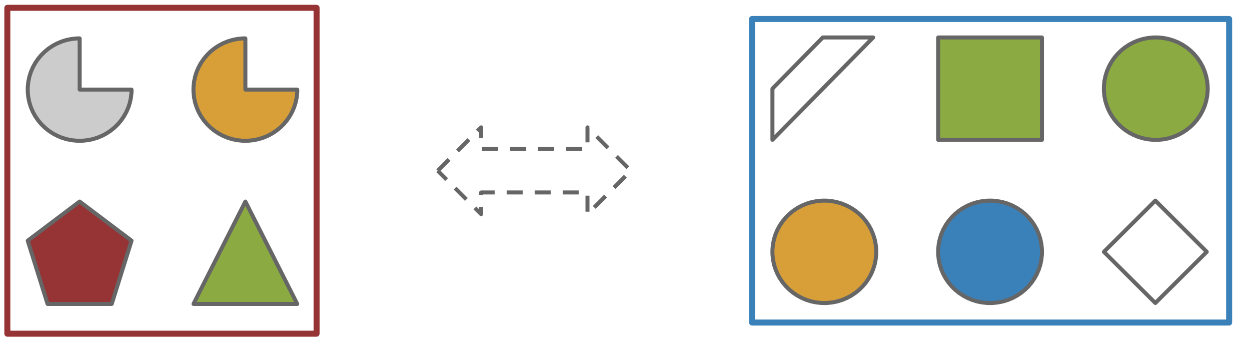
Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$P \leftarrow \textsc{GenerateRandomPopulation}(p_s)$
$\textsc{PerformFitnessEvaluation}(\delta, P)$
while $\neg C$ **do**
    $N_P \leftarrow \{\}$
    while $|N_P| < p_s$ **do**
        $p_1, p_2 \leftarrow \textsc{Selection}(s_f, P)$
        $o_1, o_2 \leftarrow \textsc{Crossover}(c_f, c_p, p_1, p_2)$
        $\textsc{Mutation}(m_f, m_p, o_1)$
        $\textsc{Mutation}(m_f, m_p, o_2)$
        $N_P \leftarrow N_P \cup \{o_1, o_2\}$
    end while
    $P \leftarrow N_P$
    $\textsc{PerformFitnessEvaluation}(\delta, P)$
end while
return P

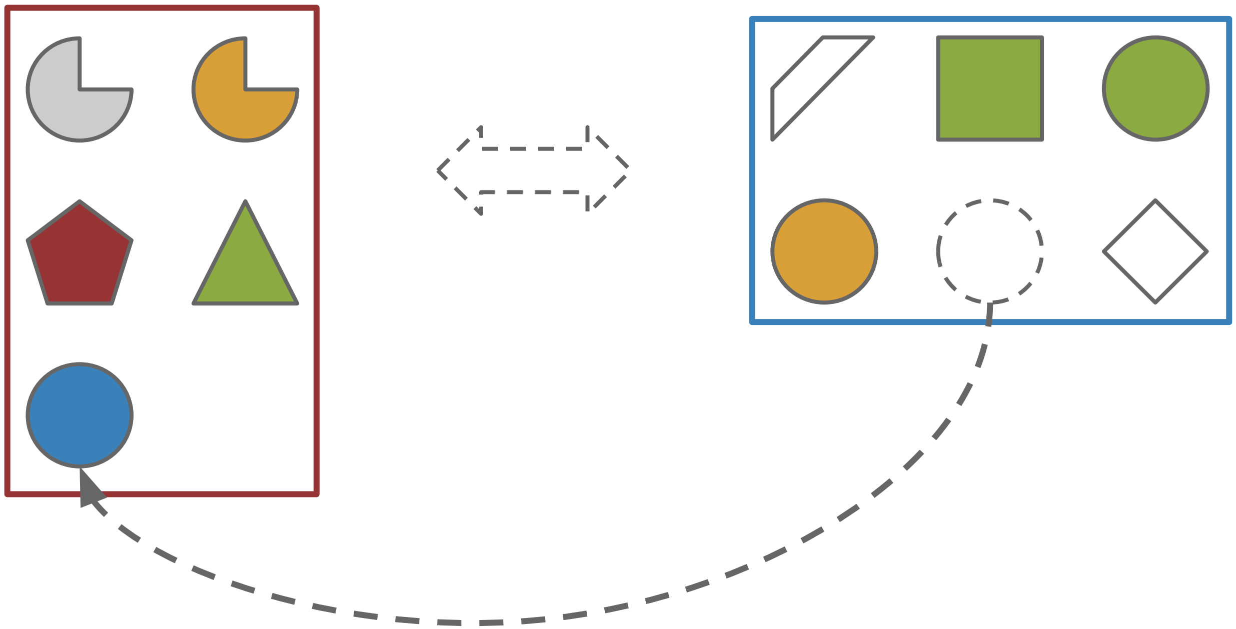
Crossover

(exchanging test cases)



Crossover

(exchanging test cases)



Standard GA

(1 out of 5 EAs)

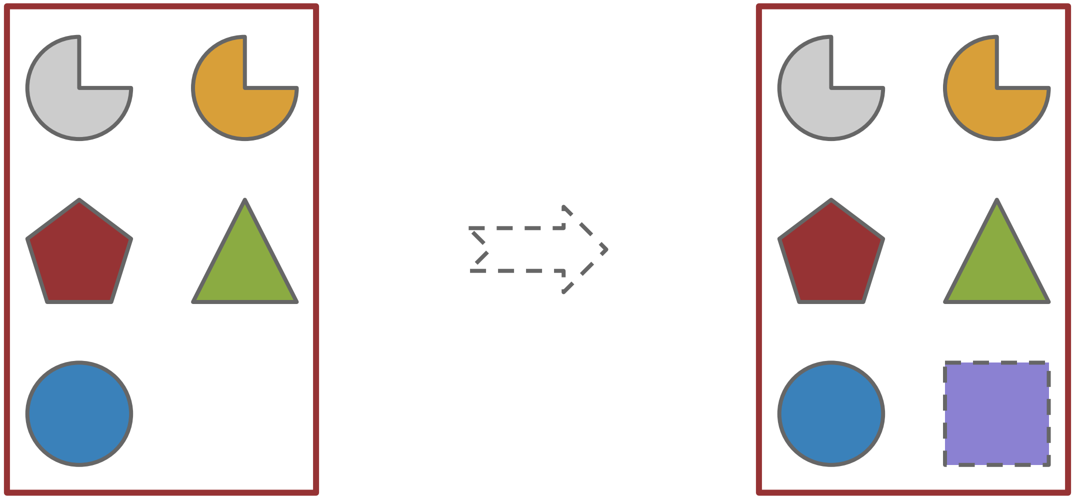
Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$P \leftarrow$ GenerateRandomPopulation(p_s)
PerformFitnessEvaluation(δ, P)
while $\neg C$ **do**
    $N_P \leftarrow \{\}$
    while $|N_P| < p_s$ **do**
        $p_1, p_2 \leftarrow$ Selection(s_f, P)
        $o_1, o_2 \leftarrow$ Crossover(c_f, c_p, p_1, p_2)
        Mutation(m_f, m_p, o_1)
        Mutation(m_f, m_p, o_2)
        $N_P \leftarrow N_P \cup \{o_1, o_2\}$
    end while
    $P \leftarrow N_P$
    PerformFitnessEvaluation(δ, P)
end while
return P

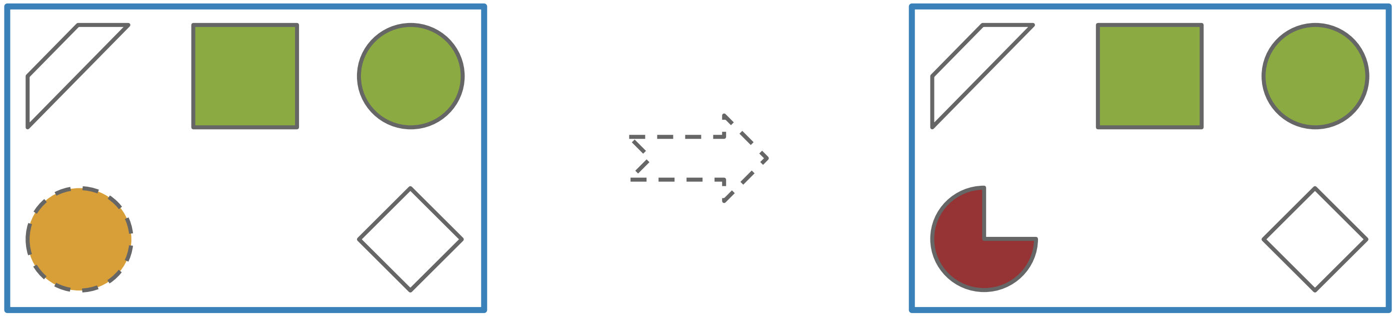
Mutation

(adding a new test case)



Mutation

(modifying an existing test case)



Standard GA

(1 out of 5 EAs)

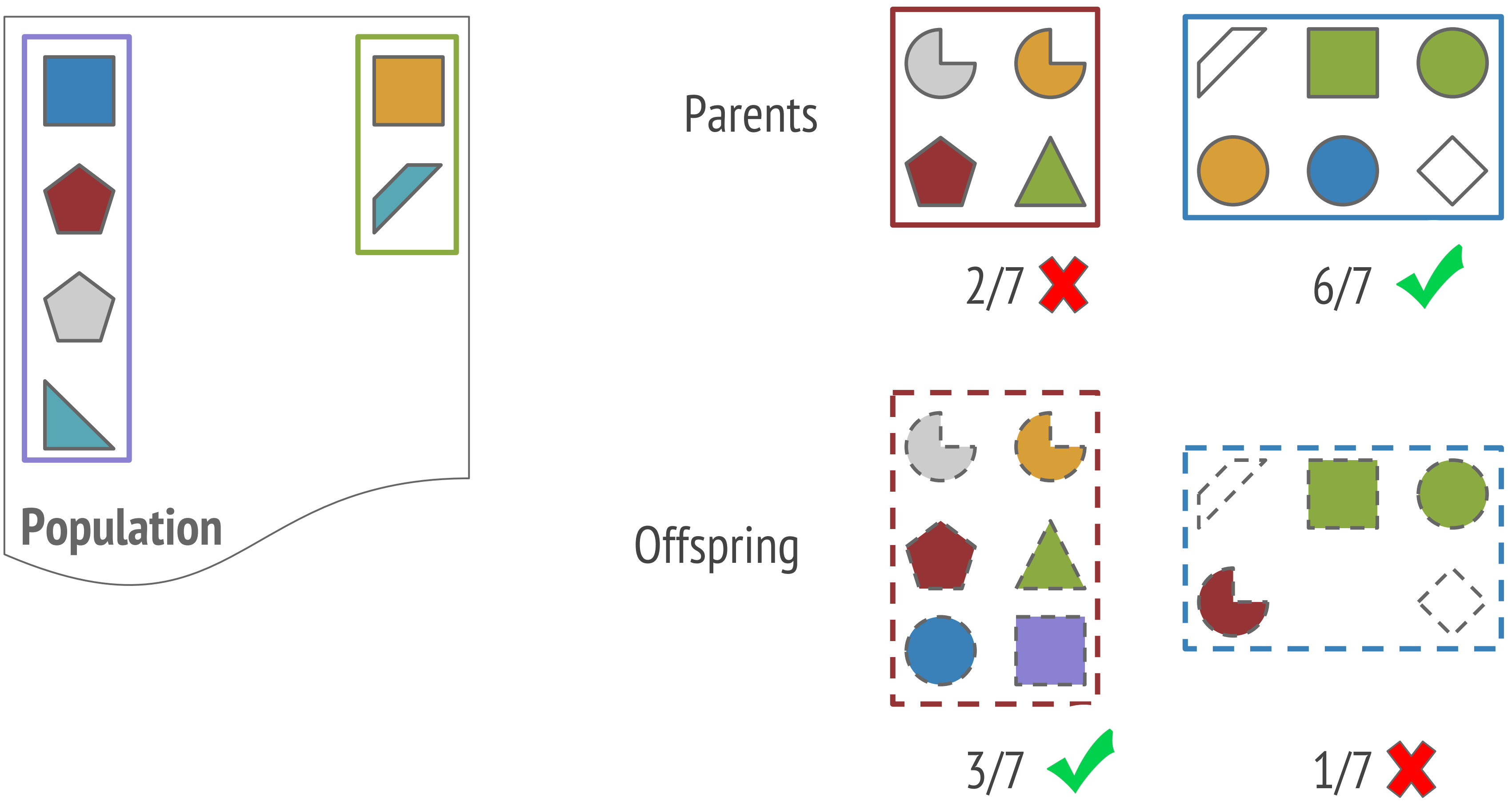
Input: Stopping condition C, Fitness function δ, Population size p_s, Selection function s_f, Crossover function c_f, Crossover probability c_p, Mutation function m_f, Mutation probability m_p

Output: Population of optimised individuals P

$P \leftarrow$ GenerateRandomPopulation(p_s)
PerformFitnessEvaluation(δ, P)
while $\neg C$ **do**
    $N_P \leftarrow \{\}$
    while $|N_P| < p_s$ **do**
        $p_1, p_2 \leftarrow$ Selection(s_f, P)
        $o_1, o_2 \leftarrow$ Crossover(c_f, c_p, p_1, p_2)
        Mutation(m_f, m_p, o_1)
        Mutation(m_f, m_p, o_2)
        $N_P \leftarrow N_P \cup \{o_1, o_2\}$
    end while
    $P \leftarrow N_P$
    PerformFitnessEvaluation(δ, P)
end while
return P

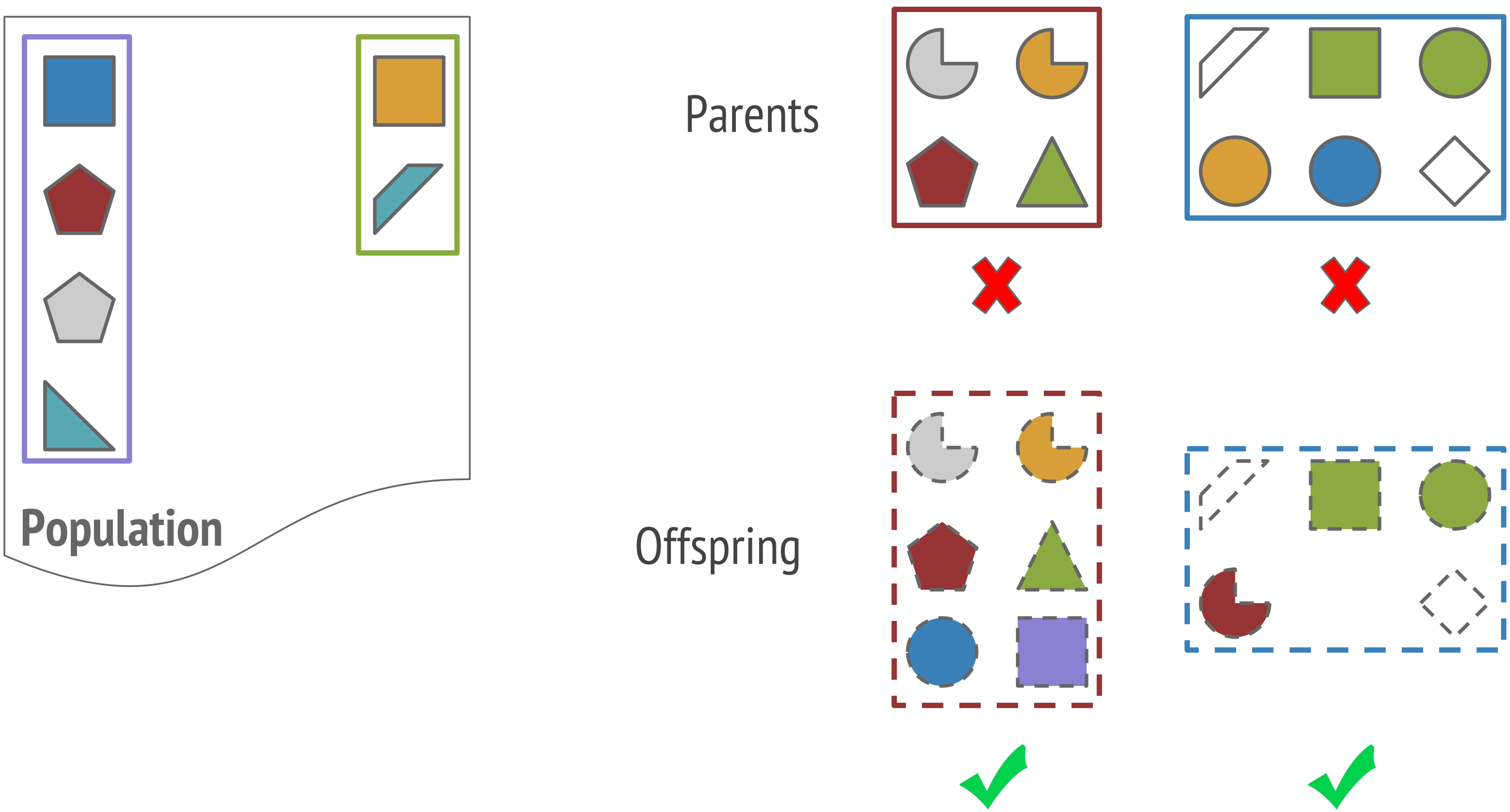
Monotonic GA

(2 out of 5 EAs)



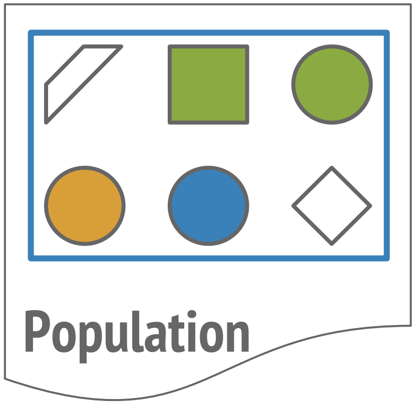
Steady-State GA

(3 out of 5 EAs)



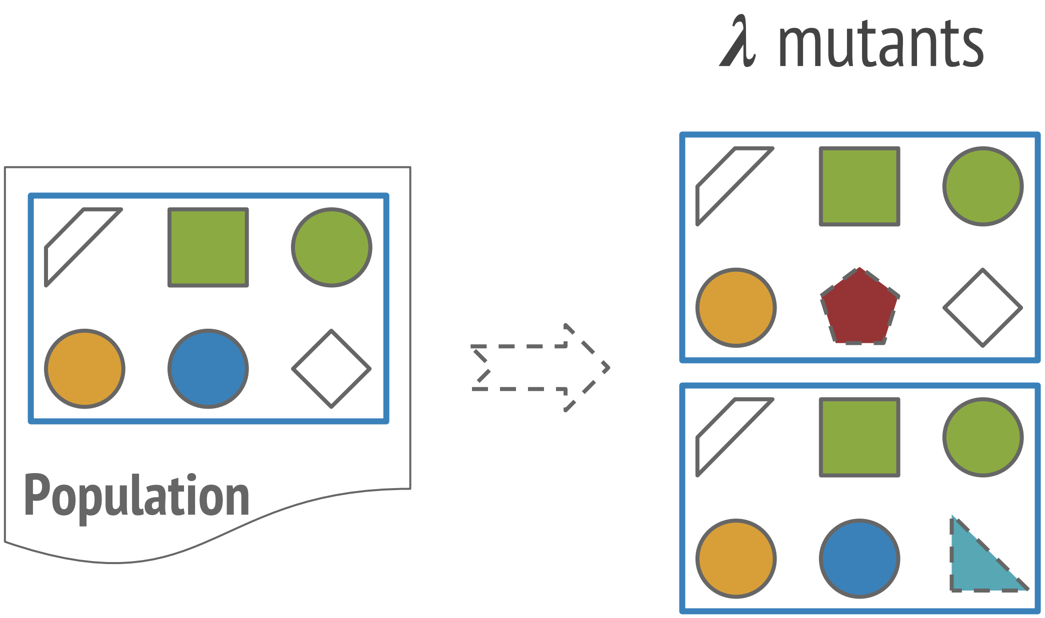
$1 + (\lambda, \lambda)$ GA

(4 out of 5 EAs)



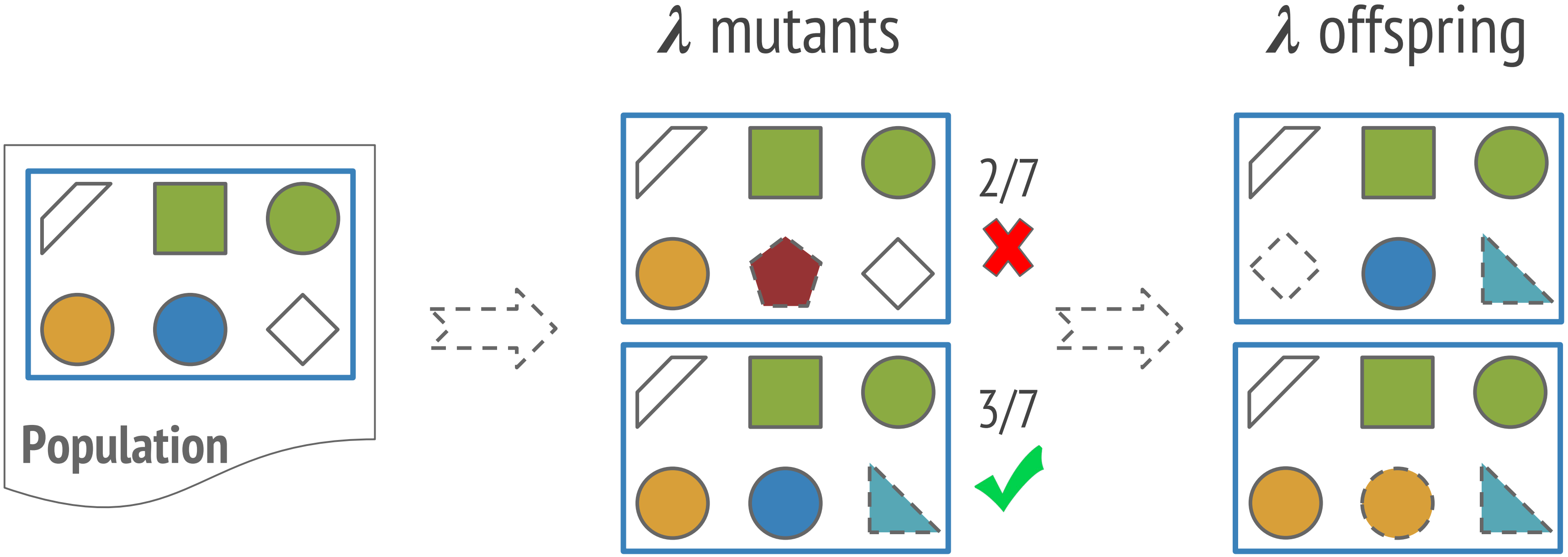
1 + (λ, λ) GA

(4 out of 5 EAs)



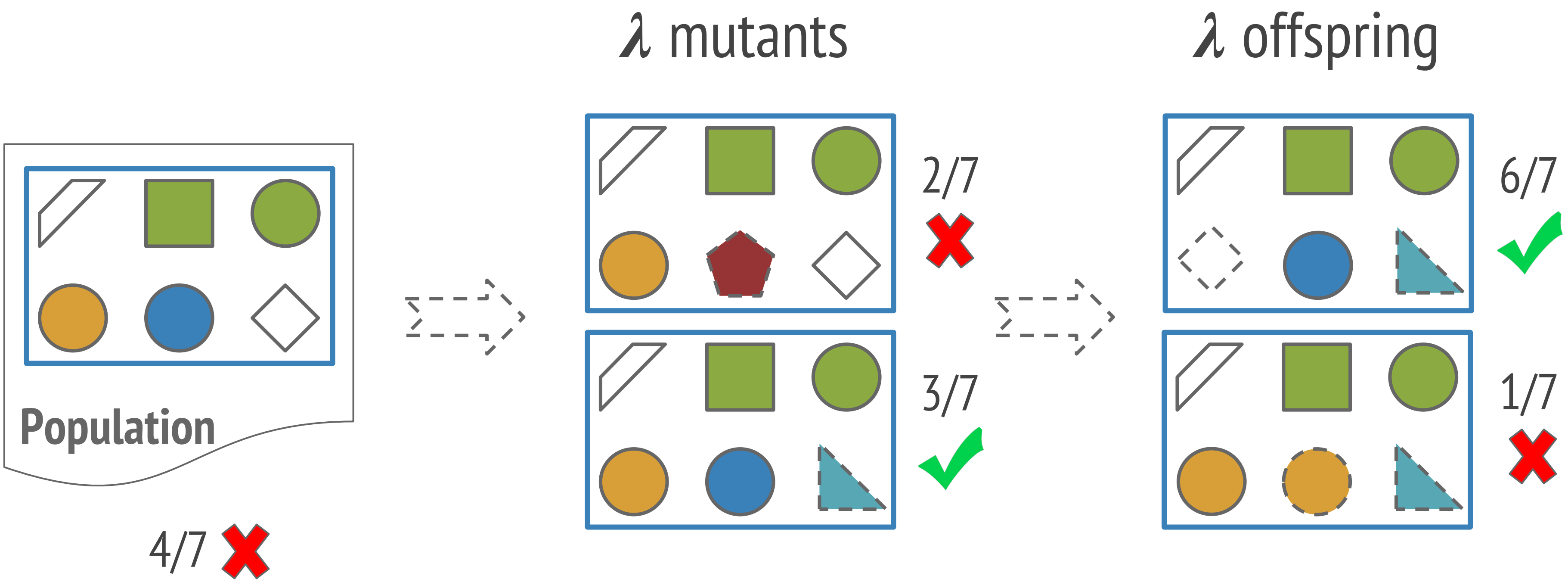
1 + (λ, λ) GA

(4 out of 5 EAs)



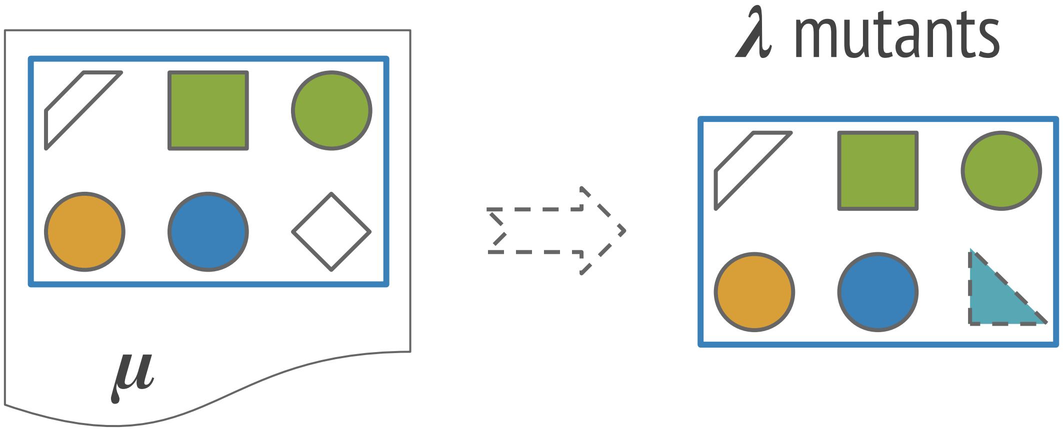
1 + (λ, λ) GA

(4 out of 5 EAs)



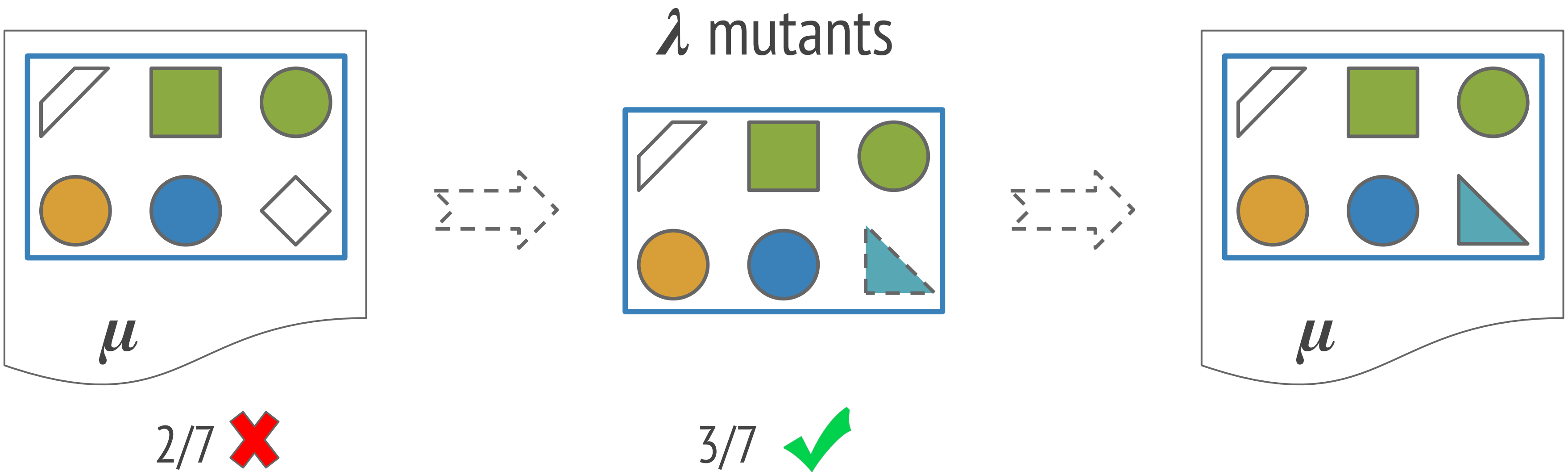
$\mu + \lambda$ Evolutionary Algorithm

(4 out of 5 EAs)



$\mu + \lambda$ Evolutionary Algorithm

(4 out of 5 EAs)



Many-Objective Sorting Algorithm (MOSA)

(5 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Crossover function c_f, Crossover probability c_p, Mutation probability m_p

Output: Archive of optimised individuals A

$p \leftarrow 0$
$N_p \leftarrow$ GenerateRandomPopulation(p_s)
PerformFitnessEvaluation(δ, N_p)
$A \leftarrow \{\}$
while $\neg C$ **do**
    $N_o \leftarrow$ GenerateOffspring(c_f, c_p, m_p, N_p)
    $R_t \leftarrow N_p \cup N_o$
    $r \leftarrow 0$
    $F_r \leftarrow$ PreferenceSorting(R_t)
    $N_{p+1} \leftarrow \{\}$
    while $|N_{p+1}| + |F_r| \leq p_s$ **do**
        CalculateCrowdingDistance(F_r)
        $N_{p+1} \leftarrow N_{p+1} \cup F_r$
        $r \leftarrow r + 1$
    end while
    DistanceCrowdingSort(F_r)
    $N_{p+1} \leftarrow N_{p+1} \cup F_r$ with size $p_s - |N_{p+1}|$
    UpdateArchive(A, N_{p+1})
    $p \leftarrow p + 1$
end while
return A

Many-Objective Sorting Algorithm (MOSA)

(5 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Crossover function c_f, Crossover probability c_p, Mutation probability m_p

Output: Archive of optimised individuals A

$p \leftarrow 0$
$N_p \leftarrow$ GenerateRandomPopulation(p_s)
PerformFitnessEvaluation(δ, N_p)
$A \leftarrow \{\}$
while $\neg C$ **do**
  $N_o \leftarrow$ GenerateOffspring(c_f, c_p, m_p, N_p)
  $R_t \leftarrow N_p \cup N_o$
  $r \leftarrow 0$
  $F_r \leftarrow$ PreferenceSorting(R_t)
  $N_{p+1} \leftarrow \{\}$
  while $|N_{p+1}| + |F_r| \leq p_s$ **do**
    CalculateCrowdingDistance(F_r)
    $N_{p+1} \leftarrow N_{p+1} \cup F_r$
    $r \leftarrow r + 1$
  end while
  DistanceCrowdingSort(F_r)
  $N_{p+1} \leftarrow N_{p+1} \cup F_r$ with size $p_s - |N_{p+1}|$
  UpdateArchive(A, N_{p+1})
  $p \leftarrow p + 1$
end while
return A

Many-Objective Sorting Algorithm (MOSA)

(5 out of 5 EAs)

Input: Stopping condition C, Fitness function δ, Population size p_s, Crossover function c_f, Crossover probability c_p, Mutation probability m_p

Output: Archive of optimised individuals A

$p \leftarrow 0$
$N_p \leftarrow \textsc{GenerateRandomPopulation}(p_s)$
$\textsc{PerformFitnessEvaluation}(\delta, N_p)$
$A \leftarrow \{\}$
while $\neg C$ **do**
    $N_o \leftarrow \textsc{GenerateOffspring}(c_f, c_p, m_p, N_p)$
    $R_t \leftarrow N_p \cup N_o$
    $r \leftarrow 0$
    $F_r \leftarrow \textsc{PreferenceSorting}(R_t)$
    $N_{p+1} \leftarrow \{\}$
    while $|N_{p+1}| + |F_r| \leq p_s$ **do**
        $\textsc{CalculateCrowdingDistance}(F_r)$
        $N_{p+1} \leftarrow N_{p+1} \cup F_r$
        $r \leftarrow r + 1$
    end while
    $\textsc{DistanceCrowdingSort}(F_r)$
    $N_{p+1} \leftarrow N_{p+1} \cup F_r$ with size $p_s - |N_{p+1}|$
    $\textsc{UpdateArchive}(A, N_{p+1})$
    $p \leftarrow p + 1$
end while
return A

Many-Objective Sorting Algorithm (MOSA)

(5 out of 5 EAs)

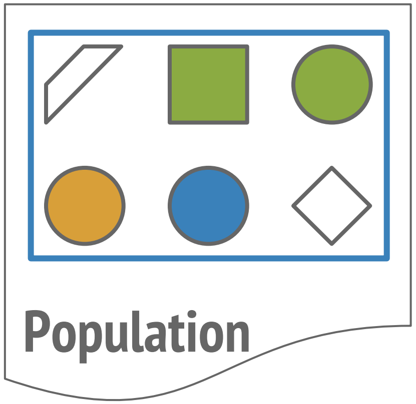
Input: Stopping condition C, Fitness function δ, Population size p_s, Crossover function c_f, Crossover probability c_p, Mutation probability m_p

Output: Archive of optimised individuals A

$p \leftarrow 0$
$N_p \leftarrow \textsc{GenerateRandomPopulation}(p_s)$
$\textsc{PerformFitnessEvaluation}(\delta, N_p)$
$A \leftarrow \{\}$
while $\neg C$ **do**
    $N_o \leftarrow \textsc{GenerateOffspring}(c_f, c_p, m_p, N_p)$
    $R_t \leftarrow N_p \cup N_o$
    $r \leftarrow 0$
    $F_r \leftarrow \textsc{PreferenceSorting}(R_t)$
    $N_{p+1} \leftarrow \{\}$
    while $|N_{p+1}| + |F_r| \leq p_s$ **do**
        $\textsc{CalculateCrowdingDistance}(F_r)$
        $N_{p+1} \leftarrow N_{p+1} \cup F_r$
        $r \leftarrow r + 1$
    end while
    $\textsc{DistanceCrowdingSort}(F_r)$
    $N_{p+1} \leftarrow N_{p+1} \cup F_r$ with size $p_s - |N_{p+1}|$
    $\textsc{UpdateArchive}(A, N_{p+1})$
    $p \leftarrow p + 1$
end while
return A

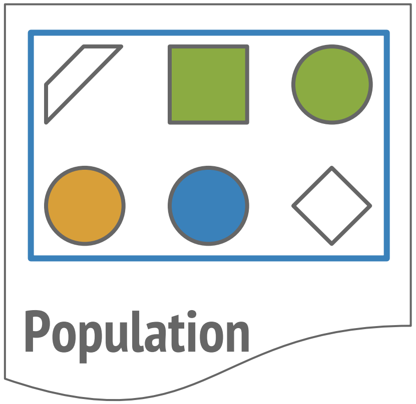
Random Search Test Generation

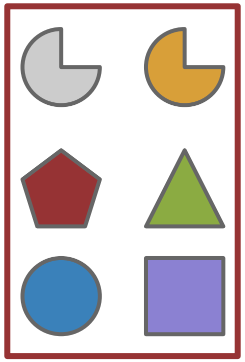
(no selection, crossover, or mutation)



Random Search Test Generation

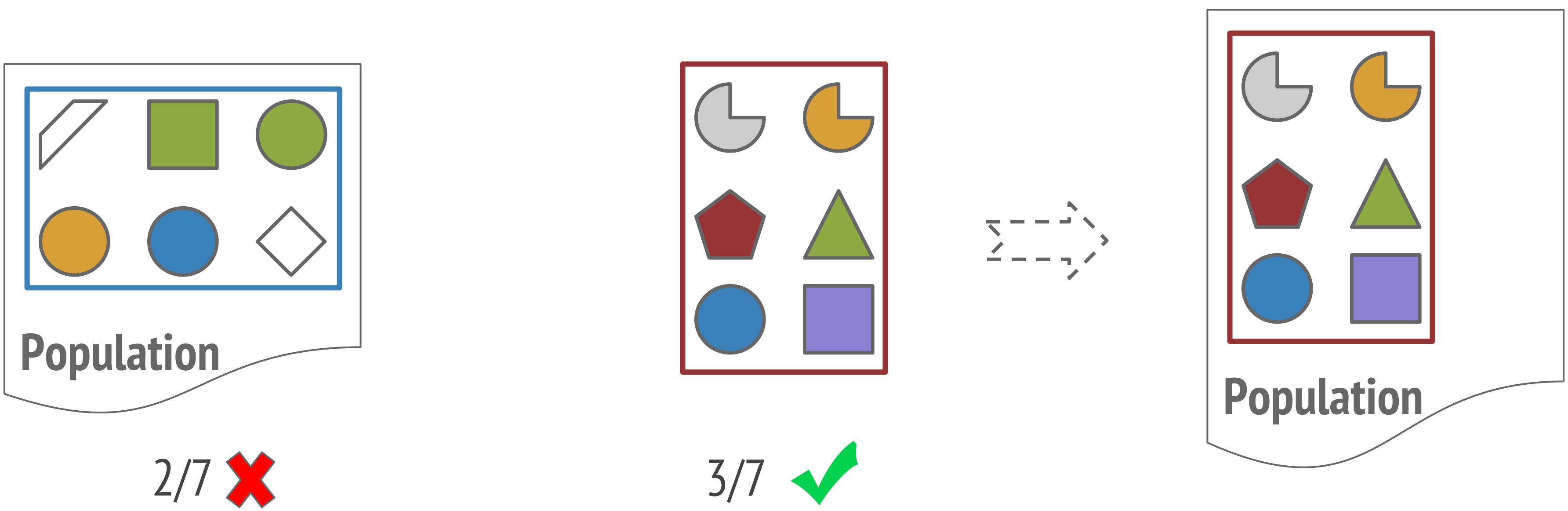
(no selection, crossover, or mutation)



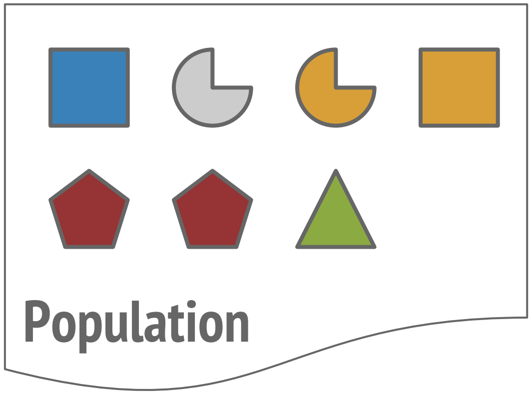


Random Search Test Generation

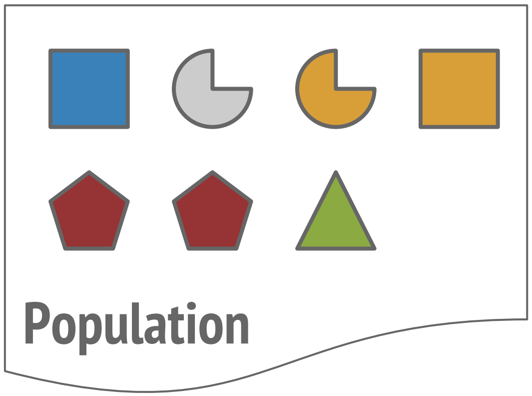
(no selection, crossover, or mutation)

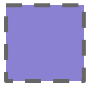


Random Test Generation

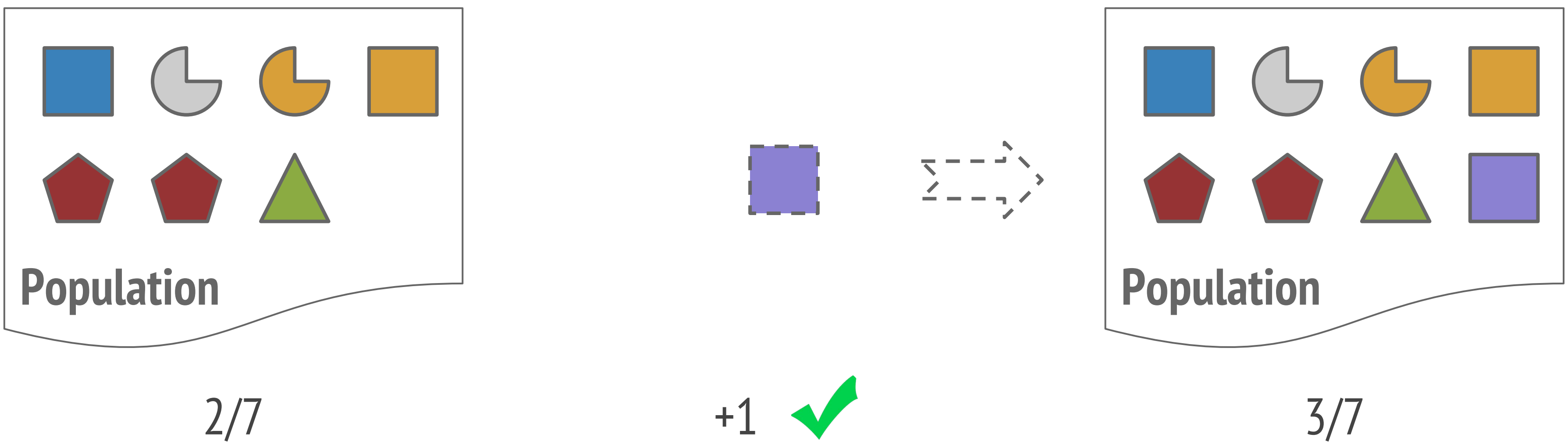


Random Test Generation





Random Test Generation



Enough of theory! Show me some results.

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average
EVOSUITE Standard GA, Steady State GA, MOSA, DynaMOSA, Random-search, and Random-testing	1 + (λ, λ) GA
	μ + λ EA

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average
EVOSUITE Standard GA, Steady State GA, MOSA, DynaMOSA, Random-search, and Random-testing	1 + (λ, λ) GA
	μ + λ EA

Tuning experiment / Larger study

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average
EVOSUITE Standard GA, Steady State GA, MOSA, DynaMOSA, Random-search, and Random-testing	1 + (λ, λ) GA
	μ + λ EA

Tuning experiment / Larger study

34 / 312

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average
EVOSUITE Standard GA, Steady State GA, MOSA, DynaMOSA, Random-search, and Random-testing	1 + (λ, λ) GA
	μ + λ EA

Tuning experiment / Larger study

34 / 312

Single & Multiple-criteria

60s & 600s

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Experimental Setup

CUTs from DynaMOSA study [17]	346 Java classes from 117 open-source projects
	1,109 statements and 259 branches on average
EVOSUITE Standard GA, Steady State GA, MOSA, DynaMOSA, Random-search, and Random-testing	1 + (λ, λ) GA
	μ + λ EA

Tuning experiment / Larger study	34 / 312	Single & Multiple-criteria	60s & 600s	30 repetitions & 10 repetitions

[17] Panichella, Kifetew, Tonella, Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, TSE 2017

Parameter Tuning

Parameter Tuning*

	Population
Standard GA, Monotonic GA, Steady-State GA, MOSA, DynaMOSA	10, 25, 50, 100
1 + (λ, λ) GA	1, 8+, 25, 50
μ + λ EA	1, 7±, 25, 50

* Random-based approaches do not require any tuning
+[5] Doerr, Doerr, Ebel, From black-box complexity to designing new genetic algorithms, Theoretical Computer Science 2015
± [13] Jansen, De Jong, Wegener, On the choice of the offspring population size in evolutionary algorithms, Evolutionary Computation 2005

Best Parameters

What population size allows EA A to achieve the highest coverage of class C ?

Population	Coverages										Avg.
10	73	100	13	21	43	6	98	62	12	100	52.8
25	80	79	73	62	24	81	46	81	84	53	66.3
50	54	78	35	26	20	7	90	59	25	4	39.8

Best Parameters

What population size allows EA A to achieve the highest coverage of class C ?

Population	Coverages										Avg.
10	73	100	13	21	43	6	98	62	12	100	52.8
25	80	79	73	62	24	81	46	81	84	53	66.3
50	54	78	35	26	20	7	90	59	25	4	39.8

$\hat{A}_{10,25} = 0.39$ *p-value* = 0.44

$\hat{A}_{10,50} = 0.60$ *p-value* = 0.29

$\hat{A}_{25,50} = \mathbf{0.76}$ *p-value* = **0.04**

Best Parameters

	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

Best Parameters

	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	$\boxed{10}$	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

Best Parameters

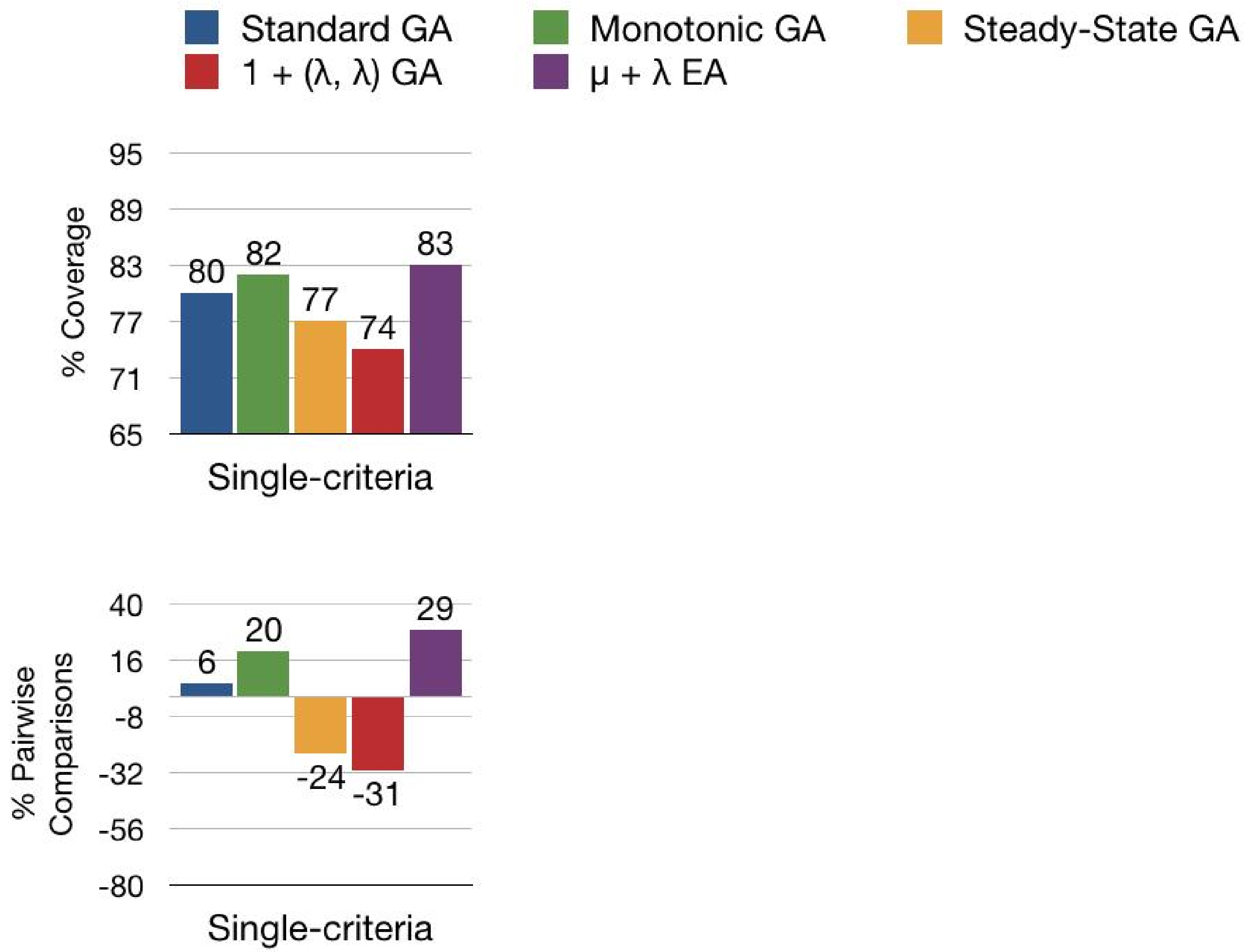
	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	$\boxed{10}$	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	$\boxed{100}$	50	25	10
DynaMOSA	25	25	---	---

Best Parameters

	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

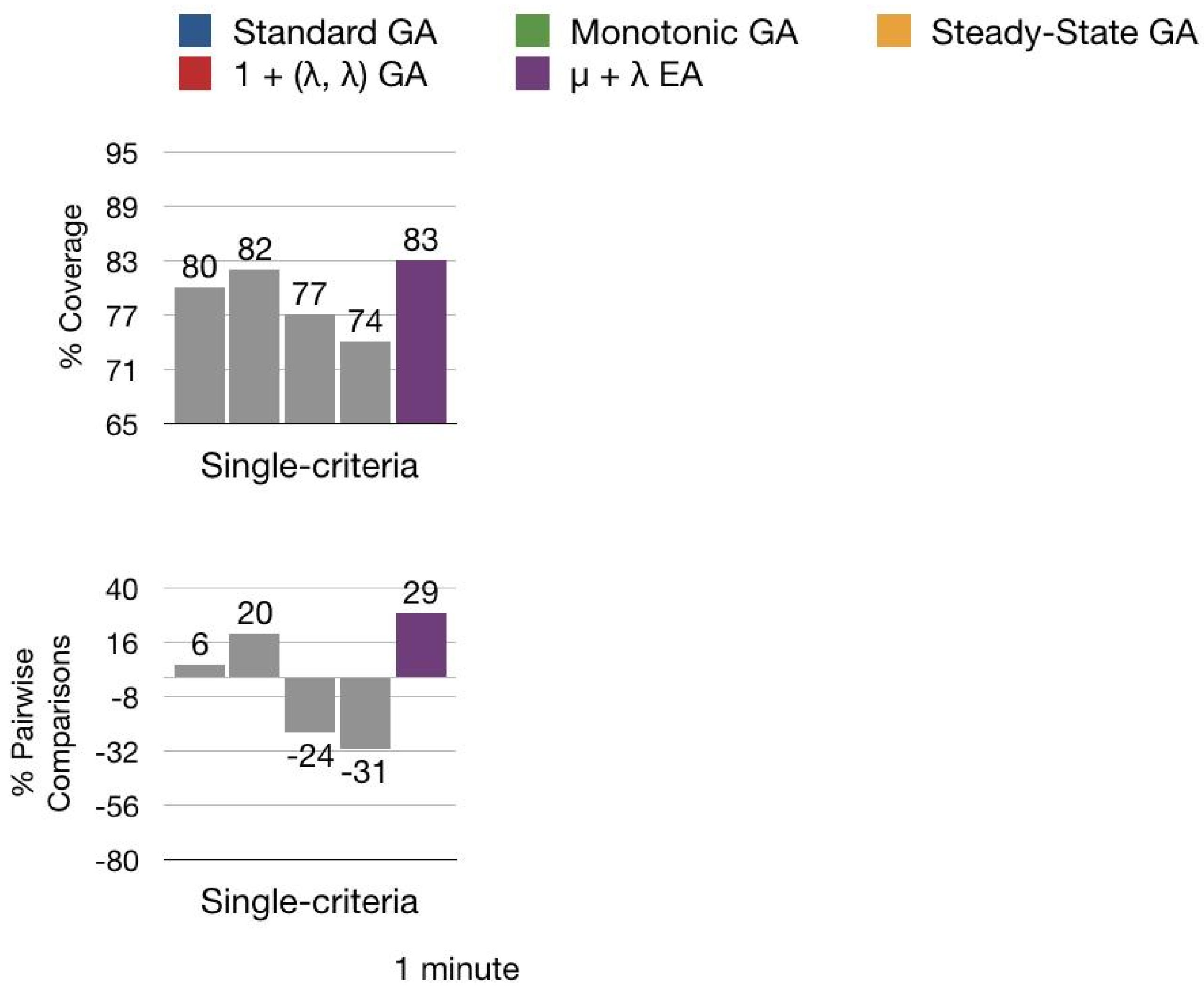
RQ1 - Best Evolutionary Algorithm

RQ1 - Best Evolutionary Algorithm

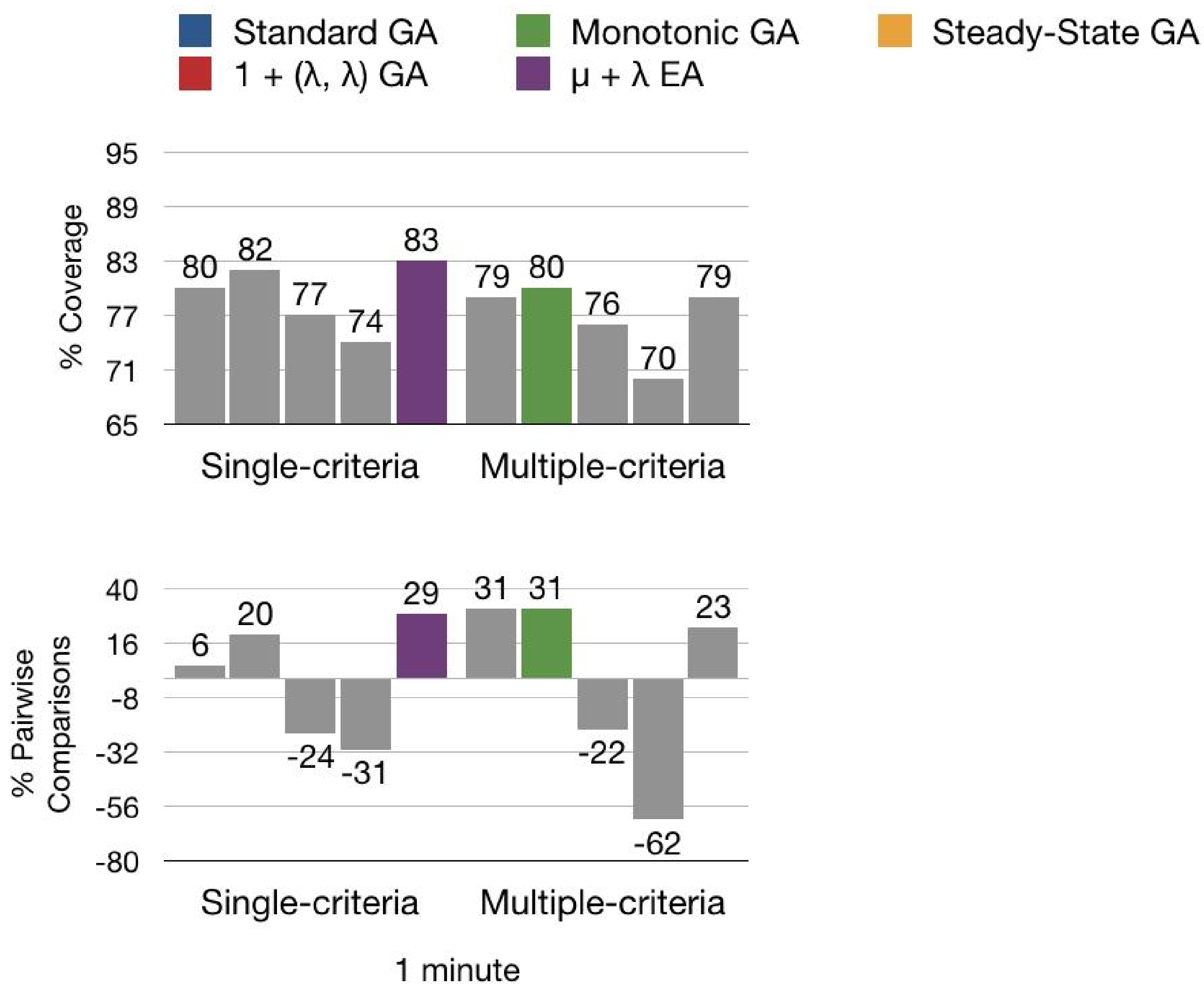


1 minute

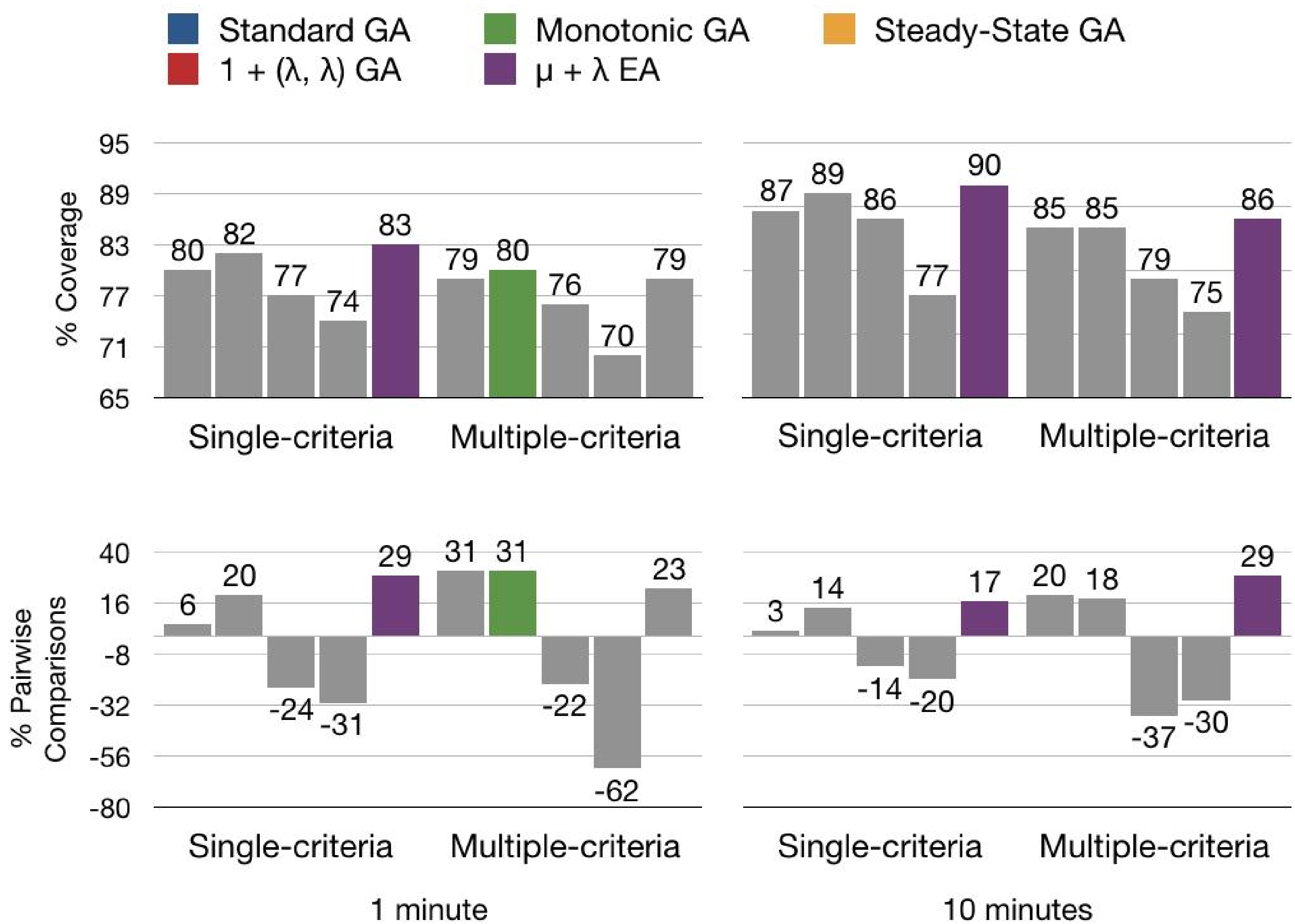
RQ1 - Best Evolutionary Algorithm



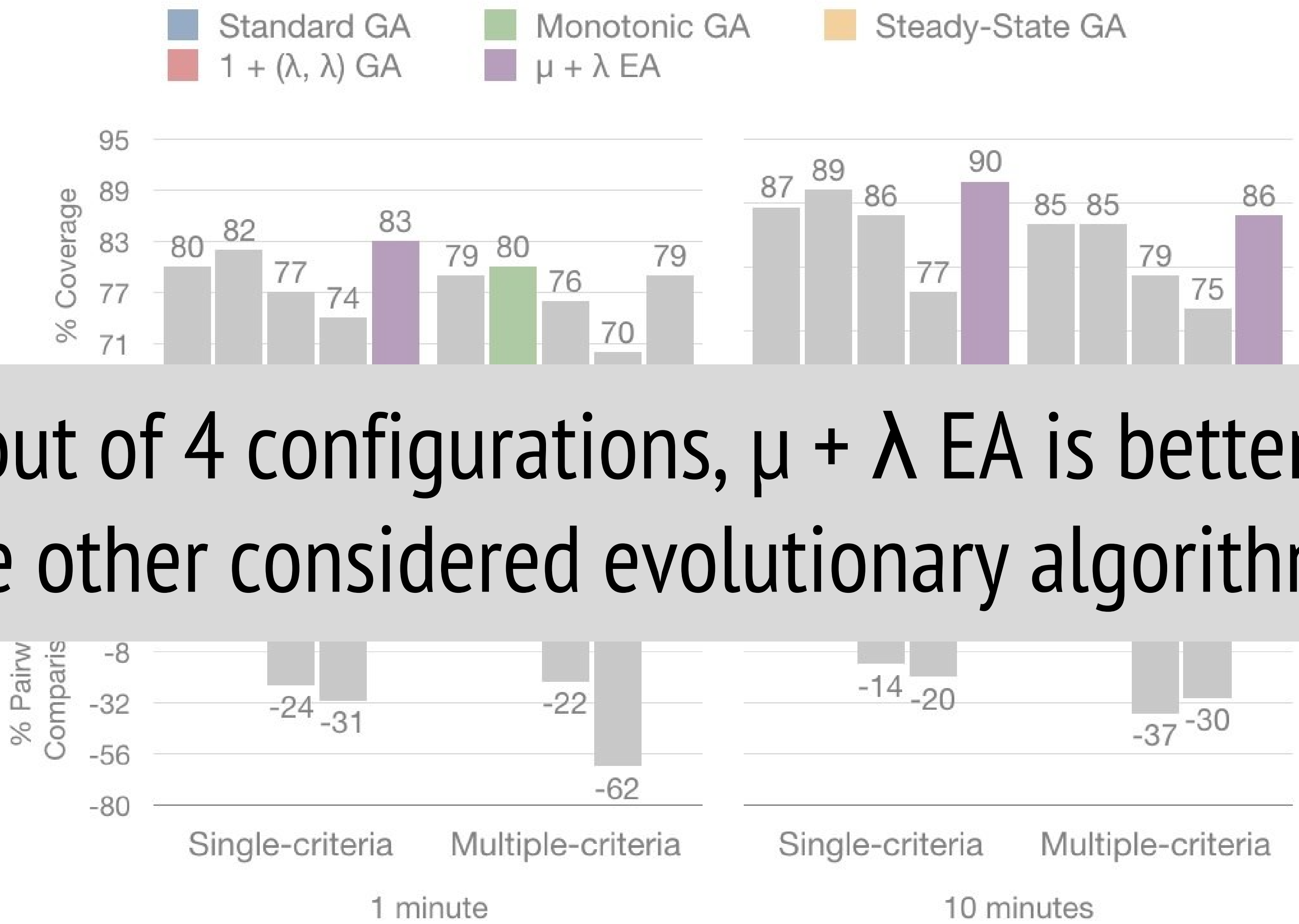
RQ1 - Best Evolutionary Algorithm



RQ1 - Best Evolutionary Algorithm



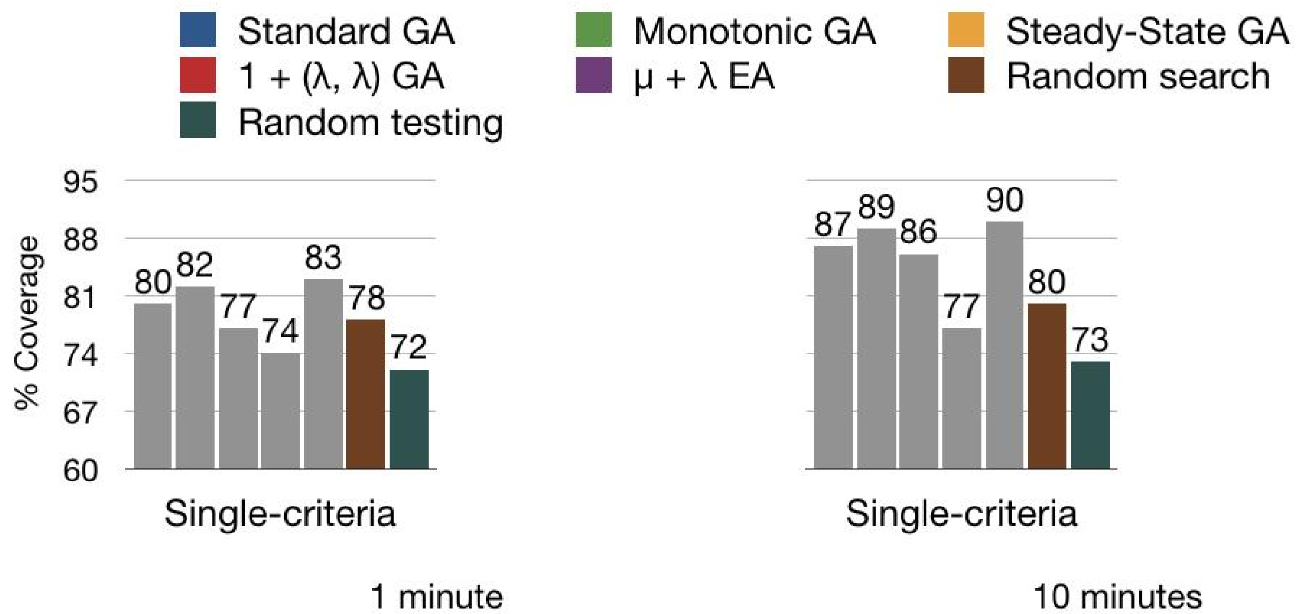
RQ1 - Best Evolutionary Algorithm



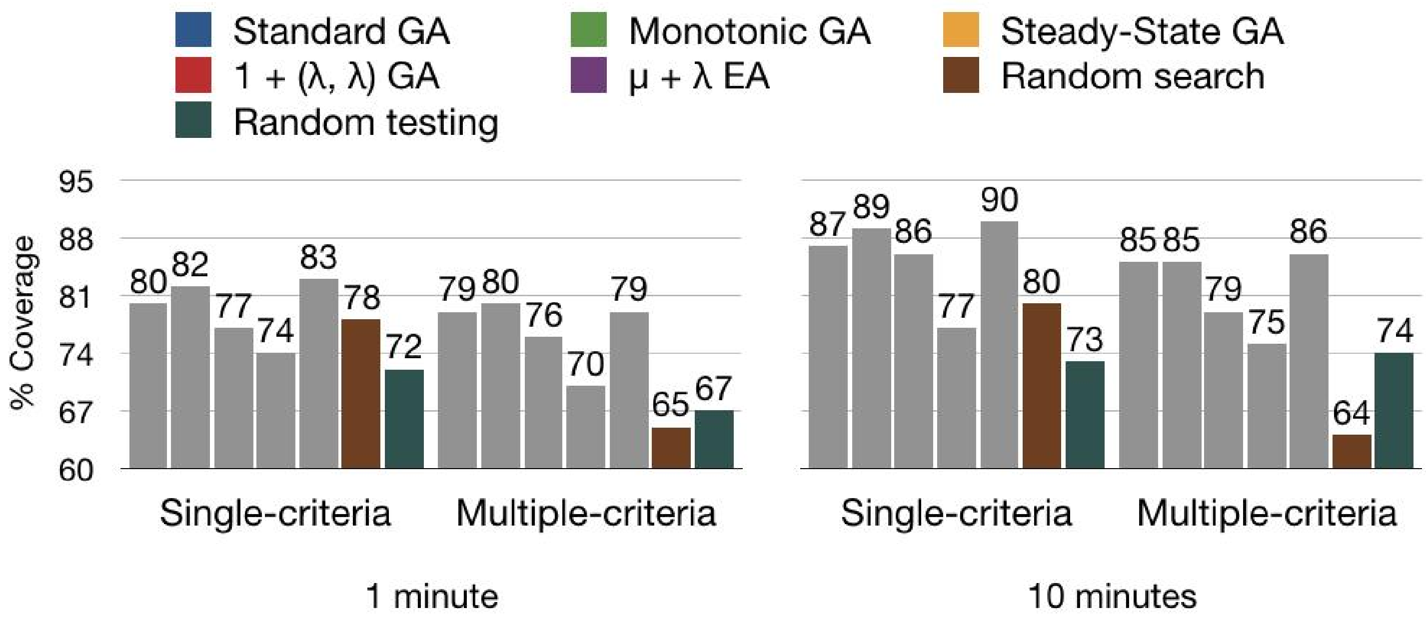
In 3 out of 4 configurations, μ + λ EA is better than the other considered evolutionary algorithms.

RQ2 - Evolutionary Search vs. Random

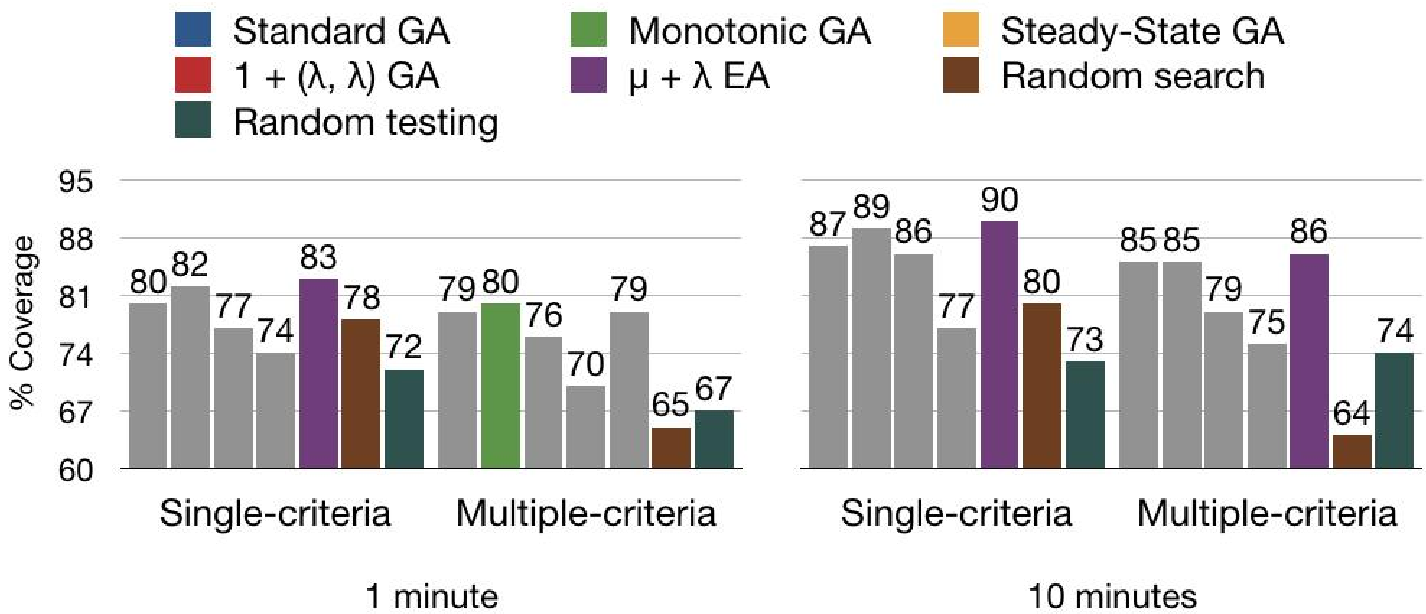
RQ2 - Evolutionary Search vs. Random



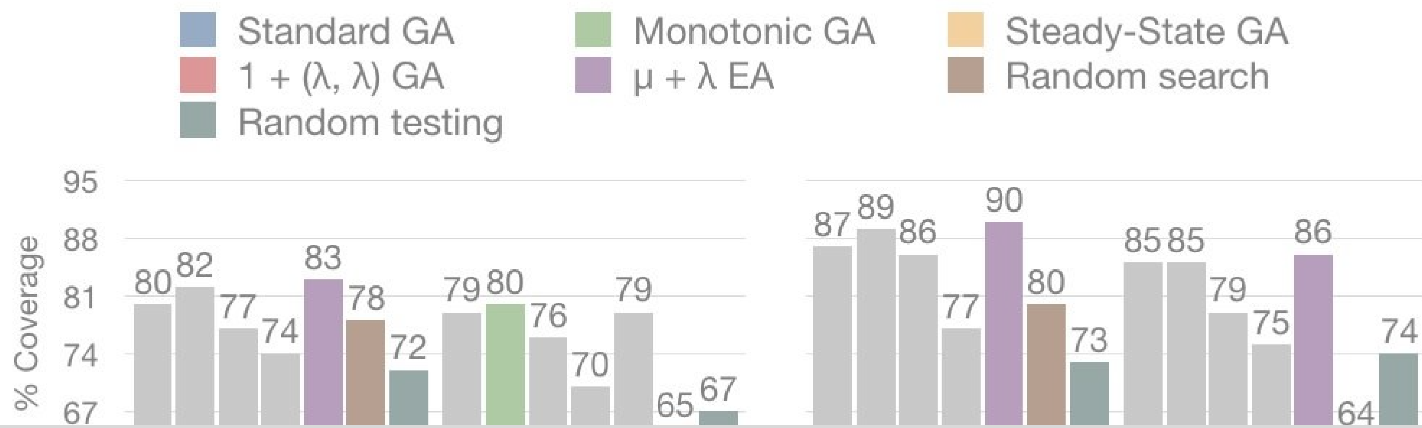
RQ2 - Evolutionary Search vs. Random



RQ2 - Evolutionary Search vs. Random



RQ2 - Evolutionary Search vs. Random

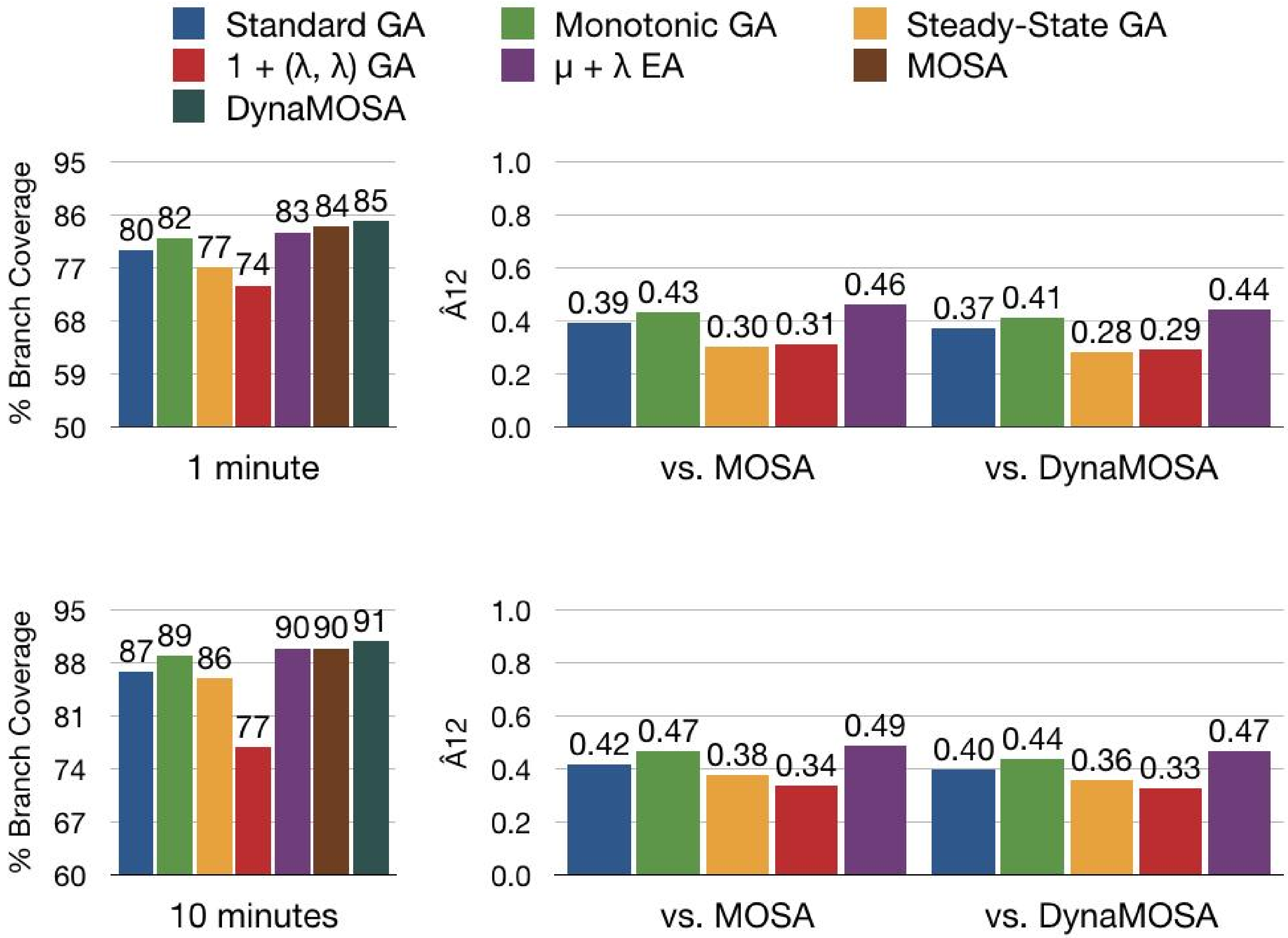


Evolutionary algorithms (in particular μ + λ EA) perform better than random search and testing.

RQ3 - Whole test suites vs. MOSA

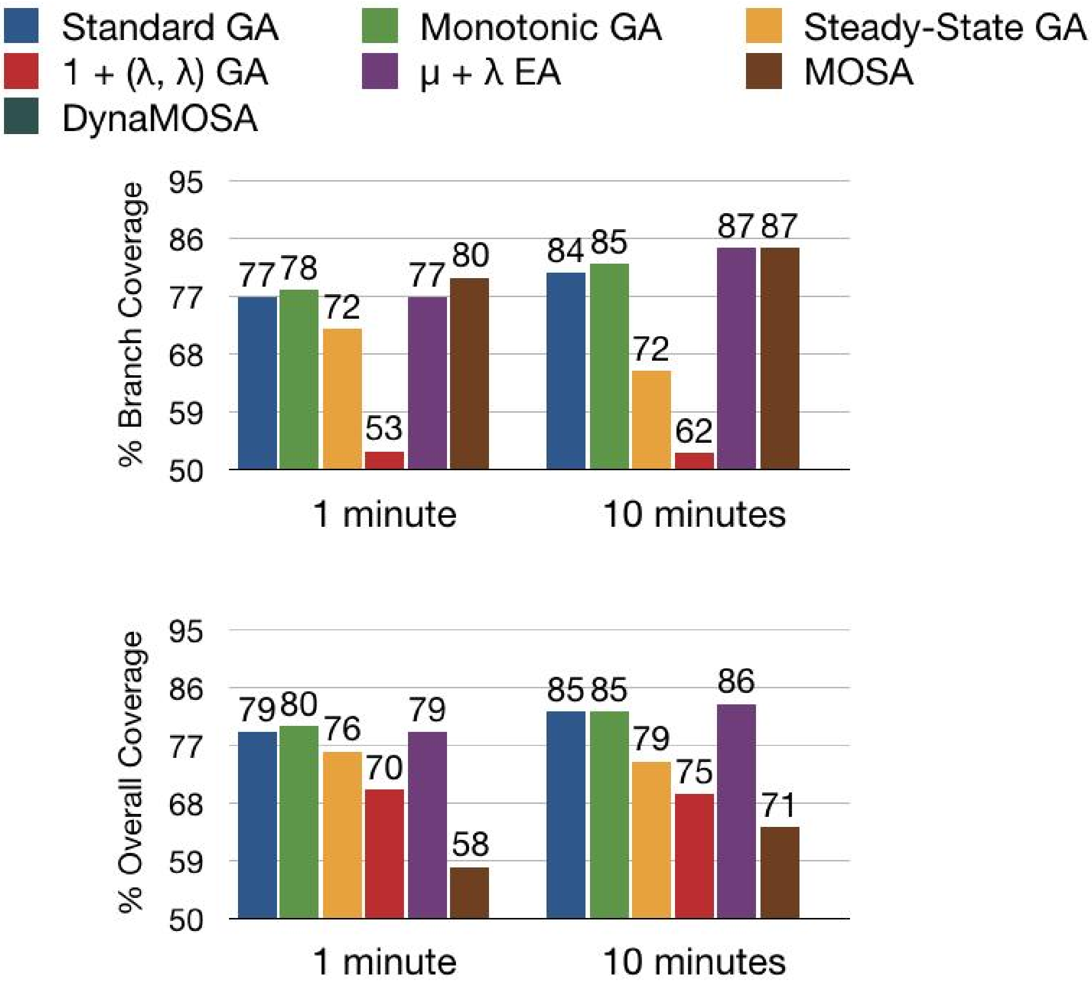
RQ3 - Whole test suites vs. MOSA

(single-criteria)



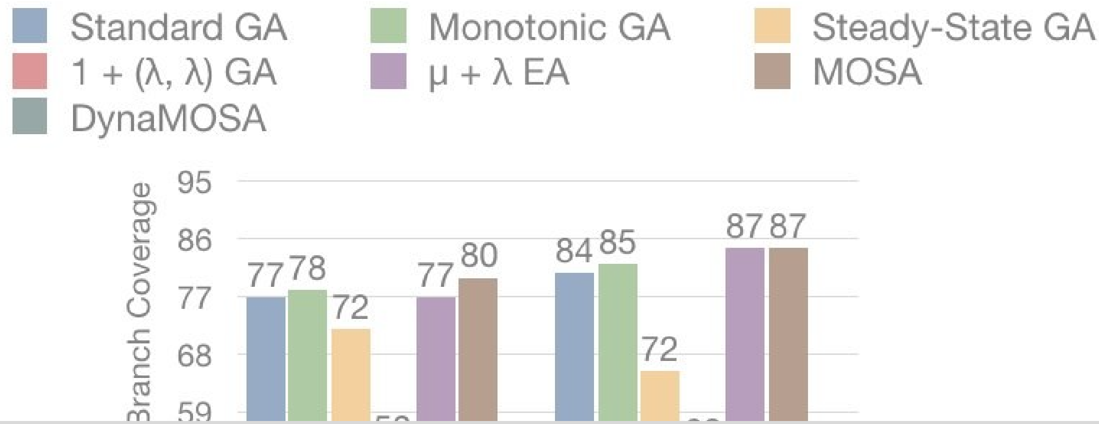
RQ3 - Whole test suites vs. MOSA

(multiple-criteria)

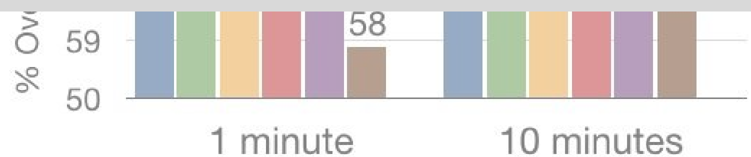


RQ3 - Whole test suites vs. MOSA

(multiple-criteria)



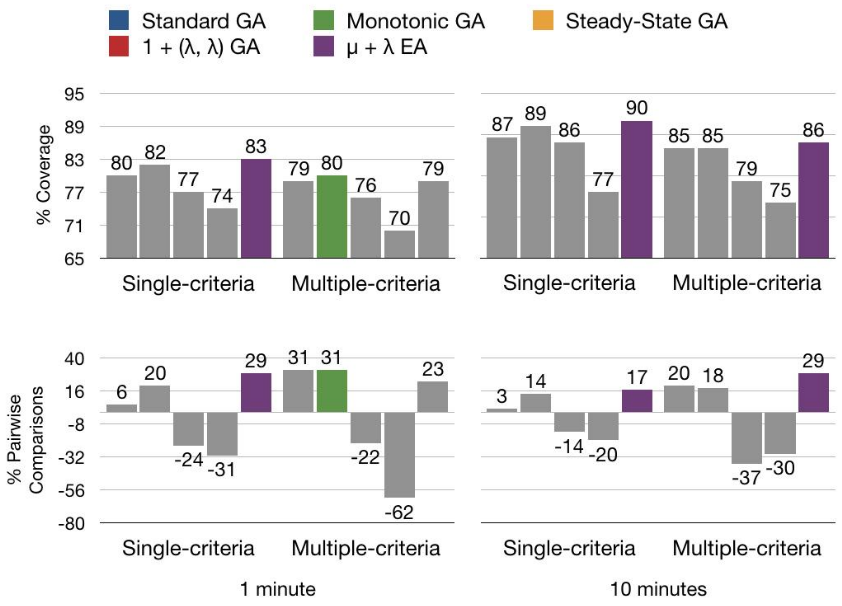
MOSA improves over EAs for individual criteria; for multiple-criteria it achieves higher branch coverage even though overall coverage is lower.



	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

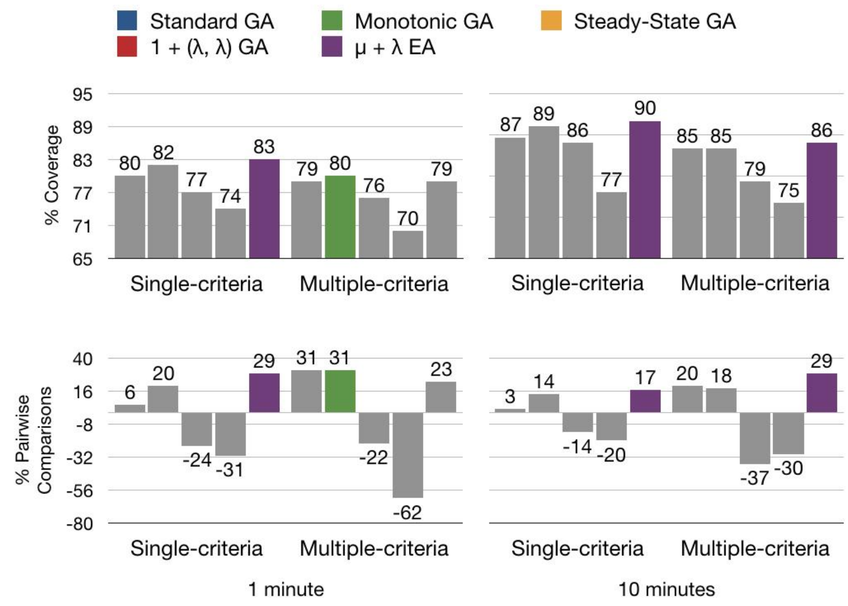
The choice of algorithm can have a substantial influence on the performance of test suite optimisation, hence tuning is important. While EvoSuite provides tuned default values, these values may not be optimal for different flavours of evolutionary algorithms.

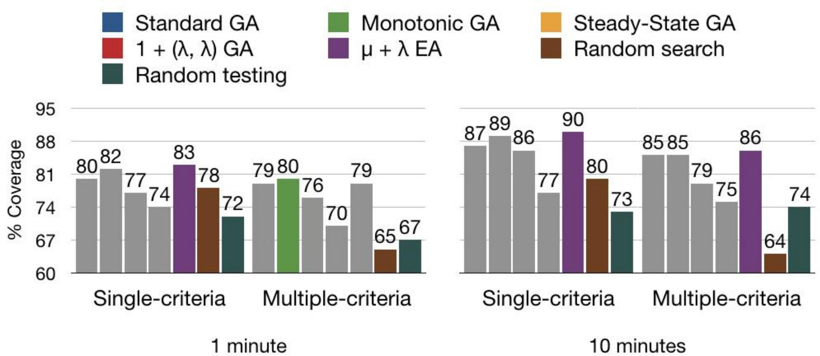
	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---



EvoSuite's default algorithm, a Monotonic GA, is an appropriate choice for EvoSuite's default configuration (1 minute search budget, multiple criteria). However, for other search budgets and optimisation goals, other algorithms such as a μ + λ EA may be a better choice.

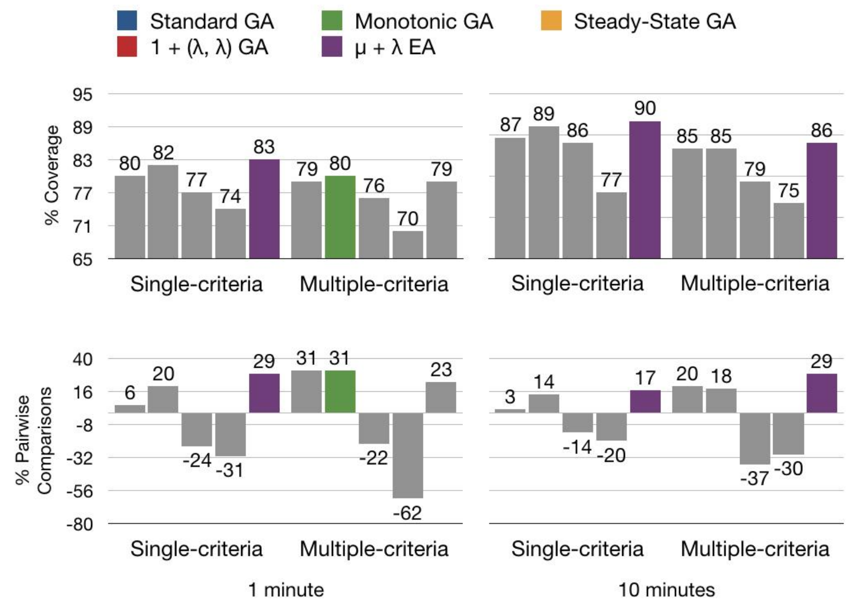
	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

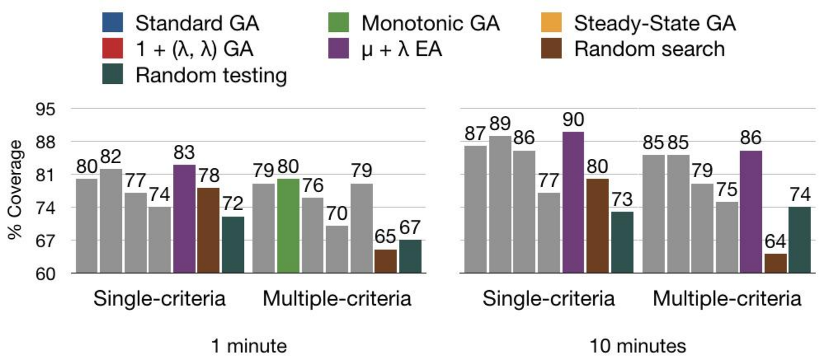


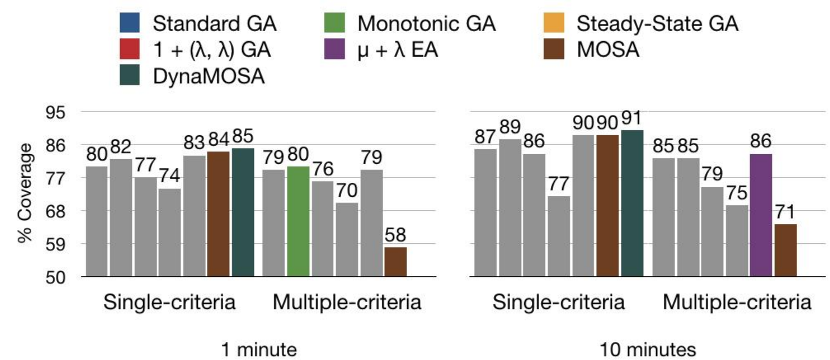


Our study shows that on complex classes evolutionary algorithms are superior to random testing and random search.

	Single-criteria		Multiple-criteria	
Algorithm	**1 minute**	**10 minutes**	**1 minute**	**10 minutes**
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---

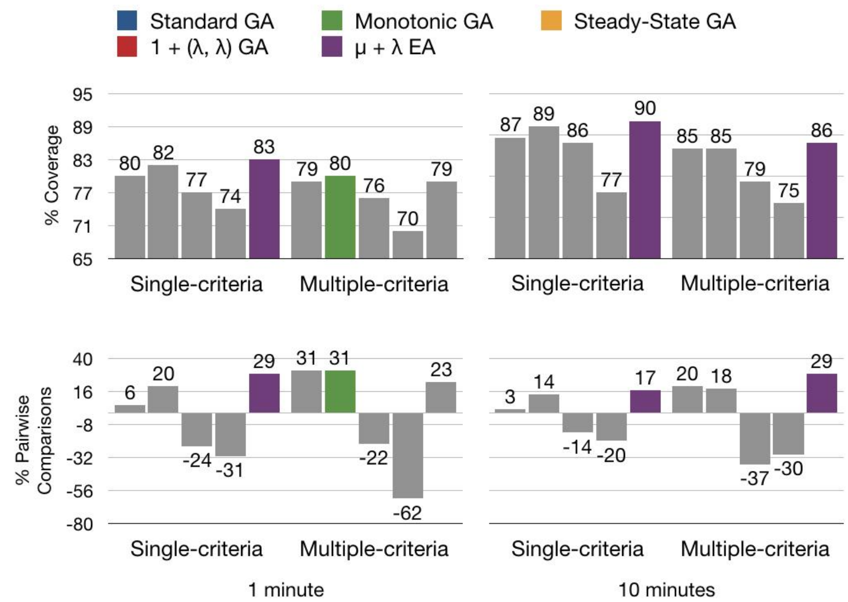


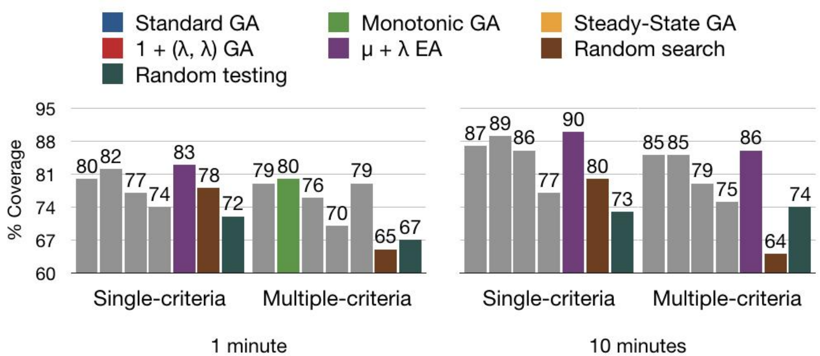


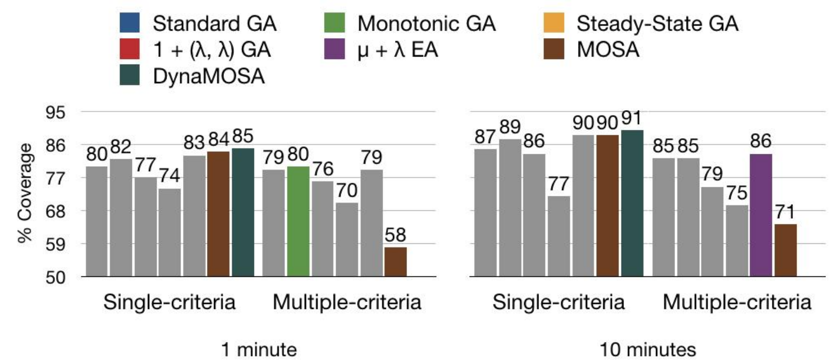


The Many Objective Sorting Algorithm (MOSA) is superior to whole test suite optimisation for single criteria. (It would be desirable to add support to all coverage criteria in DynaMOSA)

	Single-criteria		Multiple-criteria	
Algorithm	1 minute	10 minutes	1 minute	10 minutes
Standard GA	10	100	100	25
Monotonic GA	25	100	100	25
Steady-State GA	100	10	100	25
1 + (λ, λ) GA	50	50	50	8
μ + λ EA	1+7	50+50	1+7	1+1
MOSA	100	50	25	10
DynaMOSA	25	25	---	---







http://www.evosuite.org/