Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Department of Computer and Software Engineering
École Polytechnique de Montréal, Québec, Canada

PhD Dissertation Defense
December 16^{th}, 2014, Montréal QC

Pattern Trace Identification, Detection, and Enhancement in Java
SOftware Cost-effective Change and Evolution Research Lab

Motivation

Every year inadequate infrastructure software costs the U.S. economy around $60 billion.

Motivation

Every year inadequate infrastructure software costs the U.S. economy around $60 billion.

Between 1985 and 1987 : the radiotherapy machine Therac-25 sent to patients a X-ray dose 100 times greater than expected.

Motivation

Every year inadequate infrastructure software costs the U.S. economy around $60 billion.

Between 1985 and 1987 : the radiotherapy machine Therac-25 sent to patients a X-ray dose 100 times greater than expected.

- At least five deaths;
- Several other patients were severely affected by radiation.

Motivation

Every year inadequate infrastructure software costs the U.S. economy around $60 billion.

Between 1985 and 1987 : the radiotherapy machine Therac-25 sent to patients a X-ray dose 100 times greater than expected.

- At least five deaths;
- Several other patients were severely affected by radiation.

Only a high quality software can reduce errors costs.

Software testing

Expensive

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing!

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing!

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing! is insufficient

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing! is insufficient
- Selecting and generating an adequate subset of test data!

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing! is insufficient
- Selecting and generating an adequate subset of test data! is hard

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing! is insufficient
- Selecting and generating an adequate subset of test data! is hard

Challenge

Automation of test-data generation

Software testing

Expensive

- Costs more than 50% of the budget of critical software;

Difficult

- Exhaustive testing! is not possible
- Random testing! is insufficient
- Selecting and generating an adequate subset of test data! is hard

Challenge

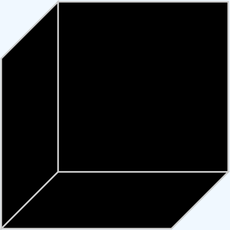
Automation of test-data generation can significantly reduce the cost of software

Software testing

Strategies

Functional Criteria

- Focus on the specification;



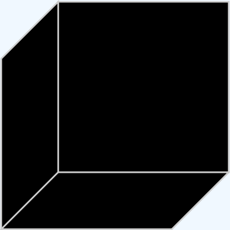
Black-box Testing

Software testing

Strategies

Functional Criteria

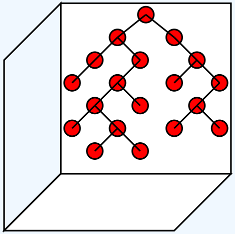
- Focus on the specification;



Black-box Testing

Structural Criteria

- Focus on the internal structure;



White-box Testing

White-box Testing(Goal)

```
int foo(int x,int y, float z, String s1,String s2) {
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();
    y= x<<y;
    x=y+x/y;
    String s=s1+s2;
    if((s.equals("OK") && (x>0) && s.length()>x)
        return y/s1.length();
    return 0;
}
```

White-box Testing(Goal)

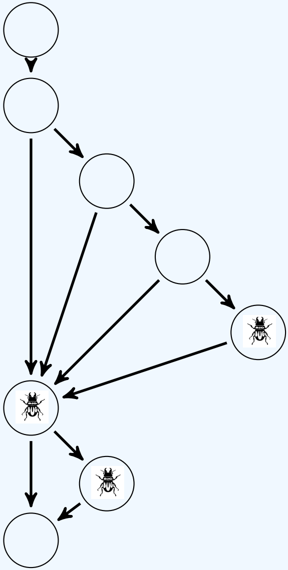
```
1   int foo(int x,int y, float z, String s1,String s2) {
2       if(y==z)
3           if(y>0)
4               if(x==10)
5                   return s1.length()+s2.length();
6       y= x<<y;
7       x=y+x/y;
8       String s=s1+s2;
9       if((s.equals("OK") && (x>0) && s.length()>x)
10          return y/s1.length();
11      return 0;
12  }
```

Unhandled Exceptions

- Line 5: undefined behavior if $s1$ or $s2$ is null.
- Line 6: undefined behavior if y is negative or >31;
- Line 7: undefined behavior if $y = 0$;
- Line 8: undefined behavior if $s1$ or $s2$ is null;
- Line 10: undefined behavior if $s1 =$ “” or is null.

White-box Testing(Requirements)

Control Flow Graph of foo

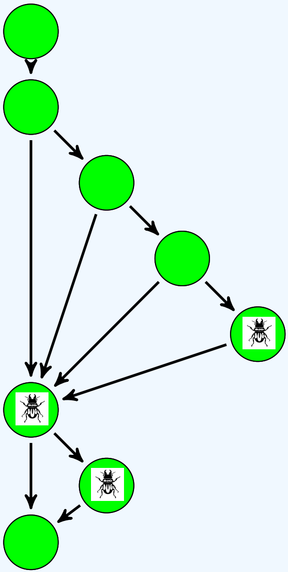


Coverage Criteria

- All-statement;
- All-branch;
- All-paths.

White-box Testing(Requirements)

Control Flow Graph of foo

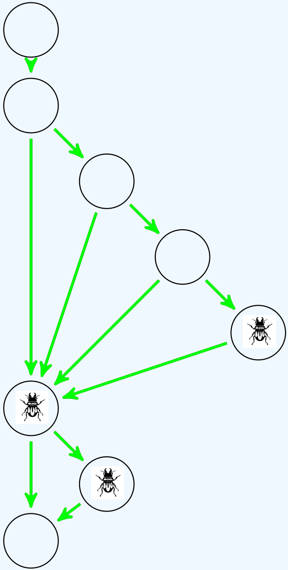


Coverage Criteria

- All-statement;
- All-branch;
- All-paths.

White-box Testing(Requirements)

Control Flow Graph of foo

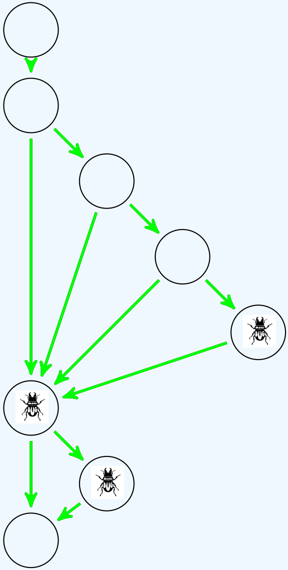


Coverage Criteria

- All-statement;
- All-branch;
- All-paths.

White-box Testing(Requirements)

Control Flow Graph of foo

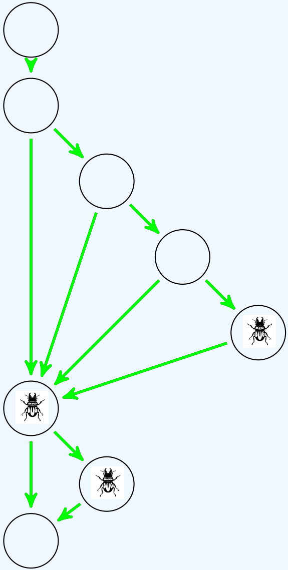


Coverage Criteria

- All-statement;
- All-branch;
- All-paths.

White-box Testing(Requirements)

Control Flow Graph of foo



Coverage Criteria

- All-statement;
- All-branch;
- All-paths.

Challenge

- Finding the set of vectors (x,y,z,s1,s2) that satisfy a given coverage criterion.

Test-Data Generation for Unit Class Testing

Difficulties

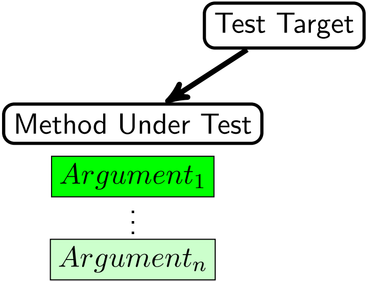
- **Encapsulation**: the state of a class is accessible only through some public methods.
- **Abstraction**: some classes are not directly instantiable;
- **Anonymous Classes**: some classes are not visible outside of the method where they are declared;

Test-Data Format of Object-Oriented Programming (OOP) is complex.

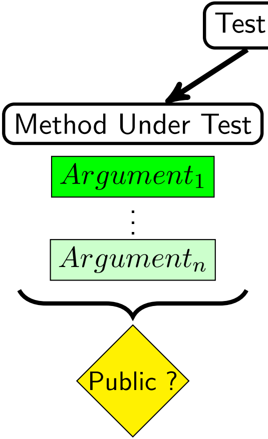
Test-Data Generation for Unit Class Testing

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
     foo(a,str);...
}
private int foo(int x, String s1){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();
    y= x<<y;
    x=y+x/y;
    String s=s1+s2;
    if((s.equals("OK") && (x>0) && s.length()>x)
        return y/s1.length();
    return 0;
}
}
```

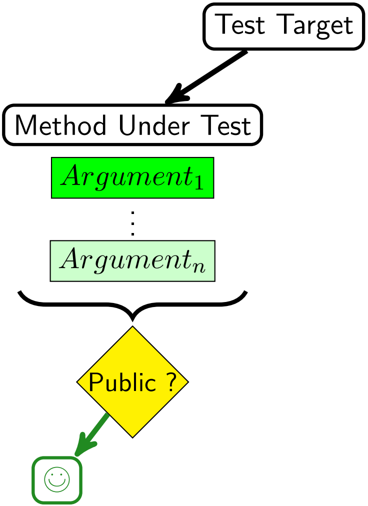
Test-Data Format of OOP



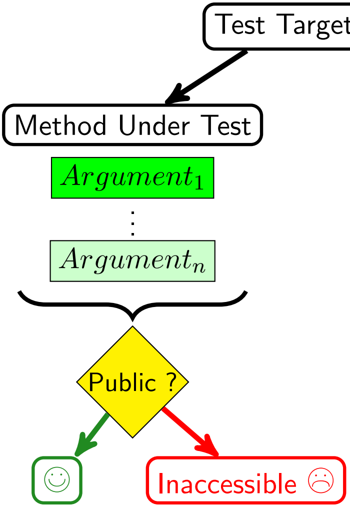
Test-Data Format of OOP



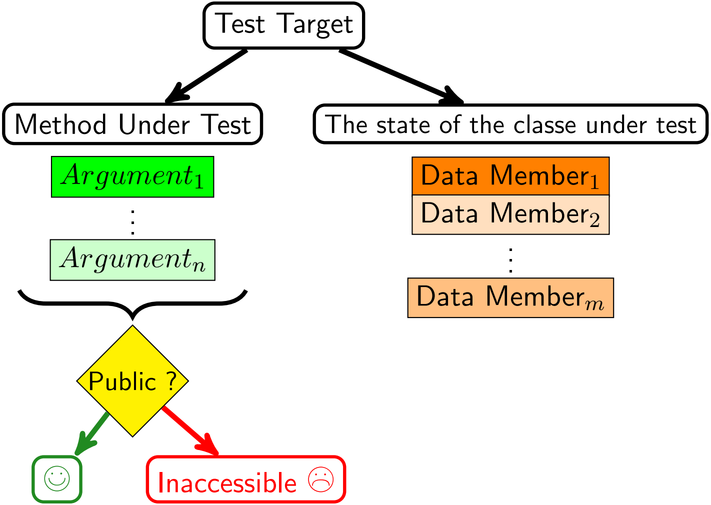
Test-Data Format of OOP



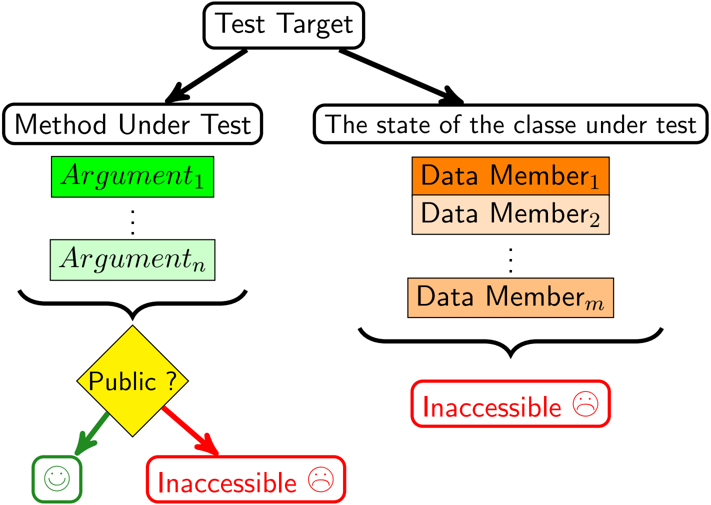
Test-Data Format of OOP



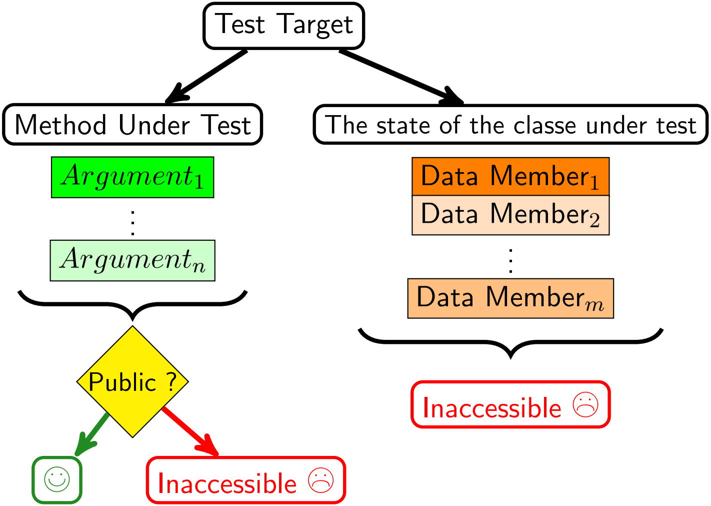
Test-Data Format of OOP



Test-Data Format of OOP



Test-Data Format of OOP



Public methods to reach inaccessible parts.

Test-Data Format of OOP

Challenge

1. Finding an instance of the class under test;
2. Finding a sequence of method calls;
3. Finding an instance of each required argument.

- The number of sequence of method calls may be unlimited;
- The number of different instances of a given class may be large.

Test-Data Format of OOP

Challenge

1. Finding an instance of the class under test;
2. Finding a sequence of method calls;
3. Finding an instance of each required argument.

- The number of sequence of method calls may be unlimited;
- The number of different instances of a given class may be large.

Pb1: The search space is a problem

Automation

Search Based (SB-STDG)

Source Code

↓

Instrumentation

↓

Instrumented Source Code

↓

Meta-heuristic and Fitness Function

↓

Test Data

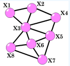
Constraint Based (CB-STDG)

Source Code

↓

Symbolic Execution

↓



Constraint Satisfaction Problem (CSP)

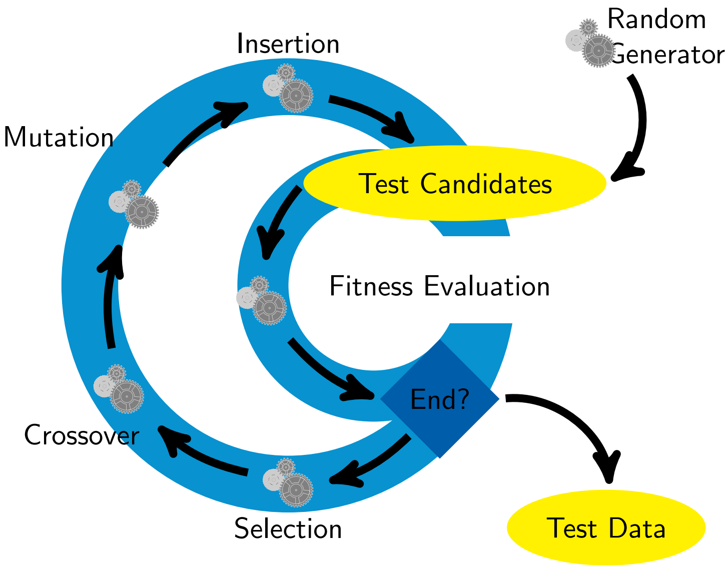
↓

Solver

↓

Test Data

Search Based(Genetic Algorithms)



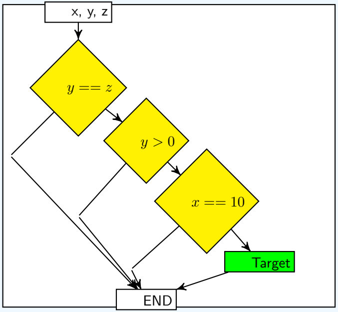
Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Search Based

```
int foo(int x,int y,float z, String s1, String s2){
  if(y==z)
    if(y>0)
      if(x==10)
        return s1.length()+s2.length();//Target
  ...
}
```

Search Based

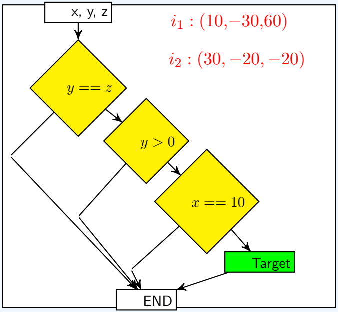
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
        ...
    }
```



Abdelilah Sakti
Supervised by
Gilles Pesant and
Yann-Gaël Guéhéneuc

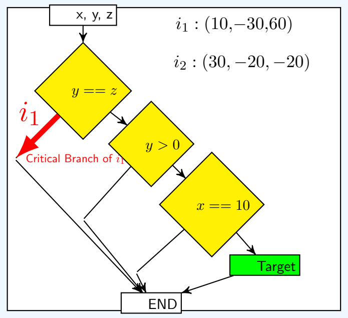
Search Based

```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
    ...
}
```



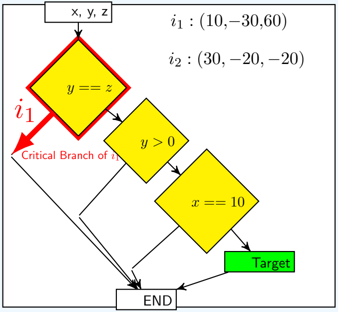
Search Based

```
int foo(int x,int y,float z, String ↩
        s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
              return s1.length()+s2.↩
                     length();//Target
    ...
  }
```



Search Based

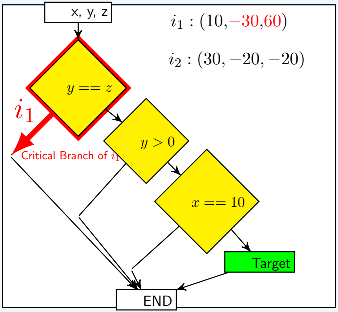
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
        ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	3	
i_2		

Search Based

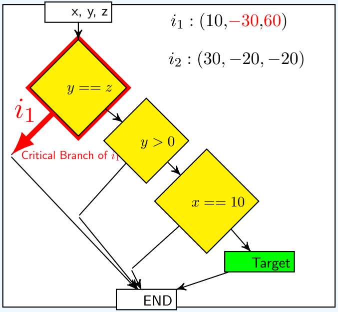
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
    ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91}$	
i_2		

Search Based

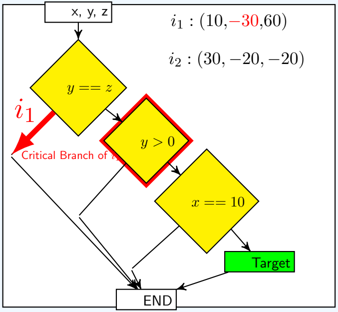
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
    ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91}$	$\frac{90}{91}$
i_2		

Search Based

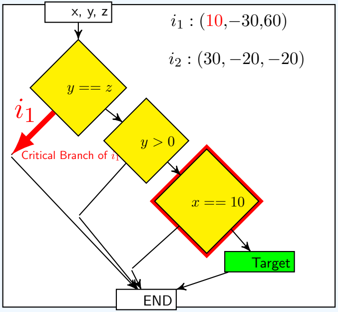
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
    ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91}$	$\frac{90}{91} + \frac{31}{32}$
i_2		

Search Based

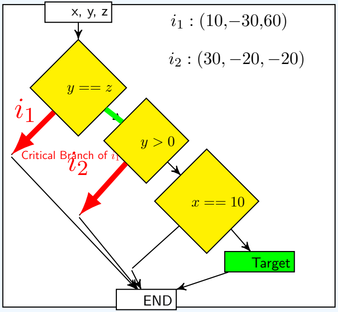
```
int foo(int x,int y,float z, String s1, String s2){
   if(y==z)
      if(y>0)
         if(x==10)
            return s1.length()+s2.length();//Target
   ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91}$	$\frac{90}{91}+\frac{31}{32}+0$
i_2		

Search Based

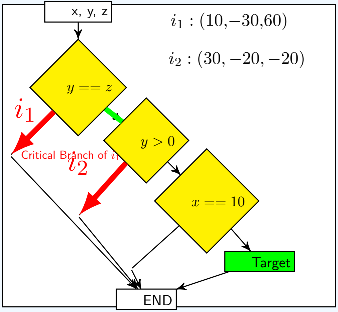
```
int foo(int x,int y,float z, String s1, String s2){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();//Target
    ...
}
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91} = 3.9890$	$\frac{90}{91} + \frac{31}{32} + 0 = 1.9577$
i_2	$2 + \frac{21}{22} = 2.9545$	$0 + \frac{21}{22} + \frac{20}{21} = 1.9068$

Search Based

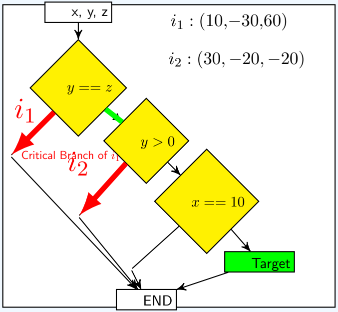
```
int foo(int x,int y,float z, String s1, String s2){
   if(y==z)
      if(y>0)
         if(x==10)
            return s1.length()+s2.length();//Target
      ...
   }
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91} = 3.9890$	$\frac{90}{91} + \frac{31}{32} + 0 = 1.9577$
i_2	$2 + \frac{21}{22} = 2.9545$	$0 + \frac{21}{22} + \frac{20}{21} = 1.9068$

Search Based

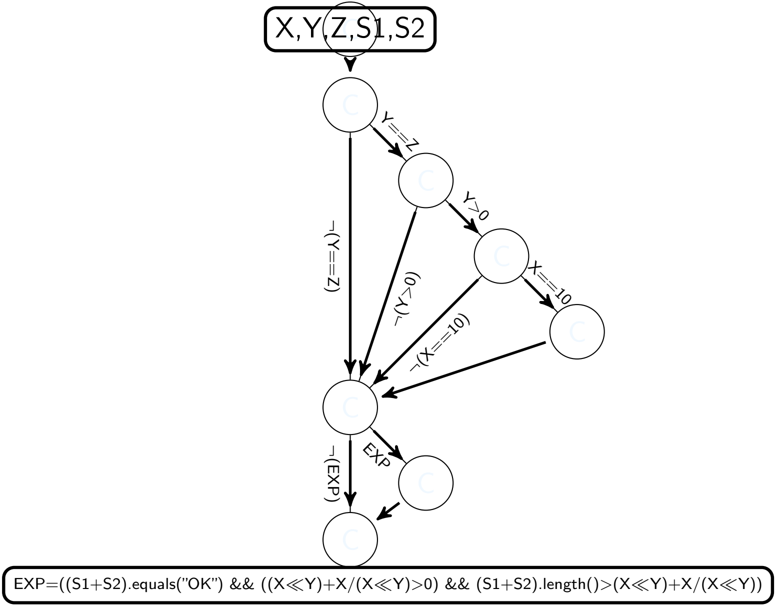
```
1  int foo(int x,int y,float z, String s1, String s2){
2      if(y==z)
3          if(y>0)
4              if(x==10)
5                  return s1.length()+s2.length();//Target
6      ...
7  }
```



Test candidate	f_{AL}	f_{SE}
i_1	$3+\frac{90}{91} = 3.9890$	$\frac{90}{91} + \frac{31}{32} + 0 = 1.9577$
i_2	$2 + \frac{21}{22} = 2.9545$	$0 + \frac{21}{22} + \frac{20}{21} = 1.9068$

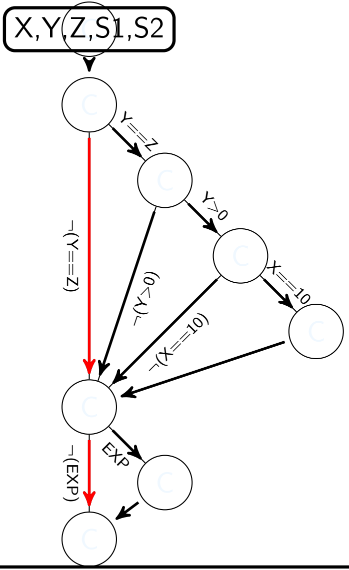
Pb2: SB-STDG may fail to generate test data to reach Line 5

Constraint Based(Symbolic Execution)



Constraint Based(Symbolic Execution)

CP1=¬(Y==Z)∧ ¬(EXP)

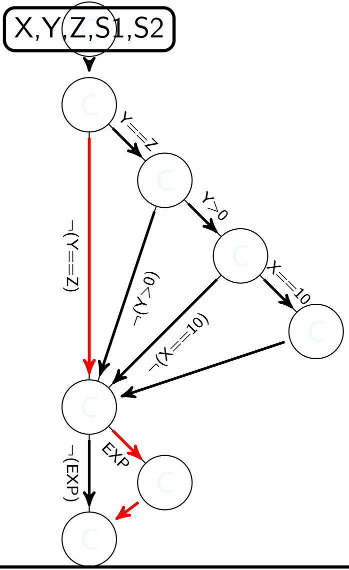


EXP=((S1+S2).equals("OK") && ((X≪Y)+X/(X≪Y)>0) && (S1+S2).length()>(X≪Y)+X/(X≪Y))

Constraint Based(Symbolic Execution)

CP1=¬(Y==Z)∧ ¬(EXP)

CP2=¬(Y==Z)∧ (EXP)



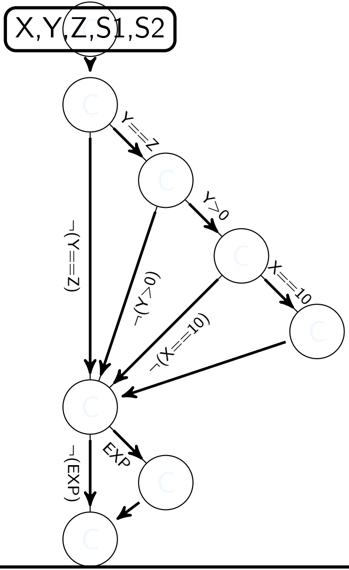
EXP=((S1+S2).equals("OK") && ((X≪Y)+X/(X≪Y)>0) && (S1+S2).length()>(X≪Y)+X/(X≪Y))

Constraint Based(Symbolic Execution)

CP1=¬(Y==Z)∧ ¬(EXP)

CP2=¬(Y==Z)∧ (EXP)

Pb3: There is no CP solver that can solve these constraints



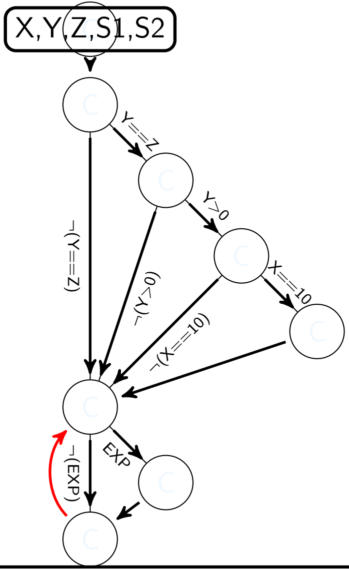
EXP=((S1+S2).equals("OK") && ((X≪Y)+X/(X≪Y)>0) && (S1+S2).length()>(X≪Y)+X/(X≪Y))

Constraint Based(Symbolic Execution)

CP1=¬(Y==Z)∧ ¬(EXP)

CP2=¬(Y==Z)∧ (EXP)

Pb3: There is no CP solver that can solve these constraints



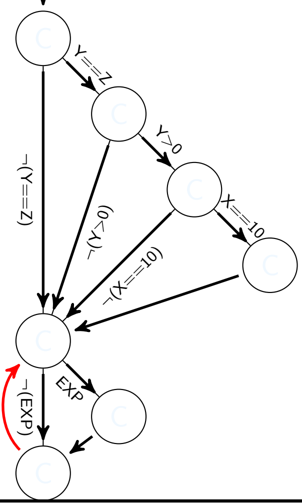
EXP=((S1+S2).equals("OK") && ((X≪Y)+X/(X≪Y)>0) && (S1+S2).length()>(X≪Y)+X/(X≪Y))

Constraint Based(Symbolic Execution)

X,Y,Z,S1,S2

CP1=¬(Y==Z)∧ ¬(EXP)

CP2=¬(Y==Z)∧ (EXP)



Pb3: There is no CP solver that can solve these constraints

Pb4: Exploring the Symbolic Tree

EXP=((S1+S2).equals("OK") && ((X≪Y)+X/(X≪Y)>0) && (S1+S2).length()>(X≪Y)+X/(X≪Y))

SB-STDG versus CB-STDG

SB-STDG

- + Scalable;
- − Incomplete;
- − Depends on many parameters.

Derives its advantages from the dynamic analysis.

CB-STDG

- − Not Scalable;
- + Complete;
- + Precision in test data generation.

Derives its advantages from the static analysis.

SB-STDG versus CB-STDG

SB-STDG

- + Scalable;
- − Incomplete;
- − Depends on many parameters.

Derives its advantages from the dynamic analysis.

CB-STDG

- − Not Scalable;
- + Complete;
- + Precision in test data generation.

Derives its advantages from the static analysis.

These approaches use complementary analyses that give them a potential for combination.

Thesis

The Goal

Improving the efficiency and effectiveness of SB-STDG for object-oriented testing and revealing bugs at an earlier stage.

The Way

- Using the main advantages of SB-STDG and CB-SDTG;
- Dynamically analyzing source code to monitor the actual execution;
- Statically analyzing source code to identify and extract relevant information;
- Exploiting relevant information to guide SB-STDG process either directly or through CB-STDG;

Contributions

Combining CB-STDG and SB-STDG

- CPA-STDG, a new CB-STDG approach;
- CSB-STDG, a new approach that combines CPA-STDG and SB-STDG;
- CB-FF, a novel constraint based fitness functions ;

Lightweight Static Analyses to improve SB-STDG

- IG-PR-IOOCC, a new SB-STDG approach for unit-class testing.

Constraint Programming Approach for Software Test Data Generation (CPA-STDG)

CPA-STDG

Idea of CPA-STDG

Idea

- Delegating the exploration of the symbolic tree to a CP Solver;
- Creating one CSP for the whole program under test and its CFG.

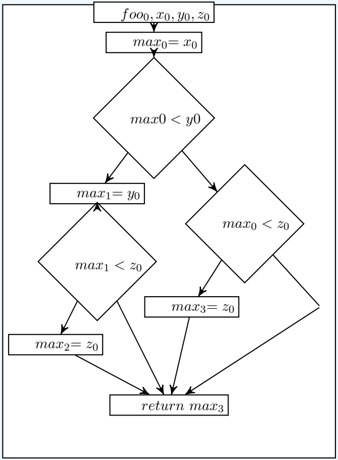
Advantages

- + Avoid the symbolic evaluation cost;
- + Benefit from the large number of search heuristics that are implanted in a Solver;
- + Avoid exploring infeasible sub-paths;
- + Easy identification of the program structure.

CPA-STDG

Constraints Generation

foo CFG

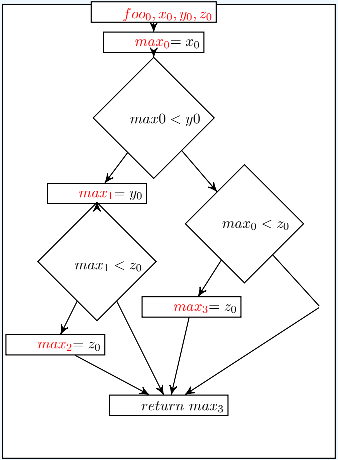


CSP of foo

CPA-STDG

Constraints Generation

foo CFG



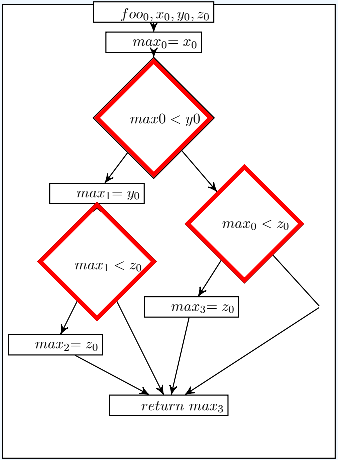
CSP of foo

Var int foo_0, x_0, y_0, z_0, max_{0-3}

CPA-STDG

Constraints Generation

foo CFG



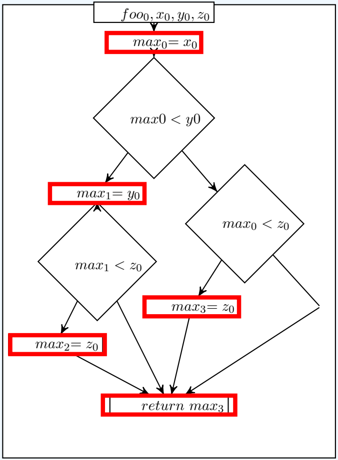
CSP of foo

Var int foo_0, x_0, y_0, z_0, max_{0-3}
Var int nD_{0-2} in {-1,0,1}

CPA-STDG

Constraints Generation

foo CFG



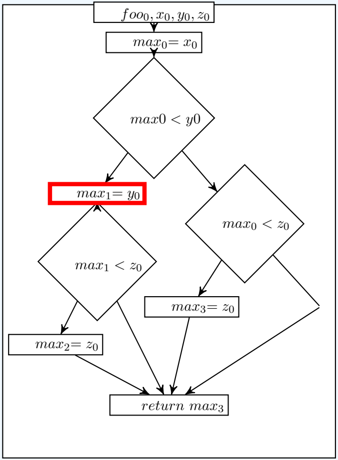
CSP of foo

Var int foo_0, x_0, y_0, z_0, max_{0-3}
Var int nD_{0-2} in {-1,0,1}
Var int nS_{0-4} in {0,1}

CPA-STDG

Constraints Generation

foo CFG



CSP of foo

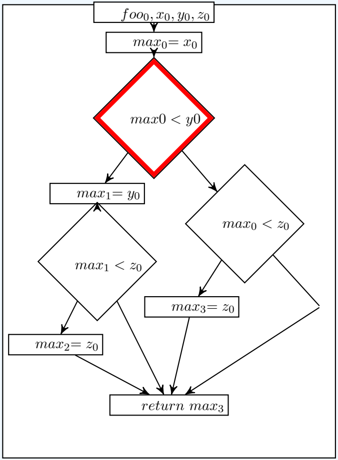
Var int foo_0, x_0, y_0, z_0, max_{0-3}
Var int nD_{0-2} in {-1,0,1}
Var int nS_{0-4} in {0,1}

$nS_1 = 1 \Rightarrow max_1 = y_0$
$nS_1 = 0 \Rightarrow max_1 = max_0$

CPA-STDG

Constraints Generation

foo CFG



CSP of foo

Var int foo_0, x_0, y_0, z_0, max_{0-3}
Var int nD_{0-2} in {-1,0,1}
Var int nS_{0-4} in {0,1}

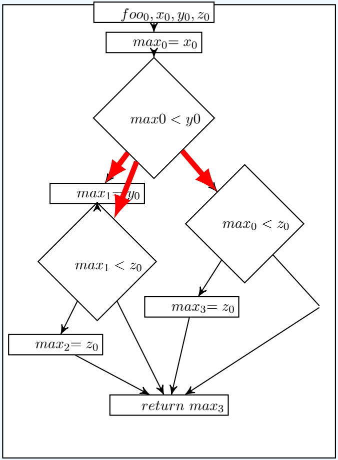
$nS_1 = 1 \Rightarrow max_1 = y_0$
$nS_1 = 0 \Rightarrow max_1 = max_0$

$nD_0 = 1 \Rightarrow max_0 < y_0$
$nD_0 = -1 \Rightarrow \neg(max_0 < y_0)$

CPA-STDG

Constraints Generation

foo CFG



CSP of foo

Var int foo_0, x_0, y_0, z_0, max_{0-3}
Var int nD_{0-2} in {-1,0,1}
Var int nS_{0-4} in {0,1}

$nS_1 = 1 \Rightarrow max_1 = y_0$
$nS_1 = 0 \Rightarrow max_1 = max_0$

$nD_0 = 1 \Rightarrow max_0 < y_0$
$nD_0 = -1 \Rightarrow \neg(max_0 < y_0)$

$nD_0 = 1 \Leftrightarrow nS_1 \neq 0$
$nD_0 = 1 \Leftrightarrow nD_1 \neq 0$
$nD_0 = -1 \Leftrightarrow nD_2 \neq 0$

CPA-STDG

Results

Comparing CPA-STDG to a Dynamic SE approach (PathCrawler)

Program	k-path	# feasible Paths	CPA-STDG		PathCrawler	
			# T. D.	T. (s)	# T. D.	T. (s)
tri_type	-	10	10	0.001	14	0.010
Sample	3	240	240	0.060	241	0.270
Merge	5	321	321	0.148	337	0.780
Merge	10	20,481	20,481	28.640	20,993	116.000

CPA-STDG
Results

Comparing CPA-STDG to a Dynamic SE approach (PathCrawler)

Program	k-path	# feasible Paths	CPA-STDG		PathCrawler	
			# T. D.	T. (s)	# T. D.	T. (s)
tri_type	-	10	10	0.001	14	0.010
Sample	3	240	240	0.060	241	0.270
Merge	5	321	321	0.148	337	0.780
Merge	10	20,481	20,481	28.640	20,993	116.000

CPA-STDG is more efficient and effective than PathCrawler.

CPA-STDG

Results

Comparing CPA-STDG to a Dynamic SE approach (PathCrawler)

Program	k-path	# feasible Paths	CPA-STDG		PathCrawler	
			# T. D.	T. (s)	# T. D.	T. (s)
tri_type	-	10	10	0.001	14	0.010
Sample	3	240	240	0.060	241	0.270
Merge	5	321	321	0.148	337	0.780
Merge	10	20,481	20,481	28.640	20,993	116.000

CPA-STDG is more efficient and effective than PathCrawler.

CPA-STDG

Results

Comparing CPA-STDG to a Dynamic SE approach (PathCrawler)

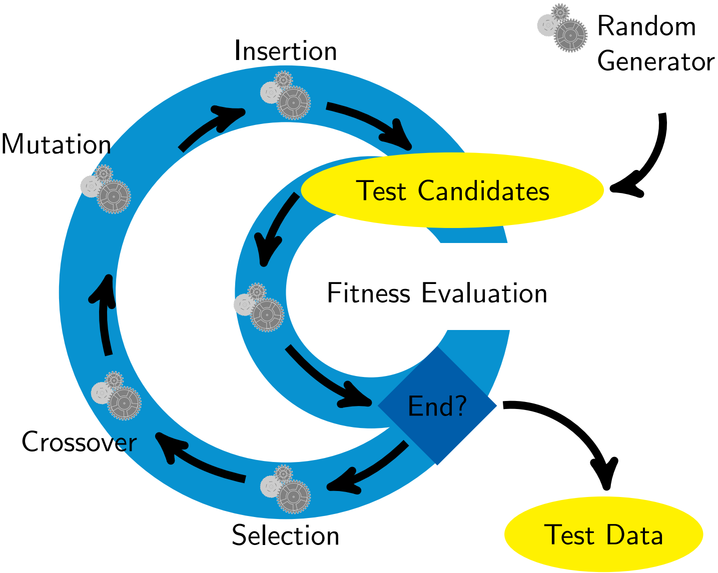
Program	k-path	# feasible Paths	CPA-STDG		PathCrawler	
			# T. D.	T. (s)	# T. D.	T. (s)
tri_type	-	10	10	0.001	14	0.010
Sample	3	240	240	0.060	241	0.270
Merge	5	321	321	0.148	337	0.780
Merge	10	20,481	20,481	28.640	20,993	116.000

CPA-STDG is more efficient and effective than PathCrawler.

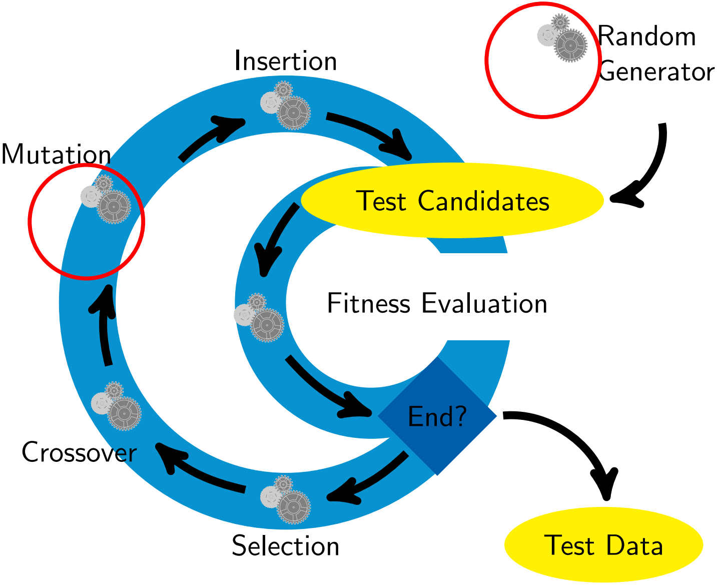
The scalability issue remains an open research question.

Constrained Search Based Software Test Data Generation (CSB-STDG).

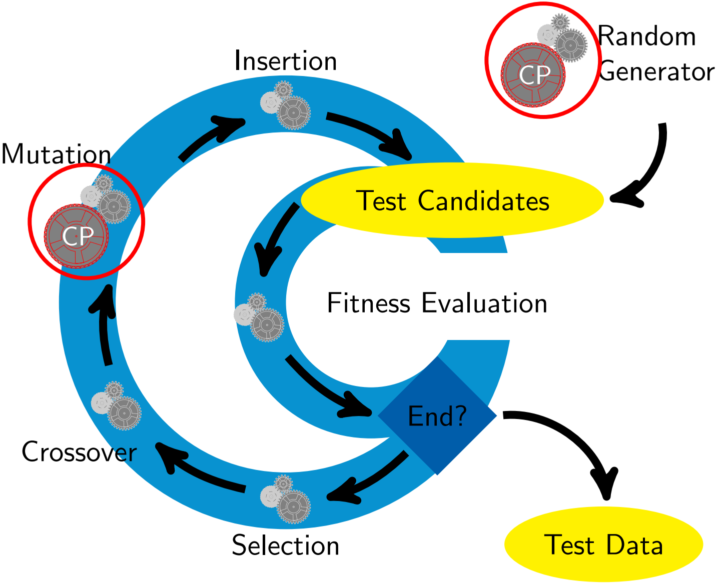
Search Based Software Test Data Generation



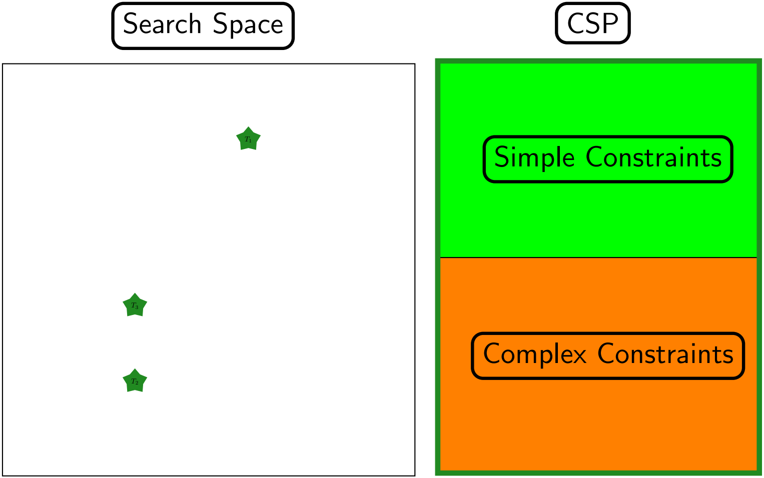
Search Based Software Test Data Generation



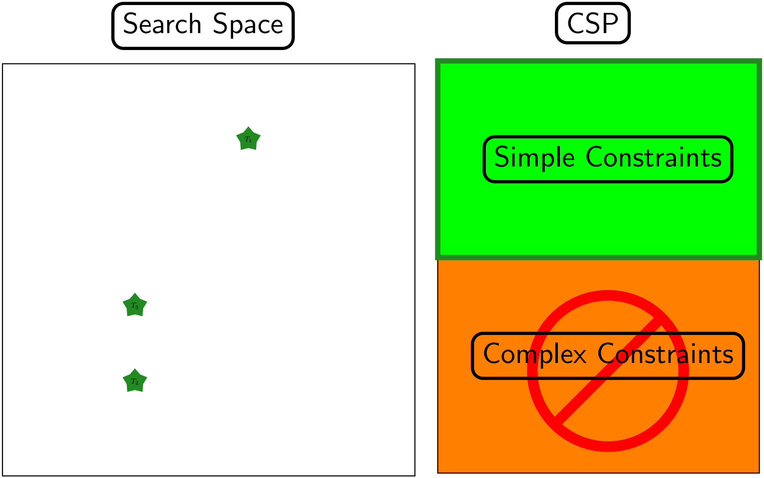
Search Based Software Test Data Generation



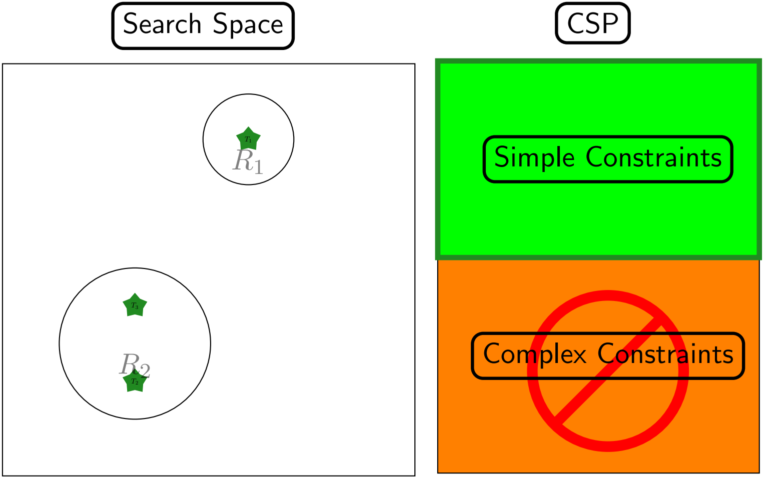
The Idea of CSB-STDG



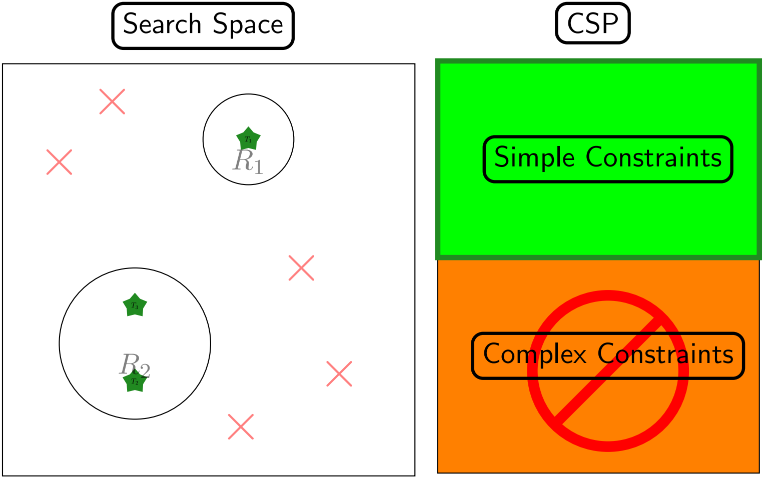
The Idea of CSB-STDG



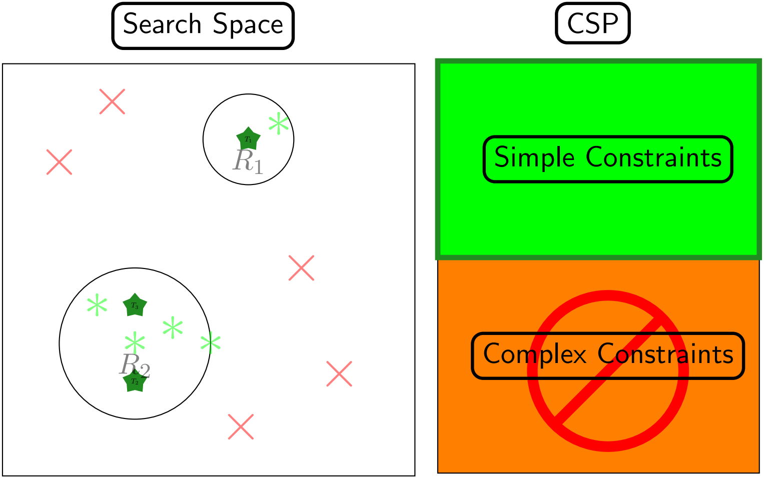
The Idea of CSB-STDG



The Idea of CSB-STDG



The Idea of CSB-STDG



Example: Unit Under Test Relaxation

```
int foo(int x,int y ~~, float z, String s1,String s2~~) {
    if(y==    ~~z~~)
        if(y>0)
            if(x==10)
                return     ~~s1.length()+s2.length()~~
     y= x<<y;
    x=y+x/y;
     ~~String s=s1+s2;~~
    if( (x>0) && R2>x)
          return y/    ~~s1.length()~~
    return 0;
 }
```

Example: Unit Under Test Relaxation

```
int foo(int x,int y , float z, String s1,String s2) {
    if(y==    z)
        if(y>0)
            if(x==10)
                return      s1.length()+s2.length()
     y= x<<y;
    x=y+x/y;
     String s=s1+s2;
    if( (x>0) && R2>x)
          return y/    s1.length()
    return 0;
 }
```

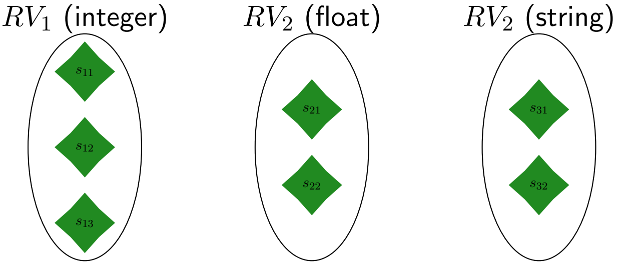
Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Example: Unit Under Test Relaxation

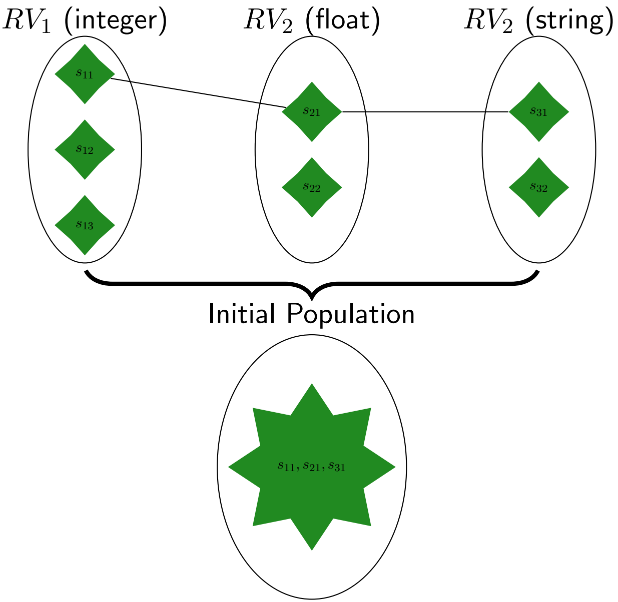
```
int foo(int x,int y                              ) {
  if(y==R0; )
    if(y>0)
      if(x==10)
        return R1;
   y= x<<y;
  x=y+x/y;

  if( (x>0) && R2>x)
      return y/R3;
  return 0;
 }
```

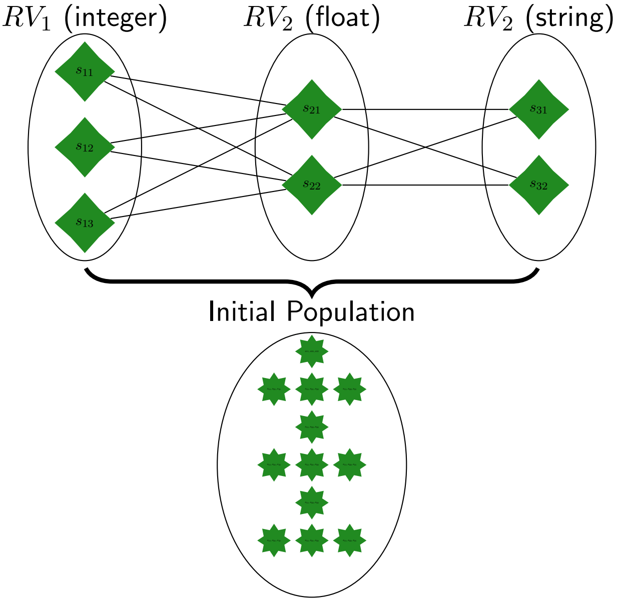
Constrained Population Generator(CPG)



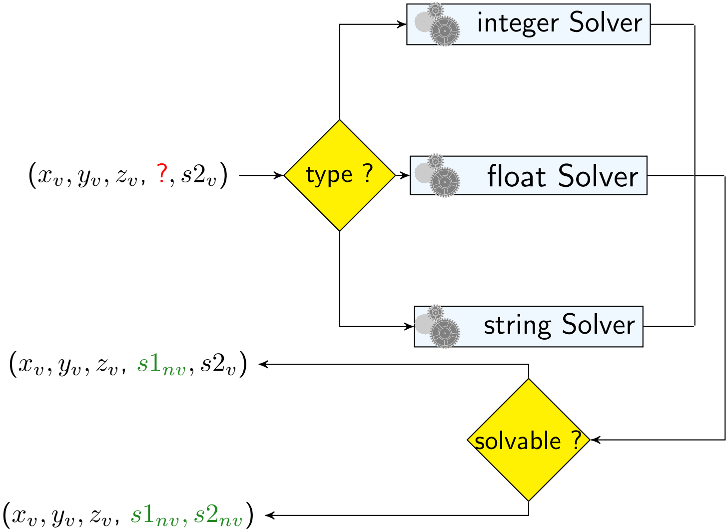
Constrained Population Generator(CPG)



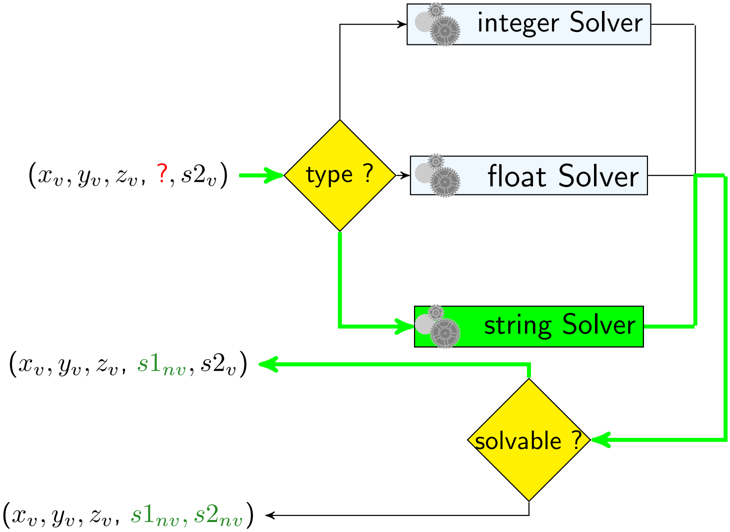
Constrained Population Generator(CPG)



Constrained Evolution Operator (CEO)



Constrained Evolution Operator (CEO)



Evaluation

Subjects

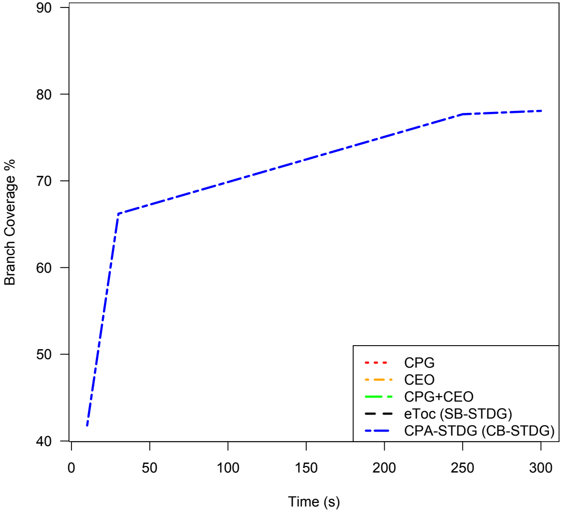
Subject	**# LOC**	**# Branches**
Integer	1244	38
BitSet	755	145
ArithmticUtils	959	102
All	**2958**	**285**

analysed approaches

- SB-STDG alone: eToc;
- CB-STDG alone: CPA-STDG;
- CSB-STDG: SB-STDG+CPG; SB-STDG+CEO.

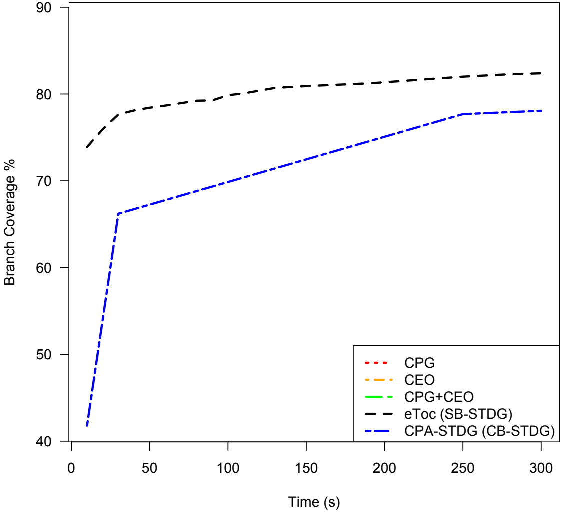
Evaluation

Results



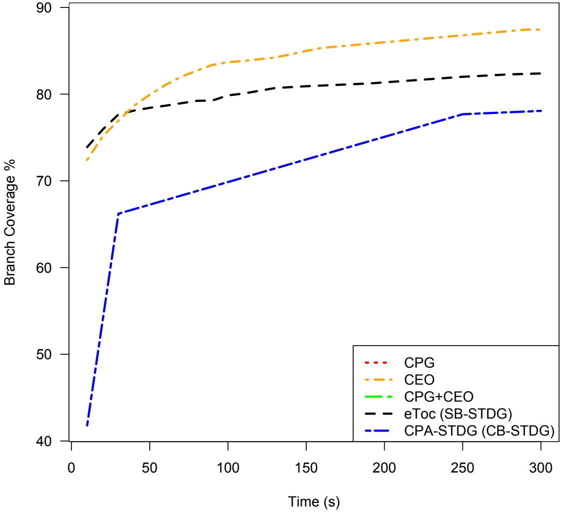
Evaluation

Results



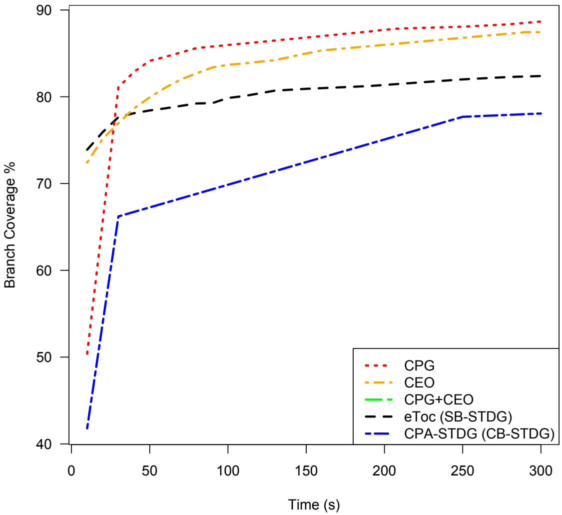
Evaluation

Results

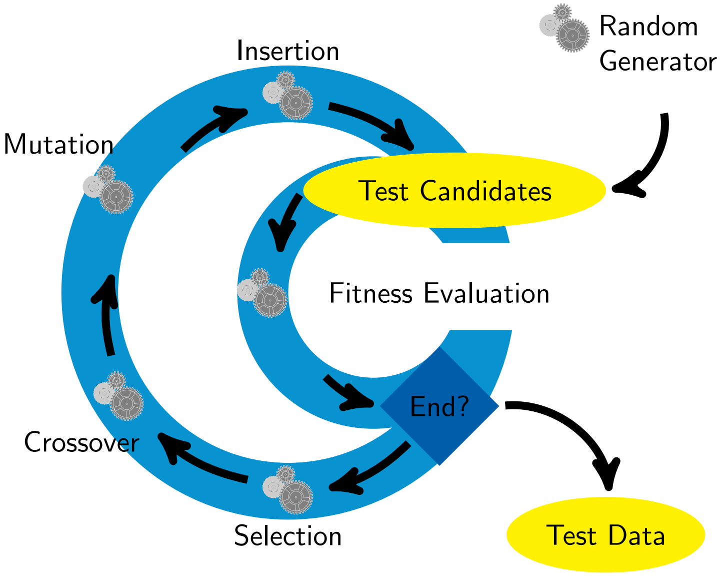


Evaluation

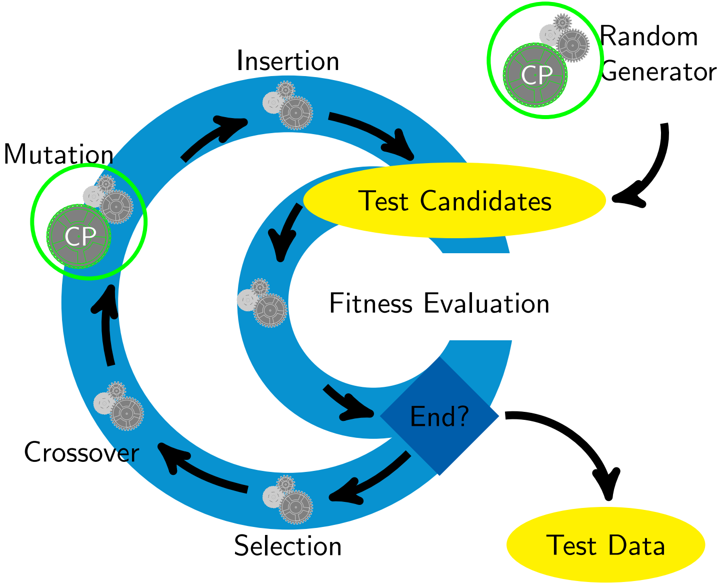
Results



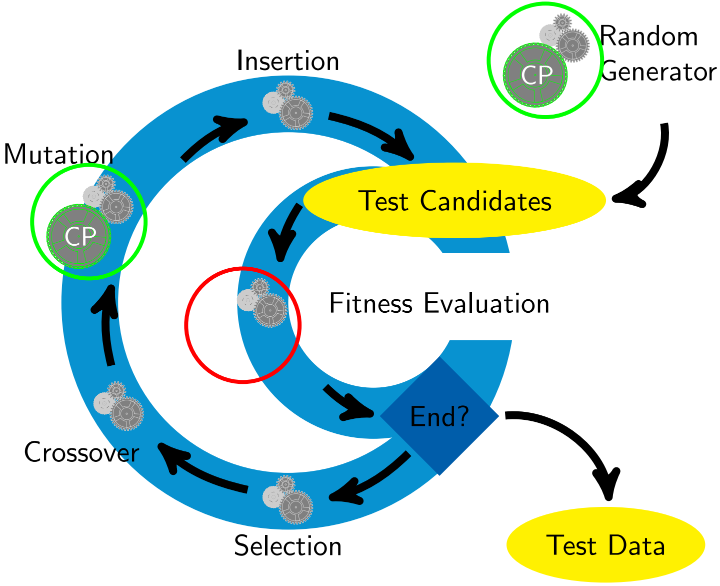
CSB-STDG


Random Generator
Insertion
Mutation
Test Candidates
Fitness Evaluation
End?
Crossover
Selection
Test Data

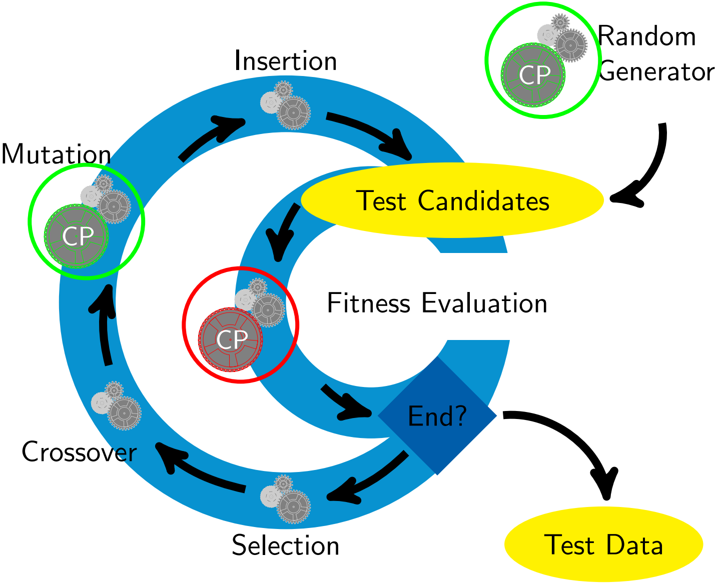
CSB-STDG


Random Generator
CP
Insertion
Mutation
CP
Test Candidates
Fitness Evaluation
End?
Crossover
Selection
Test Data

CSB-STDG


Insertion
Random Generator
CP
Mutation
CP
Test Candidates
Fitness Evaluation
End?
Crossover
Selection
Test Data

CSB-STDG


Random Generator
CP
Insertion
Mutation
CP
Test Candidates
CP
Fitness Evaluation
End?
Crossover
Selection
Test Data

Constraint Based Fitness Function (CB-FF).

The idea of CB-FF

- Prioritizing branches according to how **hard** it is to satisfy them.

The idea of CB-FF

- Prioritizing branches according to how **hard** it is to satisfy them.

Hardness

- Defining the difficulty to satisfy a constraint in terms of its **arity** and its **projection tightness**;

The idea of CB-FF

- Prioritizing branches according to how **hard** it is to satisfy them.

Hardness

- Defining the difficulty to satisfy a constraint in terms of its **arity** and its **projection tightness**;
  1. The lower the arity of the constraint, the less freedom we have to choose some of its variables in order to evolve the test candidate.

The idea of CB-FF

- Prioritizing branches according to how **hard** it is to satisfy them.

Hardness

- Defining the difficulty to satisfy a constraint in terms of its **arity** and its **projection tightness**;
  1. The lower the arity of the constraint, the less freedom we have to choose some of its variables in order to evolve the test candidate.
  2. A projection tightness close to 0 will indicate high constrainedness and hardness to satisfy a constraint.

Branch-Hardness Metrics

Difficulty Coefficient (DC)

- DC is a possible representation of the hardness of a branch;
- $DC(c) = B^2 \cdot \frac{1}{arity_c} + B \cdot (1 - tightness) + 1$.

Branch-Hardness Metrics

Difficulty Coefficient (DC)

- DC is a possible representation of the hardness of a branch;
- $DC(c) = B^2 \cdot \frac{1}{arity_c} + B \cdot (1 - tightness) + 1$.

Difficulty Level (DL)

- DL is based on DC ranking (r);
- DL is a representation of a relative hardness level of a constraint in a set of constraints;
- $DL(c, C) = \begin{cases} |C|, & \text{if } r = 0 \\ 2^{r-1} \cdot (|C| + 1), & \text{if } r > 0 \end{cases}$.

Branch-Hardness Fitness Functions

DC Fitness Function (f_{DC})

- DC is used as a penalty coefficient for breaking a constraint;
- The target of this fitness function is determining a standard-branch-distance.
- $f_{DC}(i, C) = \sum_{c \in C} DC(c) \cdot \eta(i, c)$.

Branch-Hardness Fitness Functions

DC Fitness Function (f_{DC})

- DC is used as a penalty coefficient for breaking a constraint;
- The target of this fitness function is determining a standard-branch-distance.
- $f_{DC}(i, C) = \sum_{c \in C} DC(c) \cdot \eta(i, c)$.

DL Fitness Function (f_{DL})

- DL is used as a constant penalty for breaking a constraint in a set of constraints to satisfy;
- $f_{DL}(i, C) = \sum_{c \in C} \ell(i, c) + \eta(i, c)$,

  where $\ell(i, c) = \begin{cases} 0, & \text{if } \eta(i, c) = 0 \\ DL(c, C), & \text{if } \eta(i, c) \neq 0 \end{cases}$.

Evaluation

Subjects and configuration

analyzed meta-heuristics and fitness functions

- Two widely used meta-heuristic algorithms are analyzed: Simulated Annealing (SA) and Evolutionary Algorithm (EA).
- f_{AL} and f_{SE} from the literature;
- Our fitness functions f_{DC} and f_{DL}.

Evaluation

Subjects and configuration

analyzed meta-heuristics and fitness functions

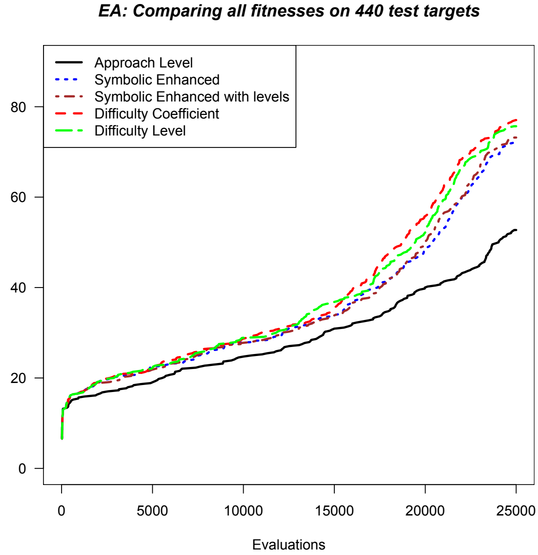
- Two widely used meta-heuristic algorithms are analyzed: Simulated Annealing (SA) and Evolutionary Algorithm (EA).
- f_{AL} and f_{SE} from the literature;
- Our fitness functions f_{DC} and f_{DL}.

Subjects

- 440 synthetic test targets that were randomly generated.
- 20 executions for every combination of fitness function and meta-heuristic algorithm.
- If test data was not found after 25,000 (respectively 100,000) fitness evaluations for EA (respectively SA), the search was terminated.

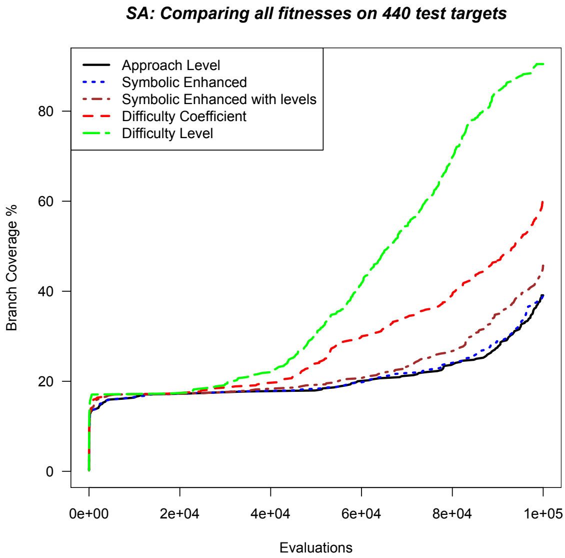
Evaluation

EA Results

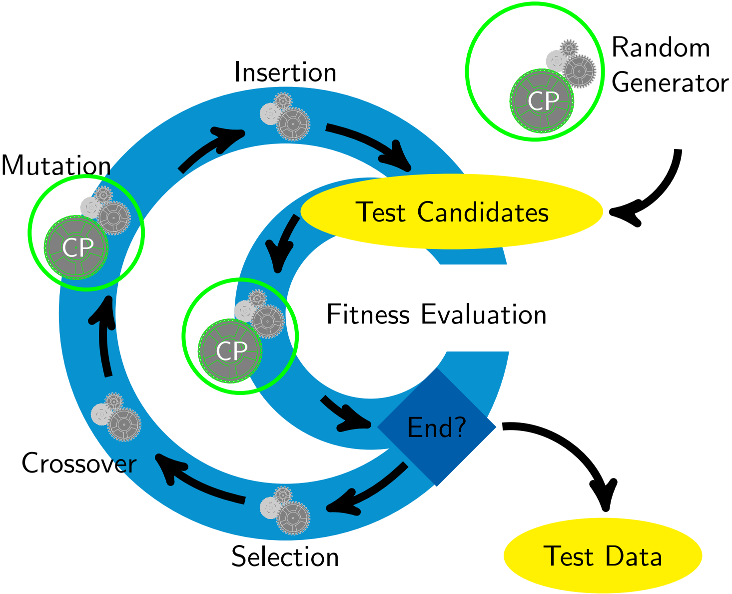


Evaluation

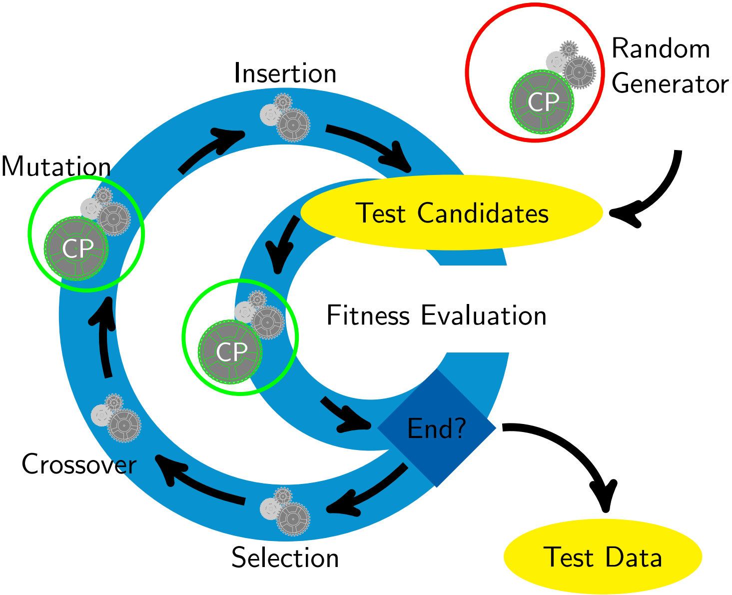
SA Results



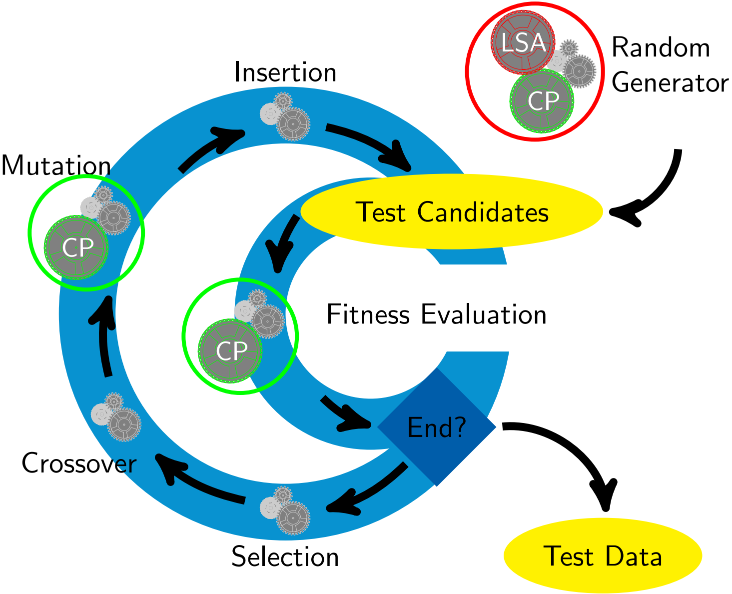
CB-FF


Random Generator
CP
Insertion
Mutation
CP
Test Candidates
CP
Fitness Evaluation
End?
Crossover
Selection
Test Data

CB-FF


Random Generator
CP
Insertion
Test Candidates
Mutation
CP
CP
Fitness Evaluation
End?
Crossover
Selection
Test Data

CB-FF


Insertion
LSA
CP
Random Generator
Mutation
CP
Test Candidates
CP
Fitness Evaluation
End?
Crossover
Selection
Test Data

Further Static Analyses to Improve SB-STDG (IG-PR-IOOCC).

IG-PR-IOOCC

The idea of IG-PR-IOOCC

Exploring relevant sequences

- Methods that may modify a data member or can reach the test target;
- A static analysis can determine the relevant methods.

Efficiently generate, diversify, and enrich instances of classes

- Instantiate a class with a minimum cost;
- Diversify instances of classes by using different means-of-instantiation;
- Seed instances of classes by using the constants existing in the source code.

IG-PR-IOOCC

The idea of IG-PR-IOOCC

	Requirements	To analyze	To extract
(I)	A means-of-instantiation of the class under test and each required argument	▶ Constructors; ▶ Factory methods; ▶ Data Members; ▶ Derived Classes (SubClasses).	Set of means-of-instantiation.
(II)	A sequence of method calls to make the class under test in a relevant state.	Data Member$_1$ Data Member$_2$ ⋮ Data Member$_m$	State Modifier$_1$ State Modifier$_2$ ⋮ State Modifier$_m$
(III)	A method that can reach the test target.	Method Under Test	Public Methods caller

IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
      ...
      foo(a,str);
      ...
}
private int foo(int x, String s1){
      ...
}
      ...
}
```

IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses

MIn

IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses

MIn FooClass()

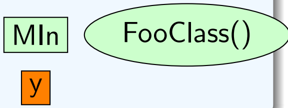
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses



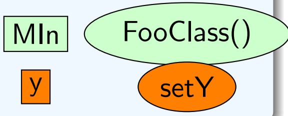
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses



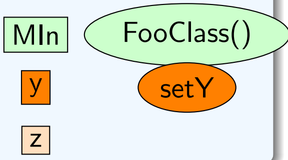
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
      ...
      foo(a,str);
      ...
}
private int foo(int x, String s1){
      ...
}
      ...
}
```

Static Analyses



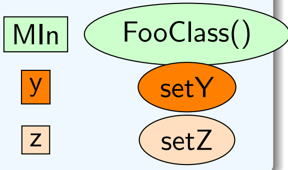
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses



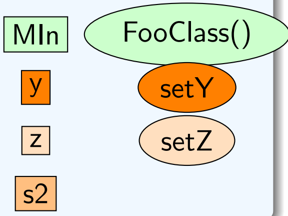
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
      ...
      foo(a,str);
      ...
}
private int foo(int x, String s1){
      ...
}
      ...
}
```

Static Analyses



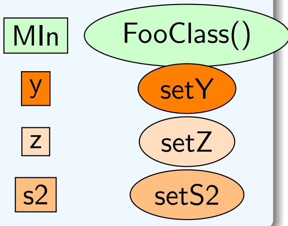
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
      ...
      foo(a,str);
      ...
}
private int foo(int x, String s1){
      ...
}
      ...
}
```

Static Analyses



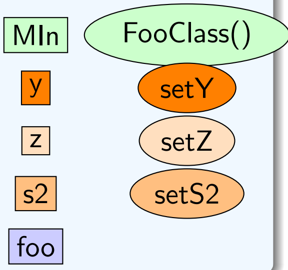
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
      ...
      foo(a,str);
      ...
}
private int foo(int x, String s1){
      ...
}
      ...
}
```

Static Analyses



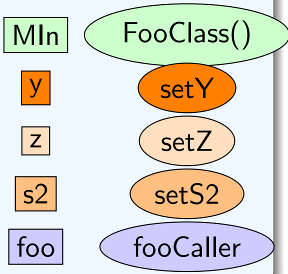
IG-PR-IOOCC

Example

Source Code

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    ...
    foo(a,str);
    ...
}
private int foo(int x, String s1){
    ...
}
    ...
}
```

Static Analyses



Evaluation I

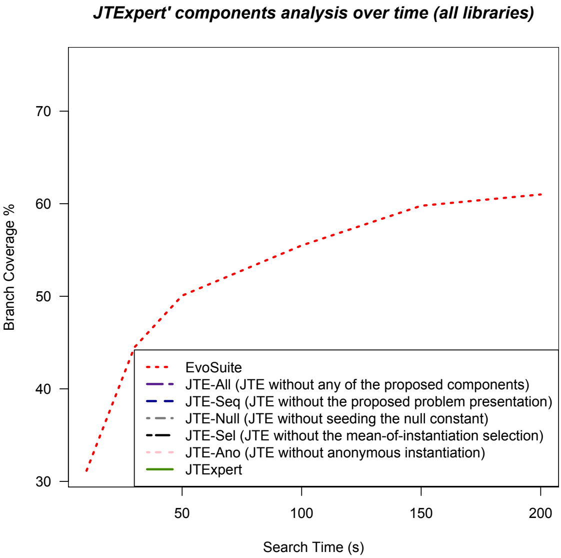
Subjects

Compared systems

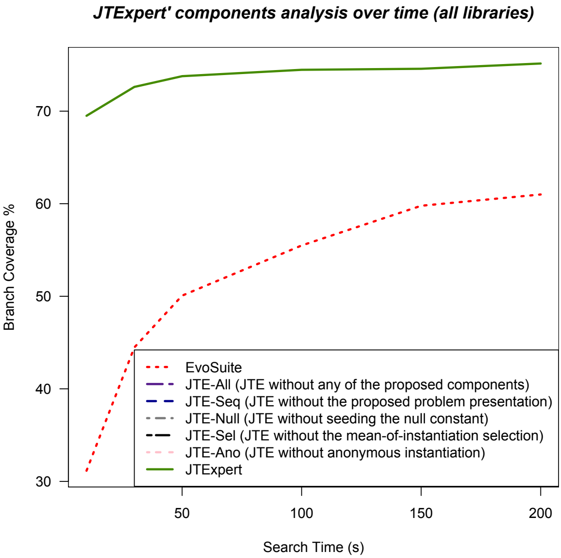
- EvoSuite;
- JTExpert.
  - Instance Generator;
  - Generator of sequences of method calls.

Libraries	Files	Classes	Methods	Branches	Lines	Instructions
Joda-Time	50	87	1,411	3,052	5,876	25,738
Barbecue	18	18	161	323	1,041	14,558
Commons-lang	5	6	366	1,942	2,134	9,139
Lucene	2	4	58	202	262	1,364
All	**75**	**115**	**1,998**	**5,519**	**9,313**	**50,799**

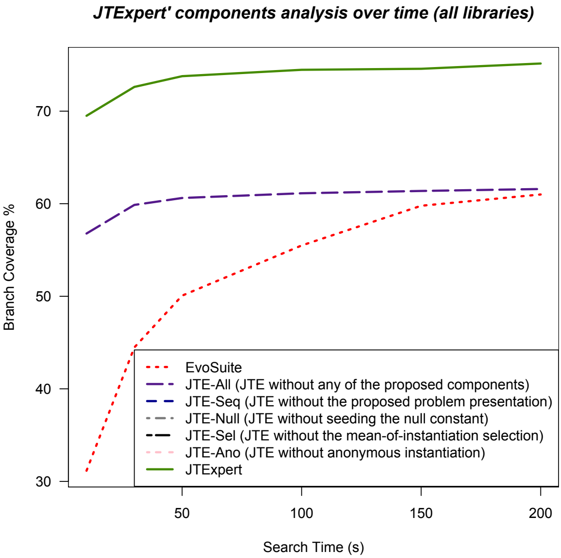
Results


JTExpert' components analysis over time (all libraries)
Branch Coverage %
Search Time (s)
30
40
50
60
70
50
100
150
200
EvoSuite
JTE-All (JTE without any of the proposed components)
JTE-Seq (JTE without the proposed problem presentation)
JTE-Null (JTE without seeding the null constant)
JTE-Sel (JTE without the mean-of-instantiation selection)
JTE-Ano (JTE without anonymous instantiation)
JTExpert

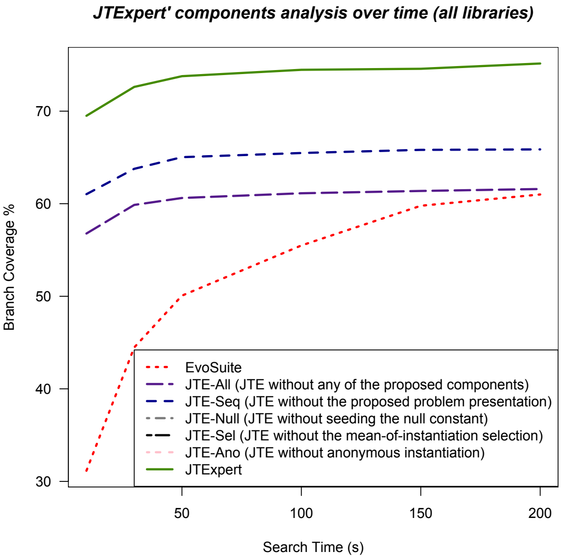
Results


JTExpert' components analysis over time (all libraries)
Branch Coverage %
30
40
50
60
70
50
100
150
200
Search Time (s)
EvoSuite
JTE-All (JTE without any of the proposed components)
JTE-Seq (JTE without the proposed problem presentation)
JTE-Null (JTE without seeding the null constant)
JTE-Sel (JTE without the mean-of-instantiation selection)
JTE-Ano (JTE without anonymous instantiation)
JTExpert

Results


JTExpert' components analysis over time (all libraries)
Branch Coverage %
Search Time (s)
30
40
50
60
70
50
100
150
200
EvoSuite
JTE-All (JTE without any of the proposed components)
JTE-Seq (JTE without the proposed problem presentation)
JTE-Null (JTE without seeding the null constant)
JTE-Sel (JTE without the mean-of-instantiation selection)
JTE-Ano (JTE without anonymous instantiation)
JTExpert

Results


JTExpert' components analysis over time (all libraries)
Branch Coverage %
Search Time (s)
30
40
50
60
70
50
100
150
200
EvoSuite
JTE-All (JTE without any of the proposed components)
JTE-Seq (JTE without the proposed problem presentation)
JTE-Null (JTE without seeding the null constant)
JTE-Sel (JTE without the mean-of-instantiation selection)
JTE-Ano (JTE without anonymous instantiation)
JTExpert

JTExpert: A Large Scale Experimentation

Exceptions-oriented Test-data Generation

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
    foo(a,str);...
}
private int foo(int x, String s1){
   if(y==z)
      if(y>0)
         if(x==10)
            return s1.length()+s2.length();
   y= x<<y;
   x=y+x/y;
   String s=s1+s2;
   if((s.equals("OK") && (x>0) && s.length()>x)
      return y/s1.length();
   return 0;
}
}
```

JTExpert: A Large Scale Experimentation

Exceptions-oriented Test-data Generation

```
public class FooClass{
private int y;
private float z;
private String s2;
public void setY(int y1) {y=y1};
public void setZ(float z1) {z=z1};
public void setS2(String s) {s2=s};
public int fooCaller(int a, String str){
     foo(a,str);...
}
private int foo(int x, String s1){
    if(y==z)
        if(y>0)
            if(x==10)
                return s1.length()+s2.length();
    y= x<<y;
    x=y+x/y;
    String s=s1+s2;
    if((s.equals("OK") && (x>0) && s.length()>x)
        return y/s1.length();
    return 0;
}
}
```

JTExpert: A Large Scale Experimentation

Subjects

The Apache Hadoop framework

The Apache Hadoop framework allows the distributed processing of large data sets across clusters of computers using simple programming models.

- 43 sub-modules;
- 3,545 Java files;
- 7,531 classes;
- 88,971 methods;
- 148,691 branches;
- 421,032 lines of code;
- 1,698,650 statements.

JTExpert: A Large Scale Experimentation

Results

Exception	Alerted	Alerted in different		
		Lines	Methods	Classes
`NullPointerException`	13,663	3,604	3,377	1,343
`ClassCastException`	658	213	206	165
`NoClassDefFoundError`	602	87	86	86
`IllegalArgumentException`	326	104	102	100
`ArrayIndexOutOfBoundsException`	296	119	108	71
`Fatal Errors`	9	2	2	2
`Other Exceptions`	492	193	180	153
All Exceptions	**16,046**	**4,322**	**4,061**	**1,920**

JTExpert: A Large Scale Experimentation

Results

Exception	Alerted	Alerted in different		
		Lines	Methods	Classes
NullPointerException	13,663	3,604	3,377	1,343
ClassCastException	658	213	206	165
NoClassDefFoundError	602	87	86	86
IllegalArgumentException	326	104	102	100
ArrayIndexOutOfBoundsException	296	119	108	71
Fatal Errors	9	2	2	2
Other Exceptions	492	193	180	153
All Exceptions	**16,046**	**4,322**	**4,061**	**1,920**

JTExpert: A Large Scale Experimentation

Results

The source of Fatal Errors

- Package: `org.apache.hadoop.io`;
- Class: `WritableComparator`;
- Method: `compareBytes(byte[] b1, int s1, int l1,byte[] b2, int s2, int l2)`;
- How to reproduce it : use `b1=null` or `b2=null`;
- The root: The class `FastByteComparisons` uses reflection to get an instance of the private class `sun.misc.Unsafe`;
- Discussed at: `http://comments.gmane.org/gmane.comp.java.hadoop.hbase.devel/39017`.

JTExpert: A Large Scale Experimentation

- New instance generator;
- New representation of the test-data generation problem;
- Strategy of revealing bugs;
- JTExpert is available for download at `https://sites.google.com/site/saktiabdel/JTExpert`.

Conclusions

- CPA-STDG (**A. SAKTI et al. JFPC' 11**);
  - CP is a promising approach for automatic test-data generation.
- CSB-STDG (**A. SAKTI et al. SSBSE' 12**) and CB-FF (**A. SAKTI et al. CPAIOR' 13**);
  - CP is useful for SB-STDG;
- IG-PR-IOOCC (**A. SAKTI et al. TSE' 14**);
  - Lightweight static analyses is a way to improve SB-STDG.

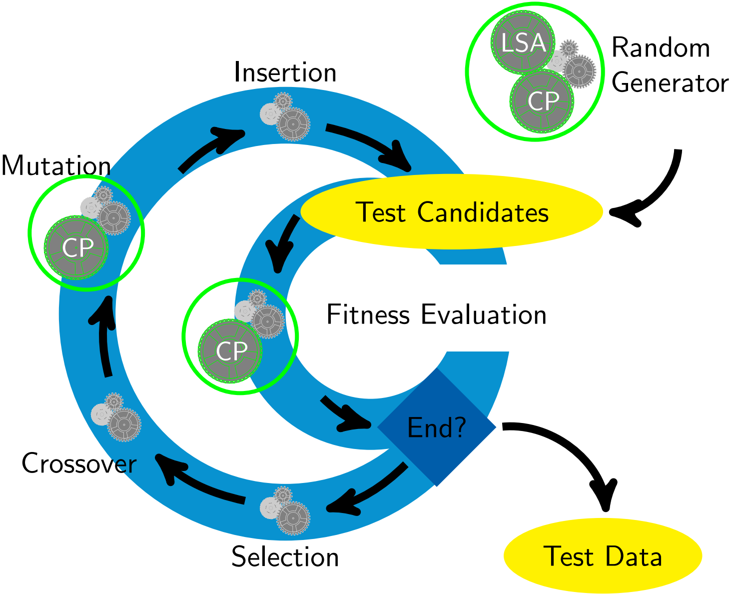
Conclusions

- CPA-STDG (**A. SAKTI et al. JFPC' 11**);
  - CP is a promising approach for automatic test-data generation.
- CSB-STDG (**A. SAKTI et al. SSBSE' 12**) and CB-FF (**A. SAKTI et al. CPAIOR' 13**);
  - CP is useful for SB-STDG;
- IG-PR-IOOCC (**A. SAKTI et al. TSE' 14**);
  - Lightweight static analyses is a way to improve SB-STDG.

Future work

- Further experiments;
- Broadening and enlarging internal static analyses;
- Statically analyzing new external sources to derive information that are relevant for test-cases generation.

Thank you, Questions?



Start fitter and finish better.