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 16th, 2014, Montréal QC
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.
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

Motivation

Software testing

Strategies

White-box Testing

Goal

Requirements

Test-Data Generation for Unit Class Testing

Test-Data Generation for Unit Class Testing

Test-Data Format of OOP

Automation

Search Based

Genetic Algorithms

Constraint Based

Symbolic Execution

SB-STDG versus CB-STDG

Thesis

Contributions
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)

```java
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;
}
```
Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

Motivation
Software testing
Strategies
White-box Testing

Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based
Genetic Algorithms
Constraint Based
Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions

White-box Testing (Goal)

```java
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;
}
```

Unhandled Exceptions

- Line 5: undefined behavior if \( s_1 \) or \( s_2 \) is null.
- Line 6: undefined behavior if \( y \) is negative or \( >31 \);
- Line 7: undefined behavior if \( y = 0 \);
- Line 8: undefined behavior if \( s_1 \) or \( s_2 \) is null;
- Line 10: undefined behavior if \( s_1 = "\) or is null.
White-box Testing (Requirements)

Control Flow Graph of foo

Coverage Criteria

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

Introduction
Motivation
Software testing
Strategies
White-box Testing
Goal

Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based
Genetic Algorithms
Constraint Based
Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions

CPA-STDG
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.

Introduction
Motivation
Software testing
Strategies
White-box Testing
Goal

Requirements
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP
Automation
Search Based
Genetic Algorithms
Constraint Based
Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions
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.

- **Introduction**
  - Motivation
  - Software testing
  - Strategies
  - White-box Testing
  - Goal

- **Requirements**
  - Test-Data Generation for Unit Class Testing
  - Test-Data Generation for Unit Class Testing
  - Test-Data Format of OOP Automation
  - Search Based
  - Genetic Algorithms
  - Constraint Based
  - Symbolic Execution
  - SB-STDG versus CB-STDG
  - Thesis
  - Contributions
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

```java
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;
    }
}
```
Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction
Motivation
Software testing Strategies
White-box Testing
Goal Requirements
Test-Data Generation for Unit Class Testing

Test-Data Format of OOP
Automation Search Based Genetic Algorithms Constraint Based Symbolic Execution SB-STDG versus CB-STDG Thesis Contributions

Test-Data Format of OOP

Test Target

Method Under Test

Argument_1

... 

Argument_n
Test-Data Format of OOP

Method Under Test

Test Target

Argument$_1$

... 

Argument$_n$

Public ?
Test-Data Format of OOP

Test Target

Method Under Test

Argument₁

...  

Argumentₙ

Public ?

© 9 / 51
Test-Data Format of OOP

Method Under Test

Argument_1

... 

Argument_n

Public ?

Test Target

Automation
Search Based
Genetic Algorithms
Constraint Based
Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions
Test-Data Format of OOP

- Method Under Test
  - \( \text{Argument}_1 \)
  - \( \ldots \)
  - \( \text{Argument}_n \)

- Test Target
  - The state of the class under test
    - \( \text{Data Member}_1 \)
    - \( \ldots \)
    - \( \text{Data Member}_m \)

- Public?
  - Inaccessible
Test-Data Format of OOP

Method Under Test

Test Target

The state of the classe under test

Public ?

Argument_1

\ldots

Argument_n

Data Member_1

\ldots

Data Member_m

Inaccessible

©
Test-Data Format of OOP

Test Target

Method Under Test

\[ \text{Argument}_1 \]

\[ \vdots \]

\[ \text{Argument}_n \]

Public ?

The state of the classe under test

\[ \text{Data Member}_1 \]

\[ \text{Data Member}_2 \]

\[ \vdots \]

\[ \text{Data Member}_m \]

Inaccessible

Public methods to reach inaccessible parts.

Introduction

Motivation

Software testing

Strategies

White-box Testing

Goal

Requirements

Test-Data Generation for Unit Class Testing

Test-Data Generation for Unit Class Testing

Thesis

Contributions
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
Introduction

Motivation
Software testing Strategies
White-box Testing Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP

Automation
Search Based (SB-STDG) vs Constraint Based (CB-STDG)

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

Conclusions

Tesis Contributions

CPA-STDG

CSB-STDG

CB-FF

IG-PR-IOOCC

Automation

Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques
Abdelilah Sakti Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc
Search Based (Genetic Algorithms)

Random Generator

Fitness Evaluation

Mutation

Insertion

Crossover

Selection

Test Candidates

Test Data

End?
**Search Based**

```plaintext
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
    ...
}
```

**Introduction**

Motivation

Software testing

Strategies

White-box Testing

Goal

Requirements

Test-Data Generation for

Unit Class Testing

Test-Data Generation for

Unit Class Testing

Test-Data Format of OOP

Automation

Search Based

**Genetic Algorithms**

Constraint Based

Symbolic Execution

SB-STDG versus CB-STDG

Thesis

Contributions
Search Based

```java
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  }
```

Constraint Based
**Search Based**

```plaintext
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
            }

        }
    }
    ... 
    }
```

Diagram:

- Node: $x, y, z$
  - $i_1: (10, -30, 60)$
  - $i_2: (30, -20, -20)$

END

Target
Search Based

```c
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
    ...
}
```

```
int exp(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
    ...
}
```
Introduction

Motivation
Software testing
Strategies
White-box Testing
Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based

Test candidate | $f_{AL}$ | $f_{SE}$
--- | --- | ---
$i_1$ | 3 | |
$i_2$ | | |
Search Based

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

<table>
<thead>
<tr>
<th>Test candidate</th>
<th>$f_{AL}$</th>
<th>$f_{SE}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$3 + \frac{90}{91}$</td>
<td></td>
</tr>
<tr>
<td>$i_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Test candidate** | $f_{AL}$ | $f_{SE}$
---|---|---
$i_1$ | $3 + \frac{90}{91}$ | $\frac{90}{91}$
$i_2$ | | |

```
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
  ...
}
```

**Introduction**

**Motivation**

Software testing

White-box Testing

Goal

Requirements

Test-Data Generation for Unit Class Testing

Test-Data Generation for Unit Class Testing

Test-Data Format of OOP Automation

Search Based

**Genetic Algorithms**

Constraint Based

Symbolic Execution

SB-STDG versus CB-STDG

Thesis

Contributions

---

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

**Abdelilah Sakti**

Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

**Search Based**
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 | $f_{AL}$ | $f_{SE}$
----------------|-----------|-----------
$i_1$           | $3 + \frac{90}{91}$ | $\frac{90}{91} + \frac{31}{32}$
$i_2$           |           |           

Thesis

Contributions

CPA-STDG

Constraint Based
Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions
Search Based

```java
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       } ...
```

**Table:**

<table>
<thead>
<tr>
<th>Test candidate</th>
<th>( f_{AL} )</th>
<th>( f_{SE} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>3 + \frac{90}{91}</td>
<td>\frac{90}{91} + \frac{31}{32} + 0</td>
</tr>
<tr>
<td>( i_2 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- \( i_1 : (10, -30, 60) \)
- \( i_2 : (30, -20, -20) \)

**Critical Branch of \( i_1 \):**

- \( x \leq 10 \)
- \( y > 0 \)

**Target:**

- \( x = 10 \)
- \( y = z \)
Introduction

Motivation
Software testing
Strategies
White-box Testing
Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based

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

Motivation
Software testing
White-box Testing
Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based

<table>
<thead>
<tr>
<th>Test candidate</th>
<th>$f_{AL}$</th>
<th>$f_{SE}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1$</td>
<td>$3 + \frac{90}{91} = 3.9890$</td>
<td>$\frac{90}{91} + \frac{31}{32} + 0 = 1.9577$</td>
</tr>
<tr>
<td>$i_2$</td>
<td>$2 + \frac{21}{22} = 2.9545$</td>
<td>$0 + \frac{21}{22} + \frac{20}{21} = 1.9068$</td>
</tr>
</tbody>
</table>

```
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

```c++
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         }
8     }
9 }
```

Test candidate $f_{AL}$ $f_{SE}$

| i₁ | $3 + \frac{90}{91} = 3.9890$ | $\frac{90}{91} + \frac{31}{32} + 0 = 1.9577$ |
| i₂ | $2 + \frac{21}{22} = 2.9545$ | $0 + \frac{21}{22} + \frac{20}{21} = 1.9068$ |

**Pb2:** SB-STDG may fail to generate test data to reach Line 5
Constraint Based (Symbolic Execution)

\[
\text{EXP} = ((S1+S2).equals("OK") \&\& ((X\ll Y)+X/(X\ll Y)>0) \&\& (S1+S2).length()>(X\ll Y)+X/(X\ll Y))
\]
Constraint Based (Symbolic Execution)

\[
CP_1 = \neg(Y==Z) \land \neg(EXP)
\]

\[
EXP = ((S1+S2).equals("OK") \land ((X\ll Y)+X/(X\ll Y)>0) \land (S1+S2).length()>(X\ll Y)+X/(X\ll Y))
\]
Introduction
Motivation
Software testing
Strategies
White-box Testing
Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP
Automation
Search Based
Genetic Algorithms
Constraint Based

Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions

Constraint Based (Symbolic Execution)

\[ \text{CP1} = \neg(Y==Z) \land \neg(\text{EXP}) \]

\[ \text{CP2} = \neg(Y==Z) \land (\text{EXP}) \]

\[ \text{EXP} = ((S1+S2).equals("OK") \land ((X \ll Y)+X/(X \ll Y) > 0) \land (S1+S2).length() > (X \ll Y)+X/(X \ll Y)) \]
Pb3: There is no CP solver that can solve these constraints
Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction
Motivation
Software testing Strategies
White-box Testing
Goal
Requirements
Test-Data Generation for Unit Class Testing
Test-Data Generation for Unit Class Testing
Test-Data Format of OOP Automation
Search Based
Genetic Algorithms
Constraint Based

Symbolic Execution
SB-STDG versus CB-STDG
Thesis
Contributions

Constraint Based (Symbolic Execution)

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

\( CP1 = \neg(Y=Z) \land \neg(EXP) \)

\( CP2 = \neg(Y=Z) \land (EXP) \)

\( \neg(X\ll Y) \land \neg(Y=Z) \land \neg(EXP) \)

\( EXP=((S1+S2).equals("OK") \land \left( (X\ll Y)+X/(X\ll Y)>0 \right) \land (S1+S2).length()>\left( (X\ll Y)+X/(X\ll Y) \right)) \)
Constraint Based (Symbolic Execution)

**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/(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.
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

\[
\begin{align*}
    \text{foo}_0, x_0, y_0, z_0 & \\
    \max_0 &= x_0 & 1 \\
    \max_0 &< y_0 & 2 \\
    \max_1 &= y_0 & 3 \\
    \max_1 &< z_0 & 4 \\
    \max_2 &= z_0 & 5 \\
    \max_3 &= z_0 & 6 \\
    \text{return} \ max_3 & & 7 \\
\end{align*}
\]
Introduction

CPA-STDG

Idea of CPA-STDG

Constraints Generation

Results

CSB-STDG

CB-FF

IG-PR-IOOCC

Conclusions
Introduction

CPA-STDG

Idea of CPA-STDG

Constraints Generation

Results

CSB-STDG

CB-FF

IG-PR-IOOCC

Conclusions

---

**CPA-STDG**

**Constraints Generation**

---

**foo CFG**

```
foo, x0, y0, z0
max0 = x0
max0 < y0
max1 = y0
max1 < z0
max2 = z0
return max3
```

---

**CSP of foo**

```plaintext
1 Var int foo0, x0, y0, z0, max0-3
2 Var int nD0-2 in {-1,0,1}
```
Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG
Idea of CPA-STDG
Constraints Generation
Results

CSB-STDG
CB-FF
IG-PR-IOOCC
Conclusions

---

**foo CFG**

```
foo0, x0, y0, z0

max0 = x0

max0 < y0

max1 = y0

max1 < z0

max2 = z0

max3 = z0

return max3
```

**CSP of foo**

1. Var int $foo0, x0, y0,$ $z0, max0−3$
2. Var int $nD0−2$ in $\{-1,0,1\}$
3. Var int $nS0−4$ in $\{0,1\}$
Introduction

CPA-STDG

Idea of CPA-STDG

Constraints Generation

Results

CSB-STDG

CB-FF

IG-PR-IOOCC

Conclusions

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

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

CPA-STDG

Constraints Generation

foo CFG

CSP of foo

```latex
Var int foo0, x0, y0, z0, max0-3
Var int nD0-2 in {-1,0,1}
Var int nS0-4 in {0,1}

nS1 = 1 \Rightarrow max1 = y0
nS1 = 0 \Rightarrow max1 = max0
```
**Introduction**

**CPA-STDG**

Idea of CPA-STDG

Constraints Generation

**Results**

**CSB-STDG**

CB-FF

IG-PR-I00CC

Conclusions

---

**CSP of foo**

```plaintext
1 Var int foo0, x0, y0, z0, max0-3
2 Var int nD0-2 in {-1,0,1}
3 Var int nS0-4 in {0,1}

4 nS1 = 1 ⇒ max1 = y0
5 nS1 = 0 ⇒ max1 = max0

6 nD0 = 1 ⇒ max0 < y0
7 nD0 = -1 ⇒ ¬(max0 < y0)
```

---

**foo CFG**

```
foo0, x0, y0, z0

max0 = x0

max0 < y0

max1 = y0

max1 < z0

max3 = z0

max2 = z0

return max3
```
Introduction

CPA-STDG

Idea of CPA-STDG

Constraints Generation

Results

CSB-STDG

CB-FF

IG-PR-IOOCC

Conclusions

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

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG

Idea of CPA-STDG

Constraints Generation

Results

CSB-STDG

CB-FF

IG-PR-IOOCC

Conclusions

CSP of foo

1. \textbf{Var} int $foo_0, x_0, y_0, z_0, max_0-3$
2. \textbf{Var} int $nD_{0-2}$ in \{-1,0,1\}
3. \textbf{Var} int $nS_{0-4}$ in \{0,1\}

\begin{align*}
&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
\end{align*}

foo CFG

CSP of foo

1. \textbf{Var} int $foo_0, x_0, y_0, z_0, max_0-3$
2. \textbf{Var} int $nD_{0-2}$ in \{-1,0,1\}
3. \textbf{Var} int $nS_{0-4}$ in \{0,1\}

\begin{align*}
&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
\end{align*}

foo CFG

CSP of foo

1. \textbf{Var} int $foo_0, x_0, y_0, z_0, max_0-3$
2. \textbf{Var} int $nD_{0-2}$ in \{-1,0,1\}
3. \textbf{Var} int $nS_{0-4}$ in \{0,1\}

\begin{align*}
&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
\end{align*}
## CPA-STDG

### Results

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

<table>
<thead>
<tr>
<th>Program</th>
<th>k-path</th>
<th># feasible Paths</th>
<th>CPA-STDG</th>
<th>PathCrawler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td># T. D.</td>
<td>T. (s)</td>
</tr>
<tr>
<td>tri_type</td>
<td>-</td>
<td>10</td>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>Sample</td>
<td>3</td>
<td>240</td>
<td>240</td>
<td>0.060</td>
</tr>
<tr>
<td>Merge</td>
<td>5</td>
<td>321</td>
<td>321</td>
<td>0.148</td>
</tr>
<tr>
<td>Merge</td>
<td>10</td>
<td>20,481</td>
<td>20,481</td>
<td>28.640</td>
</tr>
</tbody>
</table>

CPA-STDG is more efficient and effective than PathCrawler. The scalability issue remains an open research question.
### CPA-STDG Results

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

<table>
<thead>
<tr>
<th>Program</th>
<th>k-path</th>
<th># feasible Paths</th>
<th>CPA-STDG</th>
<th>PathCrawler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td># T. D.</td>
<td>T. (s)</td>
</tr>
<tr>
<td>tri_type</td>
<td>-</td>
<td>10</td>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>Sample</td>
<td>3</td>
<td>240</td>
<td>240</td>
<td>0.060</td>
</tr>
<tr>
<td>Merge</td>
<td>5</td>
<td>321</td>
<td>321</td>
<td>0.148</td>
</tr>
<tr>
<td>Merge</td>
<td>10</td>
<td>20,481</td>
<td>20,481</td>
<td><strong>28.640</strong></td>
</tr>
</tbody>
</table>

CPA-STDG is more efficient and effective than PathCrawler.
CPA-STDG

Results

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

<table>
<thead>
<tr>
<th>Program</th>
<th>k-path</th>
<th># feasible Paths</th>
<th>CPA-STDG</th>
<th>PathCrawler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td># T. D.</td>
<td>T. (s)</td>
</tr>
<tr>
<td>tri_type</td>
<td>-</td>
<td>10</td>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>Sample</td>
<td>3</td>
<td>240</td>
<td>240</td>
<td>0.060</td>
</tr>
<tr>
<td>Merge</td>
<td>5</td>
<td>321</td>
<td>321</td>
<td>0.148</td>
</tr>
<tr>
<td>Merge</td>
<td>10</td>
<td>20,481</td>
<td>20,481</td>
<td>28.640</td>
</tr>
</tbody>
</table>

CPA-STDG is more efficient and effective than PathCrawler.
CPA-STDG

Results

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

<table>
<thead>
<tr>
<th>Program</th>
<th>k-path</th>
<th># feasible Paths</th>
<th>CPA-STDG</th>
<th>PathCrawler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td># T. D.</td>
<td>T. (s)</td>
</tr>
<tr>
<td>tri_type</td>
<td>-</td>
<td>10</td>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>Sample</td>
<td>3</td>
<td>240</td>
<td>240</td>
<td>0.060</td>
</tr>
<tr>
<td>Merge</td>
<td>5</td>
<td>321</td>
<td>321</td>
<td>0.148</td>
</tr>
<tr>
<td>Merge</td>
<td>10</td>
<td>20,481</td>
<td>20,481</td>
<td>28.640</td>
</tr>
</tbody>
</table>

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

The Idea of CSB-STDG

- Test Candidates
- Fitness Evaluation
  - Selection
  - Crossover
  - Mutation
- Insertion
- Random Generator
- Test Data

CPA-STDG

CSB-STDG

- Search Based Software Test Data Generation

- The Idea of CSB-STDG
- Example: Unit Under Test Relaxation
- Constrained Population Generator (CPG)
- Constrained Evolution Operator (CEO)
- Evaluation
- Subjects
- Results

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG

CSB-STDG

Search Based Software Test Data Generation

The Idea of CSB-STDG

Example: Unit Under Test Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CB-FF

IG-PR-I0OCC

Conclusions
Search Based Software Test Data Generation

The Idea of CSB-STDG
Example: Unit Under Test Relaxation
Constrained Population Generator (CPG)
Constrained Evolution Operator (CEO)
Evaluation
Subjects
Results

Introduction
CPA-STDG
CSB-STDG
Search Based Software Test Data Generation

Test Candidates
Fitness Evaluation
Random Generator

Test Data
End?
Selection
Crossover
Mutation
Insertion

Random Generator
Test Candidates
Introduction

The Idea of CSB-STDG

Example: Unit Under Test

Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CB-FF

IG-PR-IOOCC

Conclusions

Search Based Software Test Data Generation

Test Candidates

Fitness Evaluation

Selection

Crossover

Mutation

Insertion

Random Generator

Test Data

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

Abdelilah Sakti

Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Search Based Software Test Data Generation

Random Generator

Test Candidates

Fitness Evaluation

Selection

Crossover

Mutation

Insertion

Test Data

CP-FF

IG-PR-IOOCC

Conclusions
The Idea of CSB-STDG

Search Space

CSP

Simple Constraints

Complex Constraints
The Idea of CSB-STDG

Search Space

CSP

Simple Constraints

Complex Constraints
The Idea of CSB-STDG

Search Space

CSP

Simple Constraints

Complex Constraints

Example: Unit Under Test
Relaxation
Constrained Population Generator (CPG)
Constrained Evolution Operator (CEO)
Evaluation
Subjects
Results

CB-FF
IG-PR-I00CC
Conclusions
The Idea of CSB-STDG

Search Space

CSP

Simple Constraints

Complex Constraints
The Idea of CSB-STDG

Example: Unit Under Test
Relaxation
Constrained Population Generator (CPG)
Constrained Evolution Operator (CEO)
Evaluation
Subjects
Results

CB-FF
IG-PR-IOOCC

Conclusions
Example: Unit Under Test Relaxation

```java
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

```java
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;
}
```
```
1 int foo(int x, int y)
2     if (y == R0; )
3         if (y > 0)
4             if (x == 10)
5                 return R1;
6         y = x << y;
7         x = y + x / y;
8     if (x > 0) && R2 > x)
9         return y / R3;
10     return 0;
11 }
```
Constrained Population Generator (CPG)

\[ RV_1 \text{ (integer)} \]

\[ RV_2 \text{ (float)} \]

\[ RV_2 \text{ (string)} \]

- \( s_{11} \)
- \( s_{12} \)
- \( s_{13} \)

- \( s_{21} \)

- \( s_{31} \)
- \( s_{32} \)
Automatic Test Data Generation using Constraint Programming and Search Based Software Engineering Techniques

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG

CSB-STDG
Search Based Software Test Data Generation
The Idea of CSB-STDG
Example: Unit Under Test Relaxation

Constrained Population Generator (CPG)
Constrained Evolution Operator (CEO)
Evaluation
Subjects
Results

CB-FF
IG-PR-I0OCC

Conclusions

Constrained Population Generator (CPG)

$RV_1$ (integer) $RV_2$ (float) $RV_2$ (string)

$s_{11}$ $s_{12}$ $s_{13}$ $s_{21}$ $s_{22}$ $s_{31}$ $s_{32}$

Initial Population

$s_{11}, s_{21}, s_{31}$
Constrained Population Generator (CPG)

$RV_1$ (integer) $RV_2$ (float) $RV_2$ (string)

$s_{11}$ $s_{12}$ $s_{13}$

$s_{21}$ $s_{22}$

$s_{31}$ $s_{32}$

Initial Population
Constrained Evolution Operator (CEO)

\[(x_v, y_v, z_v, ?, s2_v)\]

- **integer Solver**
- **float Solver**
- **string Solver**

- **Constrained Evolution Operator (CEO)**

\[(x_v, y_v, z_v, s1_{nv}, s2_{nv})\]

- **solvable ?**
Constrained Evolution Operator (CEO)

\((x_v, y_v, z_v, ?, s2_v)\)

Type ?

Integer Solver

Float Solver

String Solver

Solvable ?

\((x_v, y_v, z_v, s1_{nv}, s2_v)\)

\((x_v, y_v, z_v, s1_{nv}, s2_{nv})\)
Evaluation

Subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th># LOC</th>
<th># Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1244</td>
<td>38</td>
</tr>
<tr>
<td>BitSet</td>
<td>755</td>
<td>145</td>
</tr>
<tr>
<td>ArithmeticUtils</td>
<td>959</td>
<td>102</td>
</tr>
<tr>
<td>All</td>
<td>2958</td>
<td>285</td>
</tr>
</tbody>
</table>

analysed approaches

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

Results

![Graph showing branch coverage over time for different methods.](image)

- CPG
- CEO
- CPG+CEO
- eToc (SB-STDG)
- CPA-STDG (CB-STDG)

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Branch Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>100</td>
<td>70</td>
</tr>
<tr>
<td>150</td>
<td>70</td>
</tr>
<tr>
<td>200</td>
<td>80</td>
</tr>
<tr>
<td>250</td>
<td>80</td>
</tr>
<tr>
<td>300</td>
<td>90</td>
</tr>
</tbody>
</table>

Introduction

CPA-STDG

Search Based Software Test Data Generation

The Idea of CSB-STDG

Example: Unit Under Test Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CB-FF

IG-PR-I0OCC

Conclusions
Evaluation

Results

![Graph showing branch coverage over time for different methods: CPG, CEO, CPG+CEO, eToc (SB-STDG), CPA-STDG (CB-STDG). The graph indicates that CPG+CEO and CPA-STDG (CB-STDG) achieve higher branch coverage with time compared to CPG and CEO.](image-url)
Evaluation

Results

### CPA-STDG

- Search Based Software Test
- Data Generation

### CSB-STDG

- The Idea of CSB-STDG
- Example: Unit Under Test
- Relaxation
- Constrained Population Generator (CPG)
- Constrained Evolution Operator (CEO)

### Evaluation

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB-FF</td>
<td></td>
</tr>
<tr>
<td>IG-PR-I0OCC</td>
<td></td>
</tr>
</tbody>
</table>

### Conclusions
Evaluation Results

![Graph showing Branch Coverage % vs Time (s)]

- CPG
- CEO
- CPG+CEO
- eToc (SB-STDG)
- CPA-STDG (CB-STDG)
CSB-STDG

Introduction

The Idea of CSB-STDG

Example: Unit Under Test

Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CPA-STDG

Search Based Software Test Data Generation

The Idea of CSB-STDG

Example: Unit Under Test

Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Conclusions

Random Generator

Test Data

Fitness Evaluation

Selection

Crossover

Mutation

Insertion
Introduction

CPA-STDG

CSB-STDG

Search Based Software Test Data Generation

The Idea of CSB-STDG

Example: Unit Under Test Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CB-FF

IG-PR-IOOCC

Conclusions
Introduction

The Idea of CSB-STDG

Example: Unit Under Test

Relaxation

Constrained Population Generator (CPG)

Constrained Evolution Operator (CEO)

Evaluation

Subjects

Results

CB-FF

IG-PR-I00CC

Conclusions
Introduction

CPA-STDG

CSB-STDG

The Idea of CSB-STDG
Example: Unit Under Test Relaxation
Constrained Population Generator (CPG)
Constrained Evolution Operator (CEO)
Evaluation
Subjects

Results

CB-FF

IG-PR-IOOC

Conclusions
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 \textit{arity} and its \textit{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;
- \( DC(c) = B^2 \cdot \frac{1}{arity_c} + B \cdot (1 - \text{tightness}) + 1 \).
### Difficulty Coefficient (DC)

- DC is a possible representation of the hardness of a branch;
- \[ DC(c) = B^2 \cdot \frac{1}{\text{arity}_c} + B \cdot (1 - \text{tightness}) + 1. \]

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

**EA: Comparing all fitnesses on 440 test targets**

- Approach Level
- Symbolic Enhanced
- Symbolic Enhanced with levels
- Difficulty Coefficient
- Difficulty Level

The chart illustrates the evaluations across different approach levels and fitness functions, comparing them on 440 test targets.
Evaluation

SA Results

SA: Comparing all fitnesses on 440 test targets

- Approach Level
- Symbolic Enhanced
- Symbolic Enhanced with levels
- Difficulty Coefficient
- Difficulty Level

Branch Coverage %

Evaluations
CB-FF

The idea of CB-FF
Branch-Hardness Metrics
Branch-Hardness Fitness Functions
Evaluation
Subjects and configuration
EA Results
SA Results
IG-PR-IOOCC
Conclusions

Random Generator

Test Candidates

Fitness Evaluation

Test Data

Selection

Crossover

Mutation

Insertion

Random Generator

Test Candidates

Fitness Evaluation

Test Data

Selection

Crossover

Mutation

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

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG
CSB-STDG
CB-FF

The idea of CB-FF
Branch-Hardness Metrics
Branch-Hardness Fitness Functions
Evaluation
Subjects and configuration
EA Results

SA Results

IG-PR-IOOCC

Conclusions
CB-FF

The idea of CB-FF
Branch-Hardness Metrics
Branch-Hardness Fitness Functions
Evaluation
Subjects and configuration
EA Results
SA Results
IG-PR-IOOCC
Conclusions
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

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

- **Example**
- **Evaluation I**
- **Subjects**
- **Results**
- **JTExpert: A Large Scale Experimentation**
- **Exceptions-oriented Test-data Generation**
- **Subjects**
- **Results**
- **Conclusions**
IG-PR-IOOCC

Example

Source Code

```java
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){
    ...
}
}
```

Introduction

CPA-STDG

CSB-STDG

CB-FF

IG-PR-IOOCC

The idea of IG-PR-IOOCC

Evaluation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exceptions-oriented Test-data Generation

Subjects

Results

Conclusions
### Source Code

```
1 public class FooClass{
2    private int y;
3    private float z;
4    private String s2;
5    public void setY(int y1) {y=y1};
6    public void setZ(float z1) {z=z1};
7    public void setS2(String s) {s2=s};
8    public int fooCaller(int a, String str){
9        ...
10       foo(a,str);
11        ...
12    }
13    private int foo(int x, String s1){
14        ...
15    }
16    ...
17 }
```

### Static Analyses

MIn

---

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

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

**Introduction**

**CPA-STDG**

**CSB-STDG**

**CB-FF**

**IG-PR-IOOCC**

The idea of IG-PR-IOOCC

**Example**

**Evaluation I**

**Subjects**

**Results**

**JTExpert: A Large Scale Experimentation**

**Exceptions-oriented Test-data Generation**

**Subjects**

**Results**

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

Abdelilah Sakti
Supervised by Gilles Pesant and Yann-Gaël Guéhéneuc

Introduction

CPA-STDG
CSB-STDG
CB-FF
IG-PR-IOOCC
The idea of 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()
Abdelilah Sakti
Supervised by
Gilles Pesant and
Yann-Gaël Guéhéneuc

IG-PR-IOOCC
Example

Source Code

```java
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

```java
MIn
FooClass()
```
IG-PR-IOOCC

Example

Source Code

```java
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()

y

setY
Source Code

```java
public class FooClass{
    private int y;
    private float z;
    private String s;

    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 s){
        ...
    }
}
```

Static Analyses

MIn

FooClass()

y

setY

Z
Source Code

```java
1 public class FooClass{
2     private int y;
3     private float z;
4     private String s2;
5     public void setY(int y1) {y=y1};
6     public void setZ(float z1) {z=z1};
7     public void setS2(String s) {s2=s};
8     public int fooCaller(int a, String str){
9         ...
10         foo(a,str);
11         ...
12     }
13     private int foo(int x, String s1){
14         ...
15     }
16     ...
17 }
```

Static Analyses

- MIn
- FooClass()
- y
- setY
- z
- setZ
IG-PR-IOOCC

Example

Source Code

```java
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()
- y
- setY
- z
- setZ
- s2
IG-PR-IOOCC

Example

Source Code

```java
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()

setY

setZ

setS2

y

z

s2
IG-PR-IOOCC

Example

Source Code

```java
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

- `Mln`: FooClass()
- `y`: setY
- `z`: setZ
- `s2`: setS2
- `foo`: foo
IG-PR-IOOCC

Example

Source Code

```java
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()
   y    setY
   z    setZ
   s2   setS2
   foo  fooCaller
```
Evaluation I

Subjects

Compared systems

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

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

JTExpert' components analysis over time (all libraries)

- 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)**

![Graph showing branch coverage over search time](image)

- **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)

- 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
The idea of IG-PR-IOOCC

Example

Evaluation I

Subjects

Results

JTExpert: A Large Scale Experimentation

Exceptions-oriented Test-data Generation

Subjects

Results

44 / 51
JTExpert: A Large Scale Experimentation

Exceptions-oriented Test-data Generation

```java
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

1  public class FooClass{
2  private int y;
3  private float z;
4  private String s2;
5  public void setY(int y1) {y=y1};
6  public void setZ(float z1) {z=z1};
7  public void setS2(String s) {s2=s};
8  public int fooCaller(int a, String str){
9       foo(a,str);...
10  }
11  }
12  
13  private int foo(int x, String s1){
14       if(y==z)
15               if(y>0)
16                   if(x==10)
17                       return s1.length()+s2.length();
18                  y= x<<y;
19                  x=y+x/y;
20                  String s=s1+s2;
21       if((s.equals("OK") && (x>0) && s.length()>x)
22                       return y/s1.length();
23  }
24  45 / 51
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

<table>
<thead>
<tr>
<th>Exception</th>
<th>Alerted</th>
<th>Alerted in different</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lines</td>
</tr>
<tr>
<td>NullPointerException</td>
<td>13,663</td>
<td>3,604</td>
</tr>
<tr>
<td>ClassCastException</td>
<td>658</td>
<td>213</td>
</tr>
<tr>
<td>NoClassDefFoundError</td>
<td>602</td>
<td>87</td>
</tr>
<tr>
<td>IllegalArgumentException</td>
<td>326</td>
<td>104</td>
</tr>
<tr>
<td>ArrayIndexOutOfBoundsException</td>
<td>296</td>
<td>119</td>
</tr>
<tr>
<td>Fatal Errors</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Other Exceptions</td>
<td>492</td>
<td>193</td>
</tr>
<tr>
<td><strong>All Exceptions</strong></td>
<td><strong>16,046</strong></td>
<td><strong>4,322</strong></td>
</tr>
</tbody>
</table>
## JTExpert: A Large Scale Experimentation

### Results

<table>
<thead>
<tr>
<th>Exception</th>
<th>Alerted</th>
<th>Alerted in different</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lines</td>
</tr>
<tr>
<td>NullPointerException</td>
<td>13,663</td>
<td>3,604</td>
</tr>
<tr>
<td>ClassCastException</td>
<td>658</td>
<td>213</td>
</tr>
<tr>
<td>NoClassDefFoundError</td>
<td>602</td>
<td>87</td>
</tr>
<tr>
<td>IllegalArgumentException</td>
<td>326</td>
<td>104</td>
</tr>
<tr>
<td>ArrayIndexOutOfBoundsException</td>
<td>296</td>
<td>119</td>
</tr>
<tr>
<td><strong>Fatal Errors</strong></td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td><strong>Other Exceptions</strong></td>
<td>492</td>
<td>193</td>
</tr>
<tr>
<td><strong>All Exceptions</strong></td>
<td>16,046</td>
<td>4,322</td>
</tr>
</tbody>
</table>
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;

Exception-oriented Test-data Generation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exception-oriented Test-data Generation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exception-oriented Test-data Generation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exception-oriented Test-data Generation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exception-oriented Test-data Generation

Subjects

Results

JTExpert: A Large Scale Experimentation

Exception-oriented Test-data Generation

Subjects

Results
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.

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.
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.
Start fitter and finish better.