A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs

René Just[1], Franz Schweiggert[1], and Gregory M. Kapfhammer[2]

[1]Ulm University, Germany

[2]Allegheny College, USA

Saarland University, Saarbrücken, Germany

January 24, 2012

Overview of MAJOR

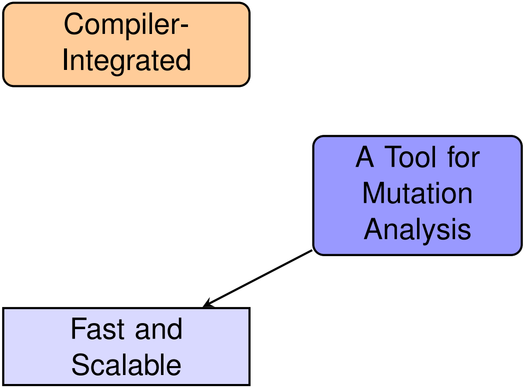
A Tool for
Mutation
Analysis

Overview of MAJOR

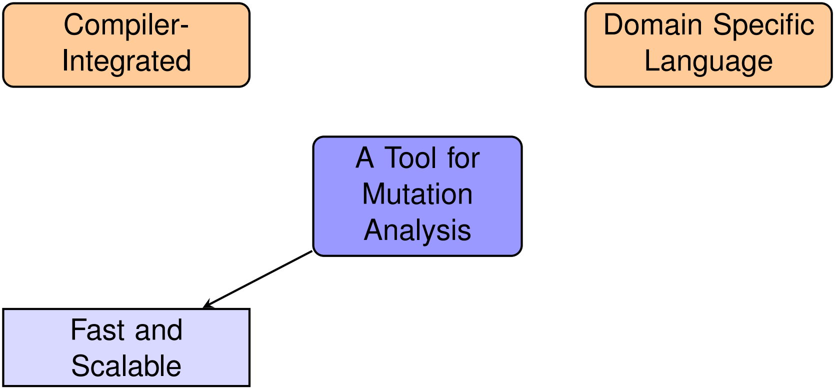
Compiler-Integrated

A Tool for Mutation Analysis

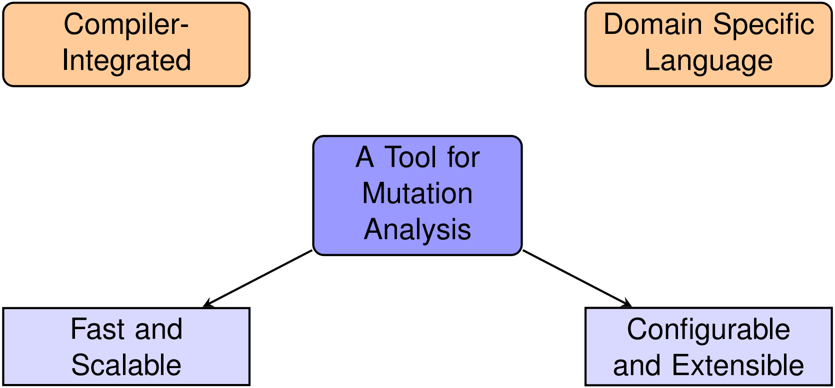
Overview of MAJOR



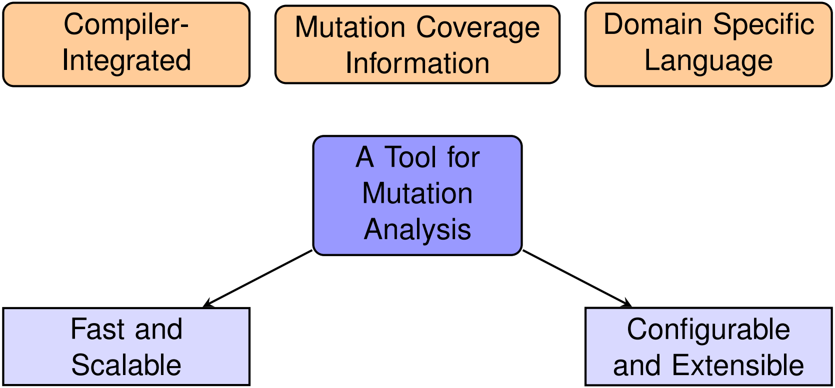
Overview of MAJOR



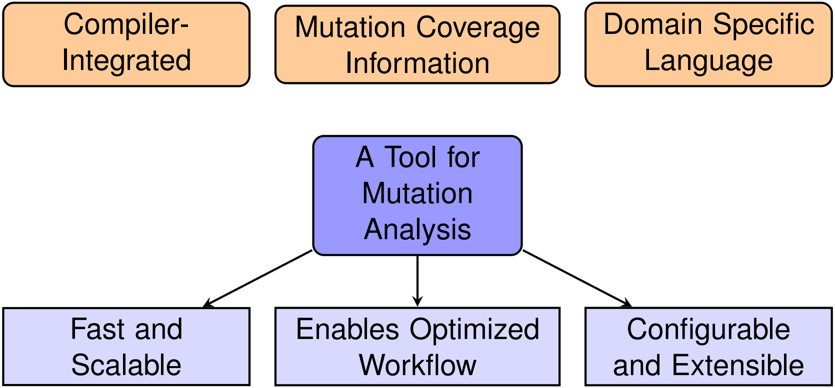
Overview of MAJOR



Overview of MAJOR



Overview of MAJOR



Overview of Mutation Analysis

Mutation Analysis

Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test

Mutation Analysis

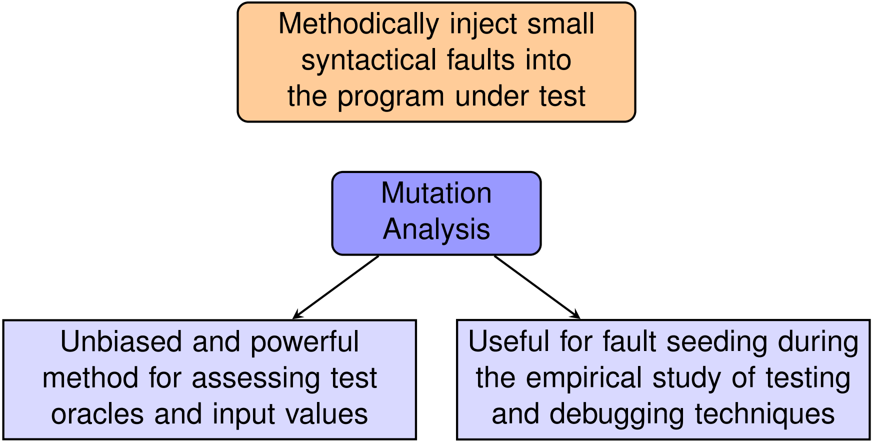
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test

Mutation Analysis

Unbiased and powerful method for assessing test oracles and input values

Overview of Mutation Analysis



Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Overview of Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

⟹

- `y = a - x;`
- `y = a + x;`
- `y = a / x;`

```
public int max(int a, int b){
   int max = a;

   if(b>a){
     max=b;
   }

   return max;
}
```

⟹

- `if(b < a)`
- `if(b != a)`
- `if(b == a)`

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;


    y = a * x;


    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x / y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Working Example

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

Mutants that are not executed cannot be killed

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

Mutants that are not executed cannot be killed

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0 &&
         COVERED(1,3))?
            a * x : a * x;

    y += b;

    return y;
}
```

Mutants that are not executed cannot be killed

Determine covered mutants with additional instrumentation

Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0 &&
         COVERED(1,3))?
            a * x : a * x;

    y += b;

    return y;
}
```

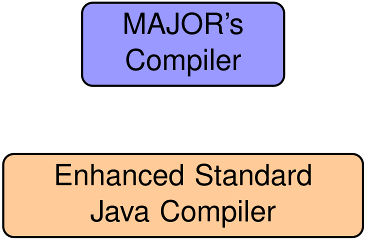
Mutants that are not executed cannot be killed

Determine covered mutants with additional instrumentation

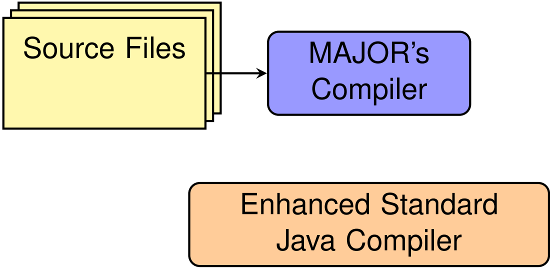
Only execute and investigate the covered mutants

MAJOR's Compiler

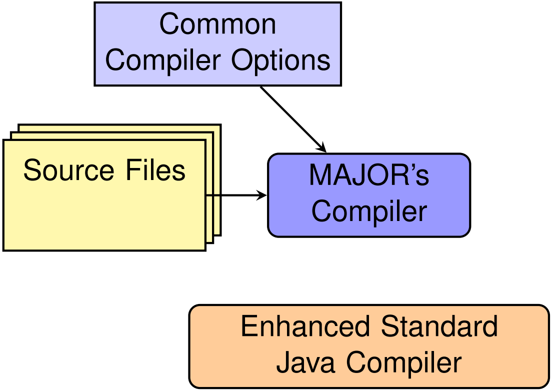
MAJOR's Compiler



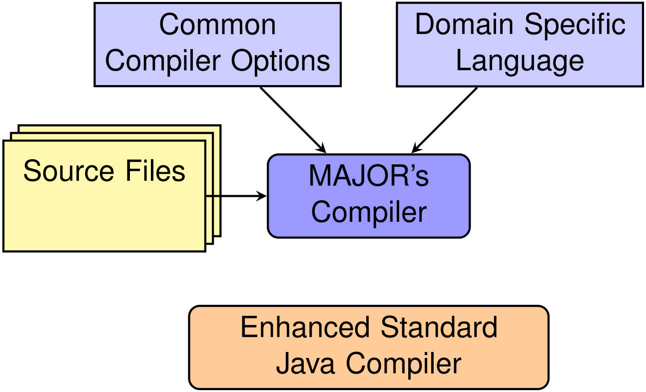
MAJOR's Compiler



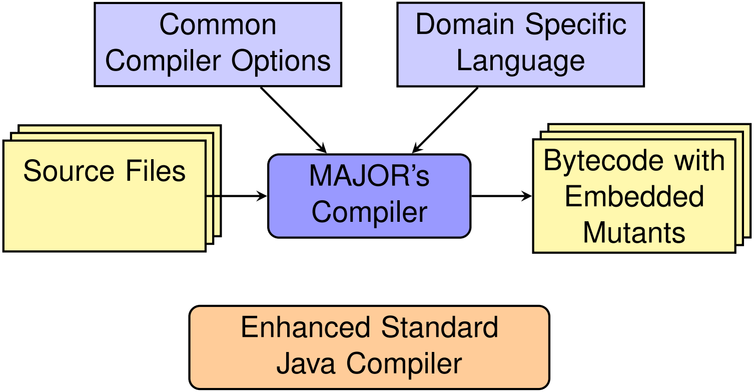
MAJOR's Compiler



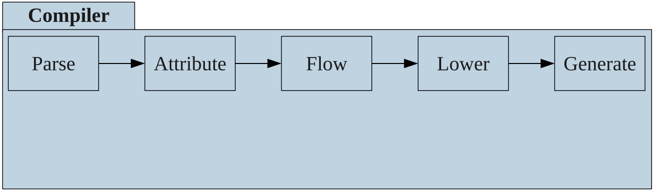
MAJOR's Compiler



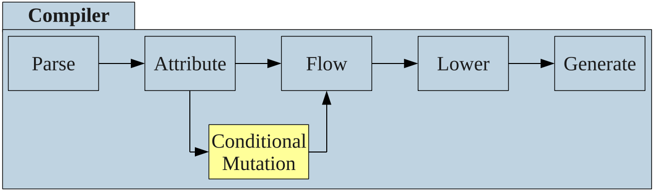
MAJOR's Compiler



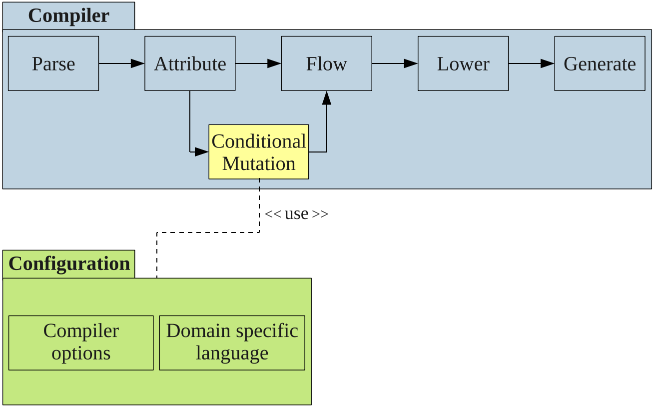
Integration into the Java Compiler



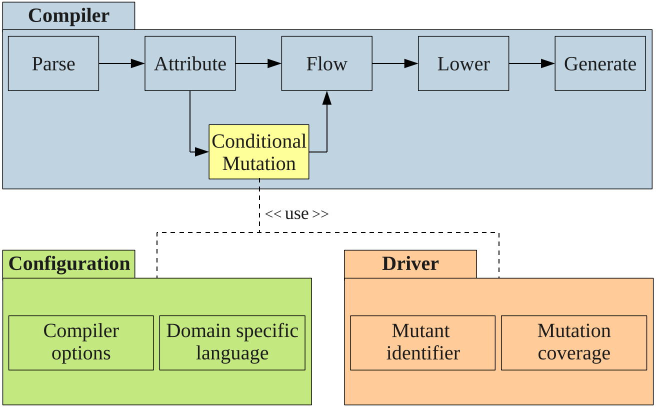
Integration into the Java Compiler



Integration into the Java Compiler



Integration into the Java Compiler



Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Java compiler developer mailing list

Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Q: *"I was wondering if there is a documentation for the Tree transformation. Is the method documentation is all that is available?"*

→ Java compiler developer mailing list

Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Q: *"I was wondering if there is a documentation for the Tree transformation. Is the method documentation is all that is available?"*

Java compiler developer mailing list

A: *"That and looking at the examples embodied in the existing code."*

MAJOR's Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

MAJOR's Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

MAJOR's Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
   BIN(&&) -> listCOR;
   BIN(||) -> listCOR;
   COR;
   LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

Define own mutation operator groups

MAJOR's Domain Specific Language

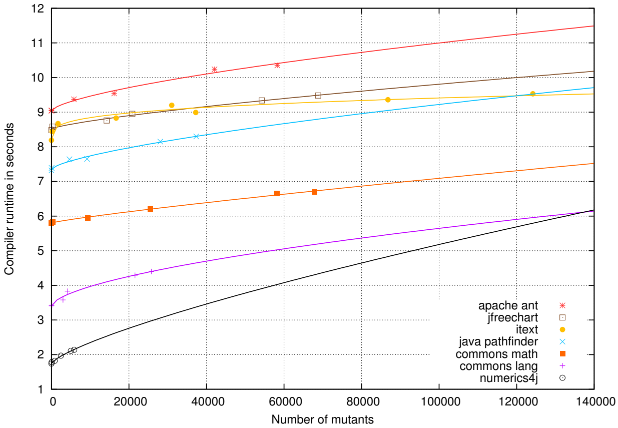
```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
   BIN(&&) -> listCOR;
   BIN(||) -> listCOR;
   COR;
   LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

Define own mutation operator groups

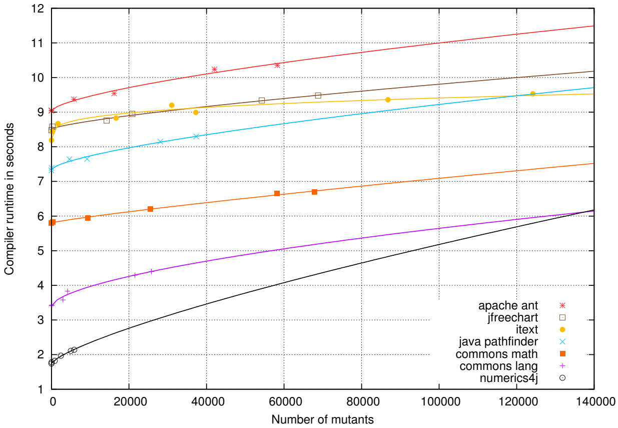
Enable operators for a specific package, class, or method

Performance Analysis



- Overhead for generating and compiling mutants is negligible

Performance Analysis



- Overhead for generating and compiling mutants is negligible

Performance Analysis

Application	*Mutants*	*Runtime of test suite* original	instrumented wcs	instrumented wcs+cov	*Memory consumption* original	instrumented
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	*Mutants*	*Runtime of test suite*			*Memory consumption*	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	Mutants	Runtime of test suite original	Runtime instrumented wcs	Runtime instrumented wcs+cov	Memory consumption original	Memory consumption instrumented
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

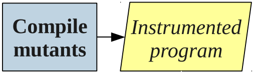
- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Performance Analysis

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

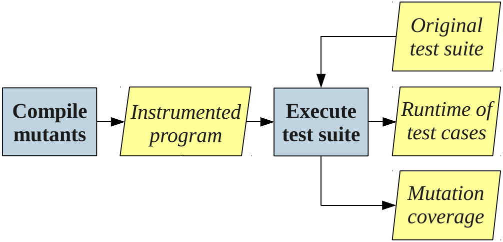
- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Optimized Mutation Analysis Process



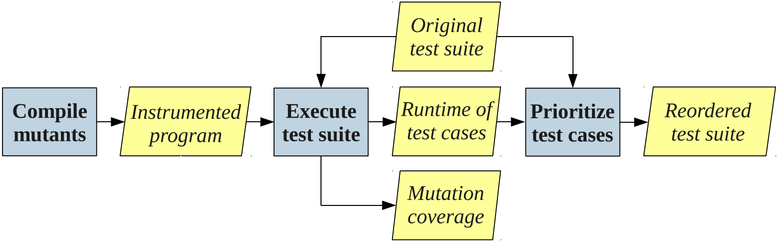
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



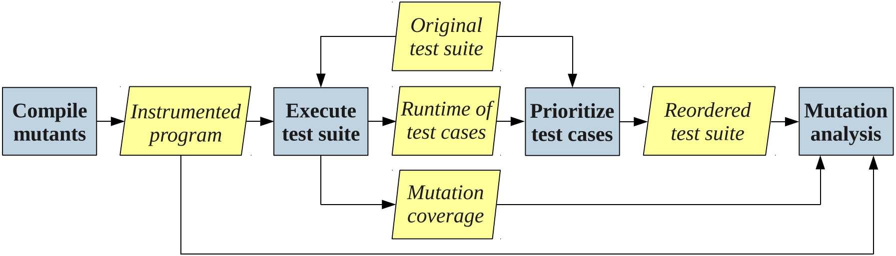
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



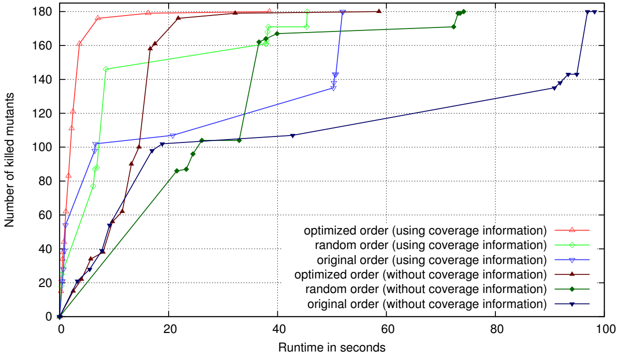
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



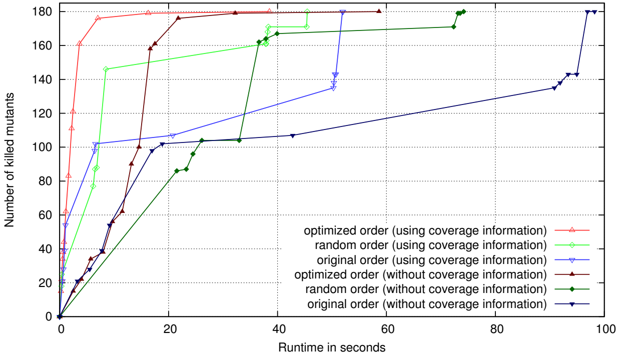
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Performance Analysis



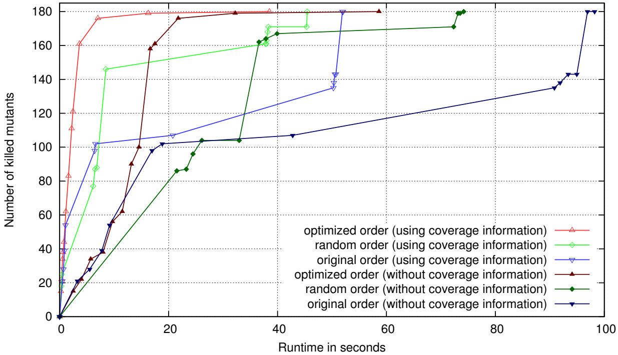
- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Performance Analysis



- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Performance Analysis



- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Furnishes its own domain specific language
- Provides mutation coverage information

Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Furnishes its own domain specific language
- Provides mutation coverage information

Characteristics of MAJOR:

- Fast and scalable technique
- Configurable and extensible mutation tool
- Enables an optimized workflow for mutation analysis

MAJOR's Compiler

So, what comes next in this talk?

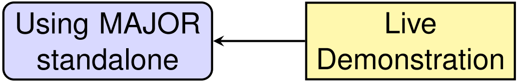
MAJOR's Compiler

So, what comes next in this talk?

Live Demonstration

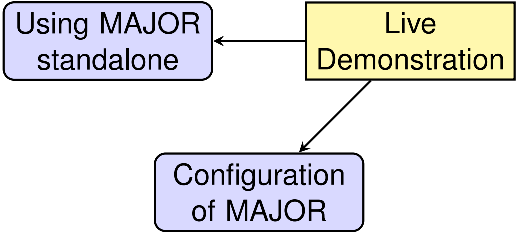
MAJOR's Compiler

So, what comes next in this talk?

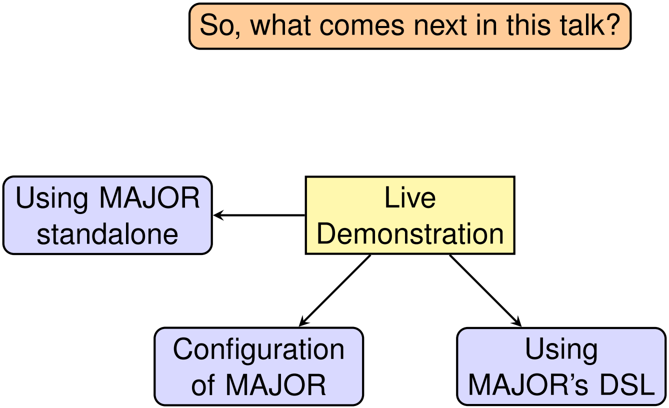


MAJOR's Compiler

So, what comes next in this talk?

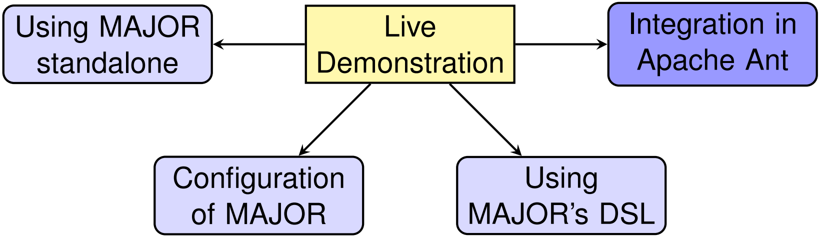


MAJOR's Compiler



MAJOR's Compiler

So, what comes next in this talk?



A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs

Thank you for your attention!

Questions?