**MAJOR: An Efficient Technique for Mutation Analysis in a Java Compiler**

René Just¹ and Gregory M. Kapfhammer²

¹Department of Applied Information Processing, Ulm University
²Department of Computer Science, Allegheny College

---

**IMPORTANT CONTRIBUTIONS**
- Enhanced the Java 6 Standard Edition compiler
- Simple compiler options enable the mutation analysis
- Easily applicable in all Java development environments
- Effectively reduces mutant generation time to a minimum

**CONDITIONAL MUTATION**
- Transforms the program’s abstract syntax tree (AST)
- Encapsulates the mutations within conditional statements

**SUPPORTED FEATURES**
- Enables second and higher order mutation analysis
- Determination of mutation coverage by running the original code
- Configurable mutation operators by means of compiler options

**MUTATION COVERAGE**
- It is impossible to kill a mutant if it is not reached and executed
- Additional instrumentation determines the covered mutations
- Mutation coverage is only examined if the tests execute the original code
- An external driver efficiently records the covered mutations as ranges
- Only those mutants covered by a test case are executed

---

**PERFORMANCE EVALUATION**

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants generated</th>
<th>Coverd</th>
<th>killed</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wcs</td>
<td>wcs+cov</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>20,144</td>
<td>120000</td>
<td>4.3 4.8 5.0</td>
<td>559 813</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>28,118</td>
<td>21,084</td>
<td>331.0 335.0 346.0</td>
<td>237 293</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>29,485</td>
<td>12,788</td>
<td>15.0 18.0 23.0</td>
<td>220 303</td>
</tr>
<tr>
<td>javapathfinder</td>
<td>124,184</td>
<td>12,793</td>
<td>4,546</td>
<td>217 225</td>
<td>182 217</td>
</tr>
<tr>
<td>commons math</td>
<td>37,331</td>
<td>5,869</td>
<td>4,900</td>
<td>5,869 4,900 401</td>
<td>153 225</td>
</tr>
<tr>
<td>commons lang</td>
<td>67,895</td>
<td>60,258</td>
<td>12,793</td>
<td>67,895 54,326 44,084</td>
<td>153 225</td>
</tr>
<tr>
<td>numerics</td>
<td>5,869</td>
<td>5,869</td>
<td>4,900</td>
<td>5,869 4,900 401</td>
<td>153 225</td>
</tr>
</tbody>
</table>

---

**IMPLEMENTATION DETAILS**
- A separate package modularly extends the compiler
- Mutation operators configurable with enhanced \(-x\) options
- AST transformation implemented by means of the visitor pattern
- AST transformation implemented by means of the visitor pattern

---

**OPTIMIZED WORKFLOW**
- Comparison of MAJOR with related techniques and tools such as mJava, Javalanche, and Jumble
- Further runtime optimizations by balancing the AST
- Implementation of several new mutation operators
- Domain specific language for specifying mutation operators
- Integration of conditional mutation into a C/C++ compiler

---

**FUTURE WORK**
- Comparison of MAJOR with related techniques and tools such as mJava, Javalanche, and Jumble
- Further runtime optimizations by balancing the AST
- Implementation of several new mutation operators
- Domain specific language for specifying mutation operators
- Integration of conditional mutation into a C/C++ compiler