



The
University
Of
Sheffield.

COMBINING MULTIPLE COVERAGE CRITERIA IN SEARCH-BASED UNIT TEST GENERATION

SSBSE 2015

José Miguel Rojas

j.rojas@sheffield.ac.uk

Joint work with José Campos, Mattia Vivanti, Gordon Fraser and Andrea Arcuri

AUTOMATED UNIT TEST GENERATION



JUnit

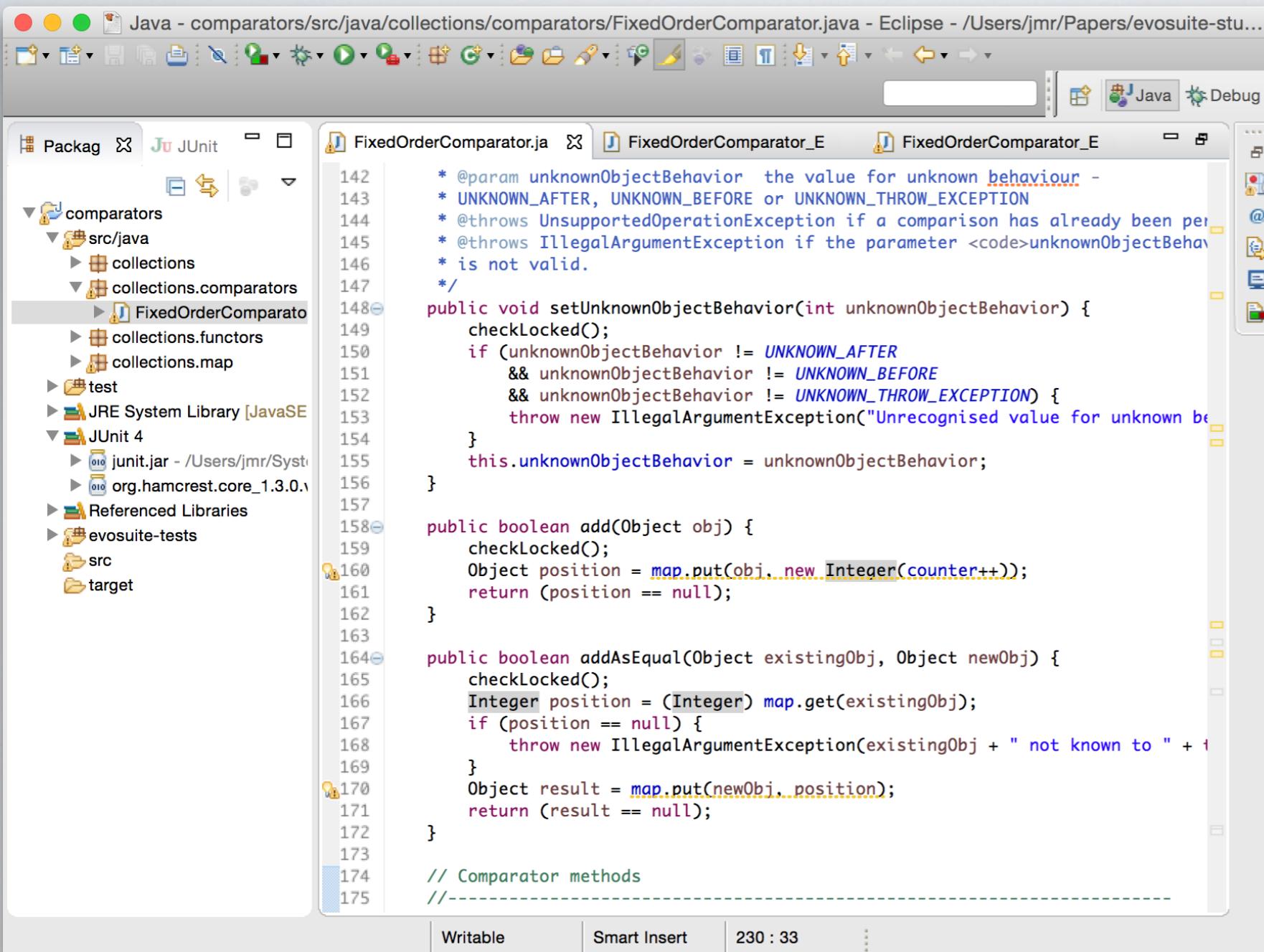
Class under Test

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Java - comparators/src/java/collections/comparators/FixedOrderComparator.java - Eclipse - /Users/jmr/Papers/evosuite-stu...
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and code navigation.
- Left Sidebar:** Package Explorer view showing the project structure:
 - Packages: comparators, src/java (collections, comparators), test, JRE System Library [JavaSE], JUnit 4, Referenced Libraries, evosuite-tests, src, target.
 - File: FixedOrderComparator.java is selected.
- Central Editor:** The code editor displays the `FixedOrderComparator.java` file. The code implements a comparator that stores objects in a map and returns their position. It includes methods for setting unknown object behavior and adding objects. The code uses annotations like `@param`, `@throws`, and `@throws`.
- Right Sidebar:** Standard Eclipse sidebar with icons for file operations, search, and code navigation.

```
* @param unknownObjectBehavior the value for unknown behaviour -  
* UNKNOWN_AFTER, UNKNOWN_BEFORE or UNKNOWN_THROW_EXCEPTION  
* @throws UnsupportedOperationException if a comparison has already been per-  
* @throws IllegalArgumentException if the parameter <code>unknownObjectBehav-  
* is not valid.  
*/  
public void setUnknownObjectBehavior(int unknownObjectBehavior) {  
    checkLocked();  
    if (unknownObjectBehavior != UNKNOWN_AFTER  
        && unknownObjectBehavior != UNKNOWN_BEFORE  
        && unknownObjectBehavior != UNKNOWN_THROW_EXCEPTION) {  
        throw new IllegalArgumentException("Unrecognised value for unknown be-  
    }  
    this.unknownObjectBehavior = unknownObjectBehavior;  
}  
  
public boolean add(Object obj) {  
    checkLocked();  
    Object position = map.put(obj, new Integer(counter++));  
    return (position == null);  
}  
  
public boolean addAsEqual(Object existingObj, Object newObj) {  
    checkLocked();  
    Integer position = (Integer) map.get(existingObj);  
    if (position == null) {  
        throw new IllegalArgumentException(existingObj + " not known to " + i)  
    }  
    Object result = map.put(newObj, position);  
    return (result == null);  
}  
  
// Comparator methods  
//-----
```

Class under Test



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Java - comparators/src/java/collections/comparators/FixedOrderComparator.java - Eclipse - /Users/jmr/Papers/evosuite-stu...
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and code navigation.
- Left Sidebar (Package Explorer):** Shows the project structure:
 - comparators
 - src/java
 - collections
 - comparators
 - FixedOrderComparator
 - test
 - JRE System Library [JavaSE]
 - JUnit 4
 - Referenced Libraries
 - evosuite-tests
 - src
 - target
- Central Editor Area:** Displays the `FixedOrderComparator.java` file content. The code implements a comparator that stores objects in a map and returns their position. It includes methods for setting unknown object behavior and adding objects. Error-prone code segments are highlighted in yellow.
- Right Sidebar:** Shows icons for code analysis, refactoring, and other development tools.

Automated
Unit Test
Generation

Class under Test

```
142 * @param unknownObjectBehavior the value for unknown behaviour -  
143 * UNKNOWN_AFTER, UNKNOWN_BEFORE or UNKNOWN_THROW_EXCEPTION  
144 * @throws UnsupportedOperationException if a comparison has already been per-  
145 * @throws IllegalArgumentException if the parameter <code>unknownObjectBehav-  
146 * is not valid.  
147 */  
148 public void setUnknownObjectBehavior(UNKNOWN unknownObjectBehavior)  
149     checkLocked();  
150     if (CUnknownObjectBehavior != null  
151         && unknownObjectBehavior != CUnknownObjectBehavior  
152         && unknownObjectBehavior != UNKNOWN_THROW_EXCEPTION)  
153         throw new UnsupportedOperationException("Unknown behavior");  
154     this.unknownObjectBehavior = unknownObjectBehavior;  
155 }  
156  
157 public boolean add(Object object)  
158     checkLocked();  
159     Object position = object;  
160     return (position != null);  
161 }  
162  
163 public boolean addAtPosition(Integer position)  
164     checkLocked();  
165     Integer previousPosition = position;  
166     if (position == null || position < 0)  
167         throw new IllegalArgumentException("Position must be non-null and non-negative");  
168     Object result = null;  
169     if (previousPosition == null)  
170         result = object;  
171     else  
172         result = position;  
173     return (result != null);  
174 }  
175  
// Comparator methods  
//-----
```

```
2+ * This file was automatically generated by EvoSuite.  
3+  
4+ package collections.comparators;  
5+  
6+ import static org.junit.Assert.assertEquals;  
7+  
8+ @RunWith(EvoRunner.class) @EvoRunnerParameters(useVNET = true)  
9+ public class FixedOrderComparator_ESTest extends FixedOrderComparator_ESTest_sca-  
10+  
11+     @Test  
12+     public void test00() throws Throwable {  
13+         Object[] objectArray0 = new Object[5];  
14+         Object object0 = new Object();  
15+         objectArray0[1] = object0;  
16+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
17+         fixedOrderComparator0.compare(objectArray0[1], object0);  
18+         boolean boolean0 = fixedOrderComparator0.isLocked();  
19+         assertTrue(boolean0);  
20+     }  
21+  
22+     @Test  
23+     public void test01() throws Throwable {  
24+         Object[] objectArray0 = new Object[5];  
25+         Object object0 = new Object();  
26+         Object object1 = new Object();  
27+         objectArray0[3] = object1;  
28+         objectArray0[4] = object0;  
29+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
30+         int int0 = fixedOrderComparator0.compare(object1, object0);  
31+         assertTrue(fixedOrderComparator0.isLocked());  
32+         assertEquals(-1, int0);  
33+     }  
34+  
35+     @Test  
36+     public void test02() throws Throwable {  
37+         Object[] objectArray0 = new Object[5];  
38+         Object object0 = new Object();  
39+         Object object1 = new Object();  
40+         objectArray0[1] = object1;  
41+         objectArray0[2] = object0;  
42+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
43+         int int0 = fixedOrderComparator0.compare(object1, object0);  
44+         assertTrue(fixedOrderComparator0.isLocked());  
45+         assertEquals(-1, int0);  
46+     }  
47+  
48+     @Test
```

Automated
Unit Test
Generation

Test Suite

Class under Test

```
142 * @param unknownObjectBehavior the value for unknown behaviour -  
143 * UNKNOWN_AFTER, UNKNOWN_BEFORE or UNKNOWN_THROW_EXCEPTION  
144 * @throws UnsupportedOperationException if a comparison has already been per-  
145 * @throws IllegalArgumentException if the parameter <code>unknownObjectBehav-  
146 * is not valid.  
147 */  
148 public void setUnknownObjectBehavior(UNKNOWN unknownObjectBehavior)  
149     checkLocked();  
150     if (CUnknownObjectBehavior != unknownObjectBehavior)  
151         && unknownObjectBehavior != UNKNOWN_THROW_EXCEPTION  
152         throw new UnsupportedOperationException("Unknown object behavior");  
153     this.unknownObjectBehavior = unknownObjectBehavior;  
154 }  
155  
156 public boolean add(Object object)  
157     checkLocked();  
158     Object position = positionFor(object);  
159     return (position != null);  
160 }  
161  
162 public boolean addAt(Integer position, Object object)  
163     checkLocked();  
164     Integer previousPosition = positionFor(object);  
165     if (position == previousPosition)  
166         throw new UnsupportedOperationException("Cannot add at same position");  
167     Object result = addAt(position, object);  
168     return (result != null);  
169 }  
170  
171 // Comparator methods  
172 //-----  
173  
174 //-----  
175
```

```
2+ * This file was automatically generated by EvoSuite.  
3+  
4+ package collections.comparators;  
5+  
6+ import static org.junit.Assert.assertEquals;  
7+  
8+ @RunWith(EvoRunner.class) @EvoRunnerParameters(useVNET = true)  
9+ public class FixedOrderComparator_ESTest extends FixedOrderComparator_ESTest_sca-  
10+  
11+     @Test  
12+     public void test00() throws Throwable {  
13+         Object[] objectArray0 = new Object[5];  
14+         Object object0 = new Object();  
15+         objectArray0[1] = object0;  
16+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
17+         fixedOrderComparator0.compare(objectArray0[1], object0);  
18+         boolean boolean0 = fixedOrderComparator0.isLocked();  
19+         assertTrue(boolean0);  
20+     }  
21+  
22+     @Test  
23+     public void test01() throws Throwable {  
24+         Object[] objectArray0 = new Object[5];  
25+         Object object0 = new Object();  
26+         Object object1 = new Object();  
27+         objectArray0[3] = object1;  
28+         objectArray0[4] = object0;  
29+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
30+         int int0 = fixedOrderComparator0.compare(object1, object0);  
31+         assertTrue(fixedOrderComparator0.isLocked());  
32+         assertEquals(-1, int0);  
33+     }  
34+  
35+     @Test  
36+     public void test02() throws Throwable {  
37+         Object[] objectArray0 = new Object[5];  
38+         Object object0 = new Object();  
39+         Object object1 = new Object();  
40+         objectArray0[1] = object1;  
41+         objectArray0[2] = object0;  
42+         FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objectArray0);  
43+         int int0 = fixedOrderComparator0.compare(object1, object0);  
44+         assertTrue(fixedOrderComparator0.isLocked());  
45+         assertEquals(-1, int0);  
46+     }  
47+  
48+     @Test
```

Automated
Unit Test
Generation

Optimised for
Branch Coverage

Test Suite

Class under Test

The screenshot shows the Eclipse IDE interface with two open files:

- FixedOrderComparator.java**: Contains methods for setting unknown object behavior, adding objects, and comparing them.
- FixedOrderComparator_ESTest.java**: A generated test class for the comparator.

The code in **FixedOrderComparator.java** includes annotations for parameters and exceptions, and several methods with logic for handling unknown object behaviors and adding objects to a map.

```
142 * @param unknownObjectBehavior the value for unknown behaviour -  
143 * UNKNOWN_AFTER, UNKNOWN_BEFORE or UNKNOWN_THROW_EXCEPTION  
144 * @throws UnsupportedOperationException if a comparison has already been per-  
145 * @throws IllegalArgumentException if the parameter <code>unknownObjectBehav-  
146 * is not valid.  
147 */  
148 public void setUnknownObjectBehavior(int unknownObjectBehavior) {  
149     checkLocked();  
150     if (UnknownObjectBehavior != UNKNOWN_AFTER  
151         && UnknownObjectBehavior != UNKNOWN_BEFORE  
152         && UnknownObjectBehavior != UNKNOWN_THROW_EXCEPTION) {  
153         throw new IllegalArgumentException("Unrecognised value for unknown ob-  
154     }  
155     this.unknownObjectBehavior = unknownObjectBehavior;  
156 }  
157  
158 public boolean add(Object obj) {  
159     checkLocked();  
160     Object position = map.put(obj, new Integer(counter++));  
161     return (position == null);  
162 }  
163  
164 public boolean addAsEqual(Object existingObj, Object newObj) {  
165     checkLocked();  
166     Integer position = (Integer) map.get(existingObj);  
167     if (position == null) {  
168         throw new IllegalArgumentException(existingObj + " not known to " +  
169     }  
170     Object result = map.put(newObj, position);  
171     return (result == null);  
172 }  
173  
174 // Comparator methods  
175
```

The test class **FixedOrderComparator_ESTest.java** contains test cases for the comparator's methods, using assertions to verify the behavior.

```
38 Object object0 = new Object();  
39 Object object1 = new Object();  
40 objectArray0[3] = object1;  
41 objectArray0[4] = object0;  
42 FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(object-  
43 int int0 = fixedOrderComparator0.compare(object1, object0);  
44 assertEquals(fixedOrderComparator0.isLocked(),  
45 assertEquals(-1, int0);  
46 }  
47  
48 @Test
```

Automated
Unit Test
Generation

Optimised for
Branch Coverage

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class ArrayList
public int set(int idx,
               int element){
    checkRange(idx);
    incrModCount();
    int oldval = _data[idx];
    _data[idx] = element;
    return oldval;
}
```

```
@Test
void test9(){
    ArrayList l =
        new ArrayList();
    // Undeclared exception!
    try {
        int int0 = l.set(20, 20);
        fail("Expecting IOOBExc");
    } catch(IOOBExc e) {
        // Should be at least 0
        // and less than 0,
        // found 20
    }
}
```

Class Under Test

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class ArrayList
public int set(int idx,
               int element){
    checkRange(idx);
    incrModCount();
    int oldval = _data[idx];
    _data[idx] = element;
    return oldval;
}
```

```
@Test
void test9(){
    ArrayList l =
        new ArrayList();
    // Undeclared exception!
    try {
        int int0 = l.set(20, 20);
        fail("Expecting IOOBExc");
    } catch(IOOBExc e) {
        // Should be at least 0
        // and less than 0,
        // found 20
    }
}
```

Class Under Test

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class Complex
public Complex log(){
    if (isNaN) return NaN;
    double r = log(abs());
    double i;
    i = atan2(imaginary, real); }
    return createComplex(r, i);
}
public Complex pow(double x){
    Complex c = this.log();
    return c.multiply(x).exp();
}
```

```
@Test
void test1() {
    Complex c0 = new Complex(NaN);
    Complex c1 = c0.pow(NaN);
    double d = c1.getArgument();
    assertEquals(NaN, d, 0.01);
}

@Test
void test2() {
    Complex c0 = Complex.ZERO;
    Complex c1 = c0.pow(c0);
    assertFalse(c1.isInfinite());
    assertTrue(c1 isNaN());
}
```

Class Under Test

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

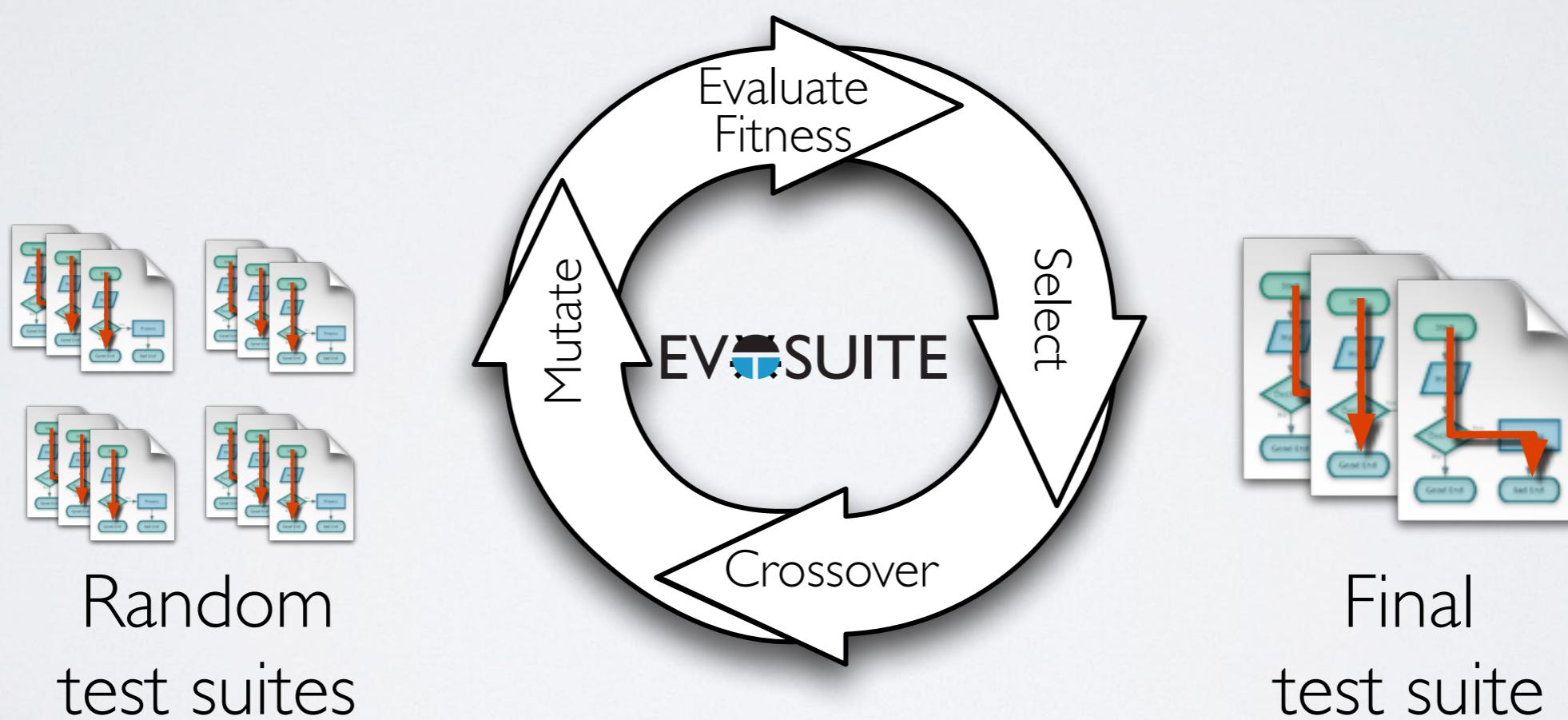
```
// Class Complex
public Complex log(){
    if (isNaN) return NaN;
    double r = log(abs());
    double i;
    i = atan2(imaginary, real);
    return createComplex(r, i);
}
public Complex pow(double x){
    Complex c = this.log();
    return c.multiply(x).exp();
}
```

Class Under Test

```
@Test
void test1() {
    Complex c0 = new Complex(NaN);
    Complex c1 = c0.pow(NaN);
    double d = c1.getArgument();
    assertEquals(NaN, d, 0.01);
}
@Test
void test2() {
    Complex c0 = Complex.ZERO;
    Complex c1 = c0.pow(c0);
    assertFalse(c1.isInfinite());
    assertTrue(c1 isNaN());
}
```

Test Suite

WHOLE TEST SUITE GENERATION



FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
throws NullPointerException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



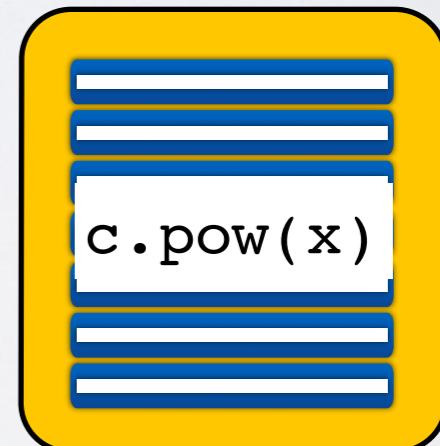
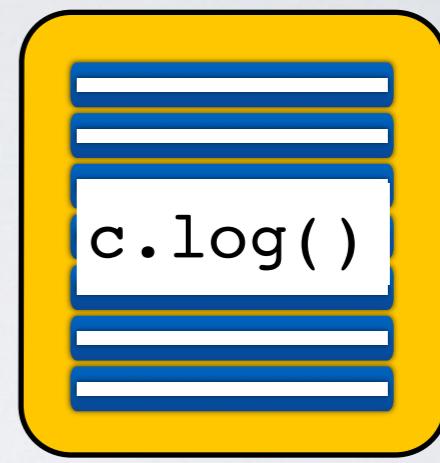
⋮



Method Coverage

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

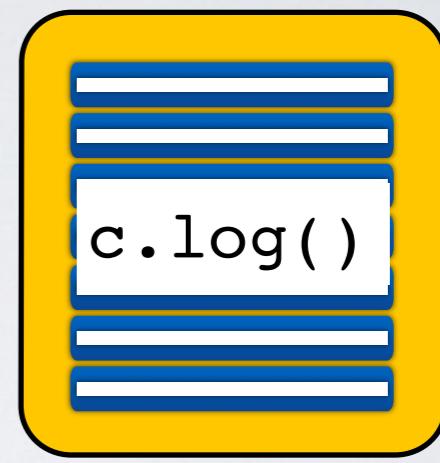
```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
throws NullPointerException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



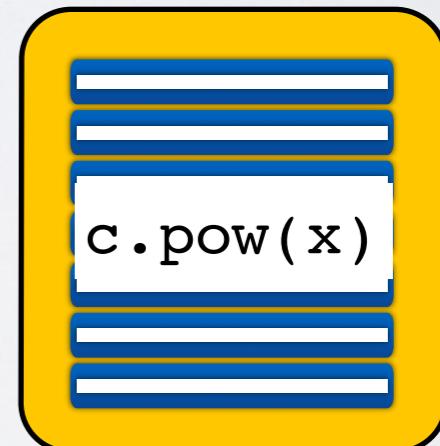
Top-level Method Coverage

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
throws NullPointerException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



⋮



⋮

Top-level Method Coverage No exception

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
    1   if (isNaN) return NaN;
    2   double r = log(abs());
    3   double i;
    4   i = atan2(imaginary, real);
    5   return createComplex(r, i);
}

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



⋮



⋮

Line Coverage

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



⋮

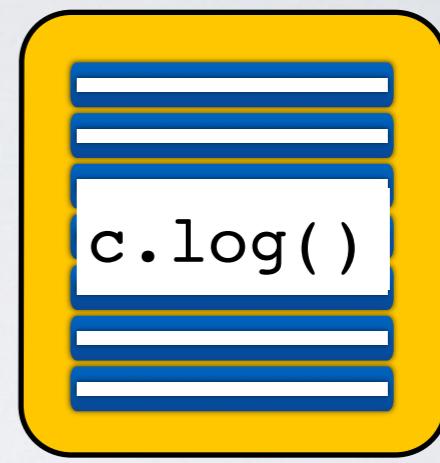


⋮

Branch Coverage

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



⋮



Direct Branch Coverage

FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real); { 0
5  return createComplex(r, i); +}
}

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



FITNESS FUNCTIONS (I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullPointerException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



⋮

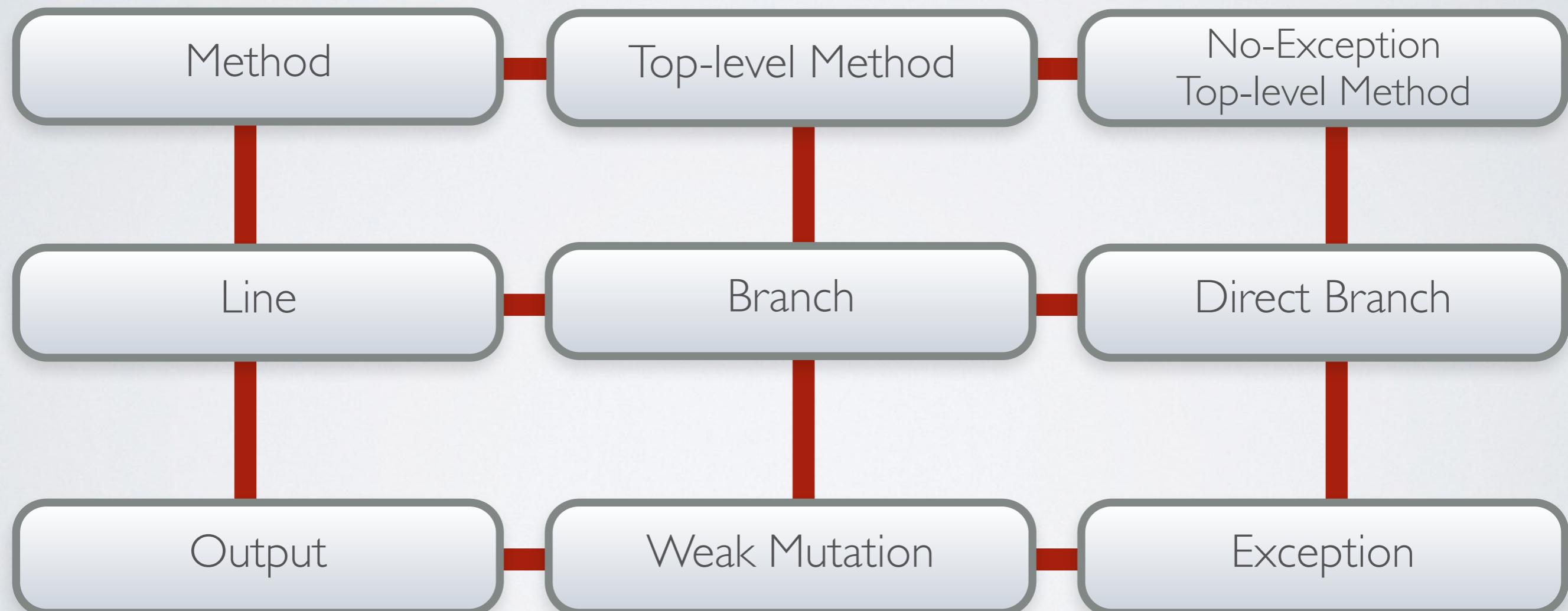


⋮

Number of exceptions

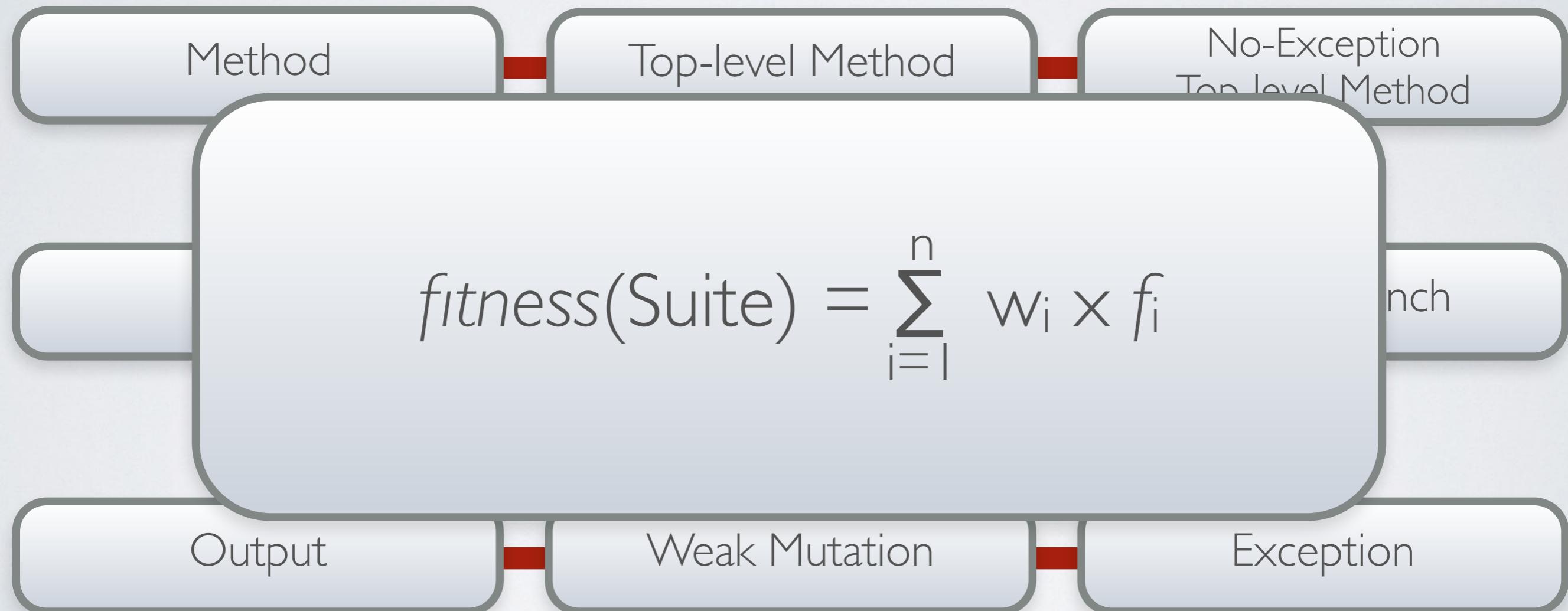
FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA



FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA



RESEARCH QUESTIONS

RESEARCH QUESTIONS

- **RQ1.** What are the effects of adding a second coverage criterion on test suite size and coverage?

RESEARCH QUESTIONS

- **RQ1.** What are the effects of adding a second coverage criterion on test suite size and coverage?
- **RQ2.** How does combining multiple coverage criteria influence the test suite size?

RESEARCH QUESTIONS

- **RQ1.** What are the effects of adding a second coverage criterion on test suite size and coverage?
- **RQ2.** How does combining multiple coverage criteria influence the test suite size?
- **RQ3.** How does combining multiple coverage criteria affect the performance of the constituent criteria?

RESEARCH QUESTIONS

- **RQ1.** What are the effects of adding a second coverage criterion on test suite size and coverage?
- **RQ2.** How does combining multiple coverage criteria influence the test suite size?
- **RQ3.** How does combining multiple coverage criteria affect the performance of the constituent criteria?
- **RQ4.** How does coverage vary with increasing search budget?

EXPERIMENTAL SETUP

EXPERIMENTAL SETUP



EXPERIMENTAL SETUP



SFI 110 Corpus

EXPERIMENTAL SETUP



SFI110 Corpus

| 110 Projects
+20,000 Classes

EXPERIMENTAL SETUP

EVSUITE

SFIIO Corpus

| 110 Projects
+20,000 Classes



650

EXPERIMENTAL SETUP

EVSUITE

SFIIO Corpus

| 110 Projects
+20,000 Classes



650



10

EXPERIMENTAL SETUP



SFI110 Corpus

| 110 Projects
+20,000 Classes



650



10



2/10

EXPERIMENTAL SETUP



SFI 110 Corpus

| 110 Projects
+20,000 Classes



650



10



2/10



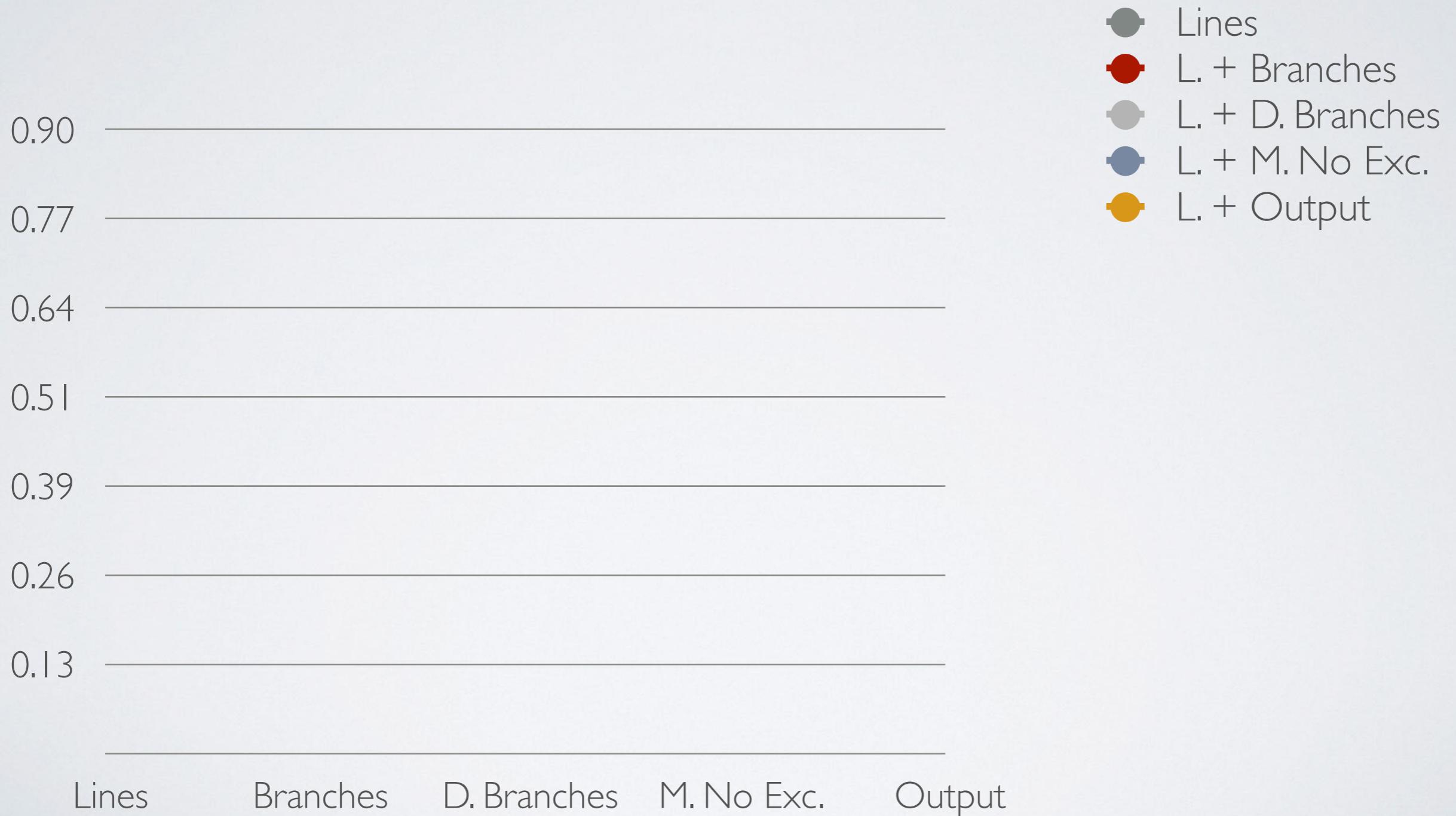
40/5

TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion

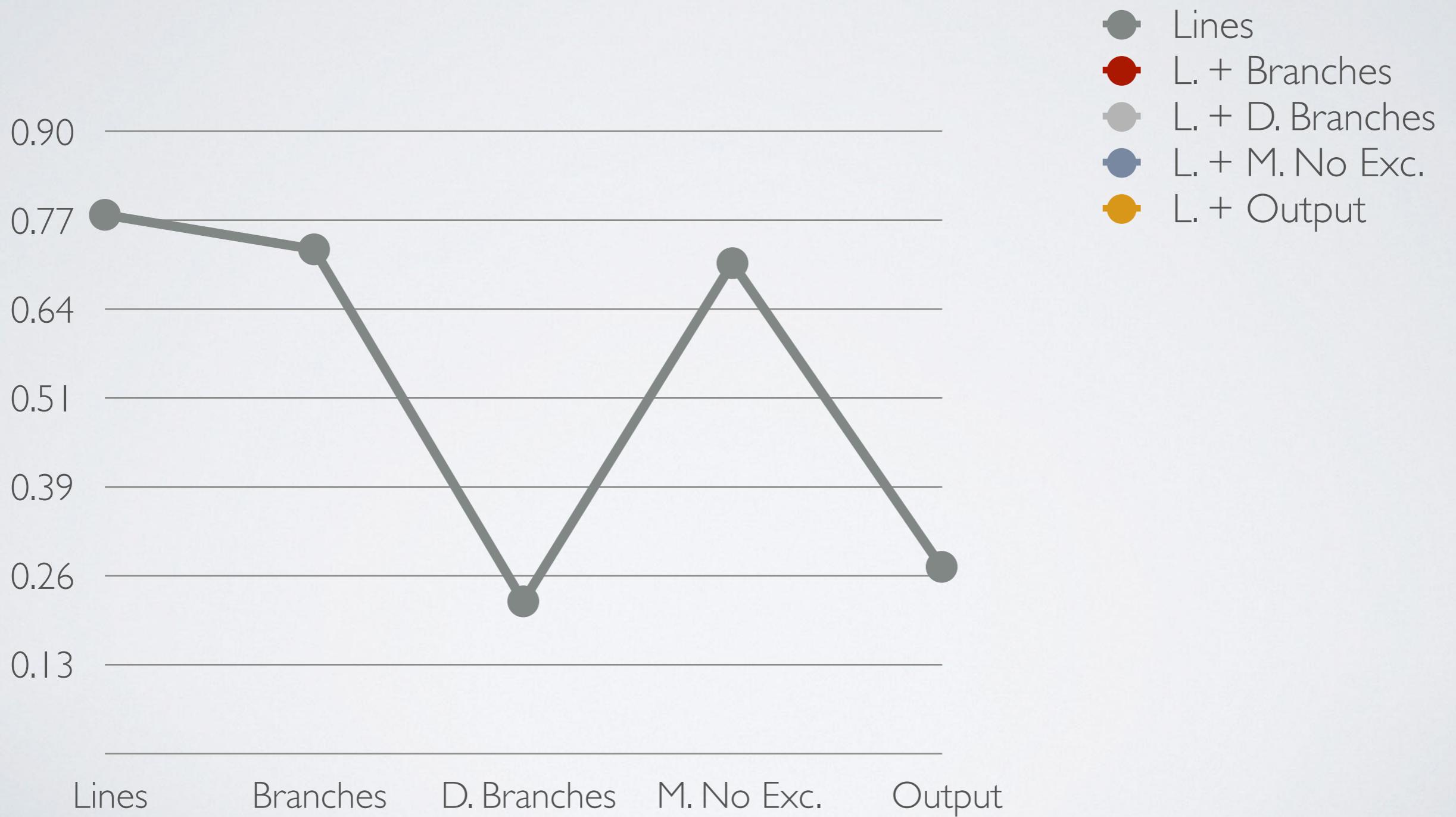
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



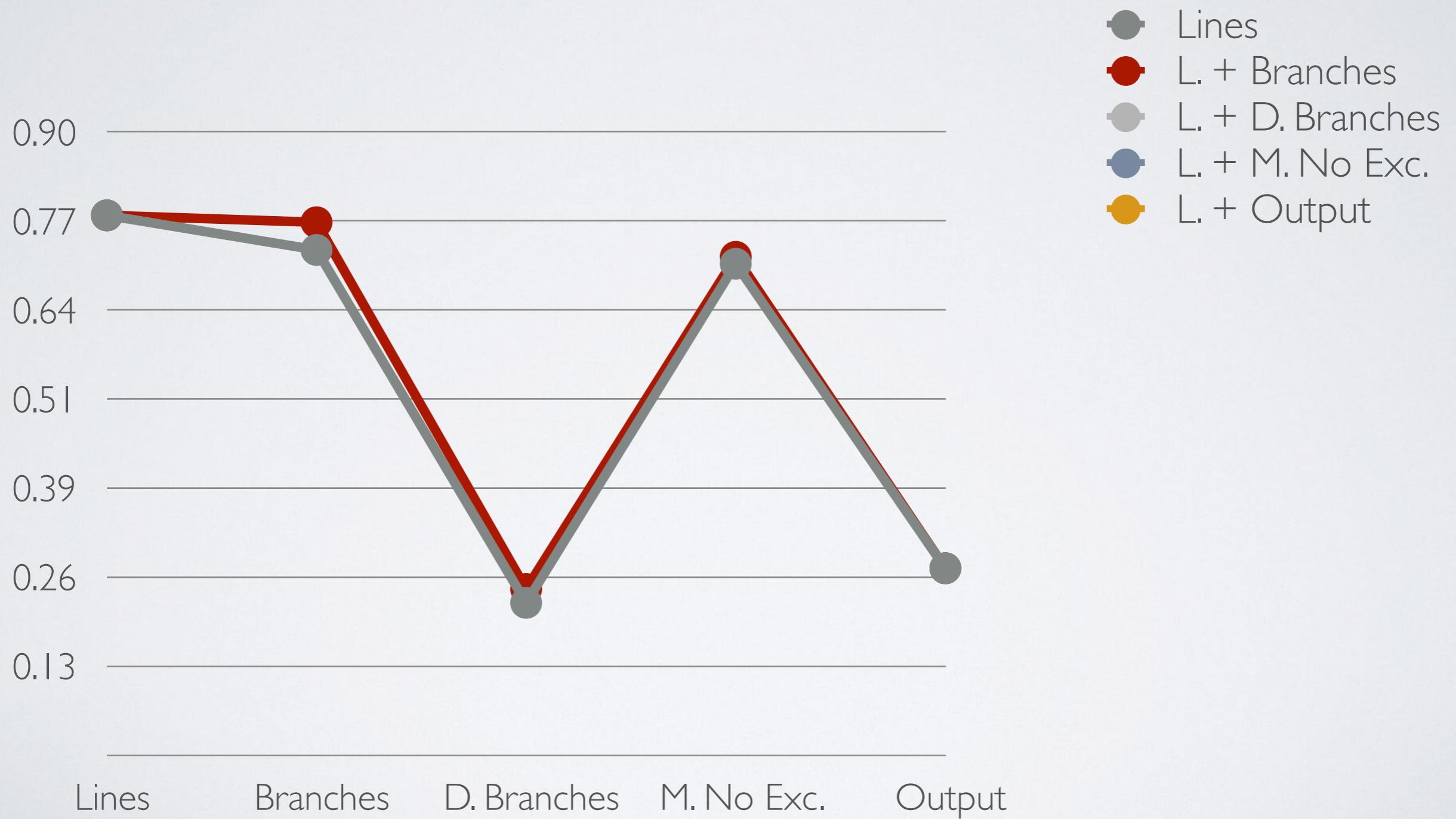
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



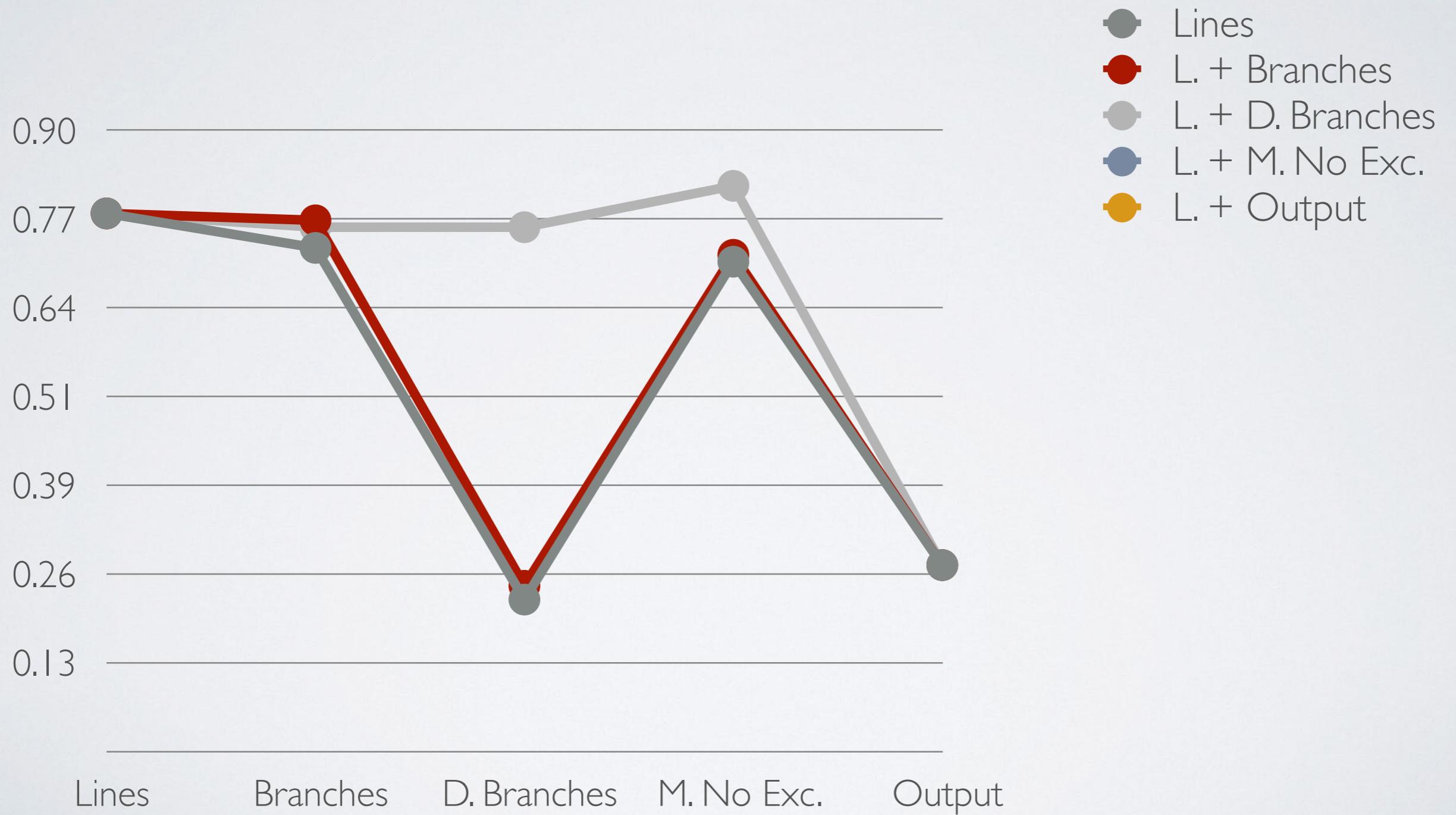
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



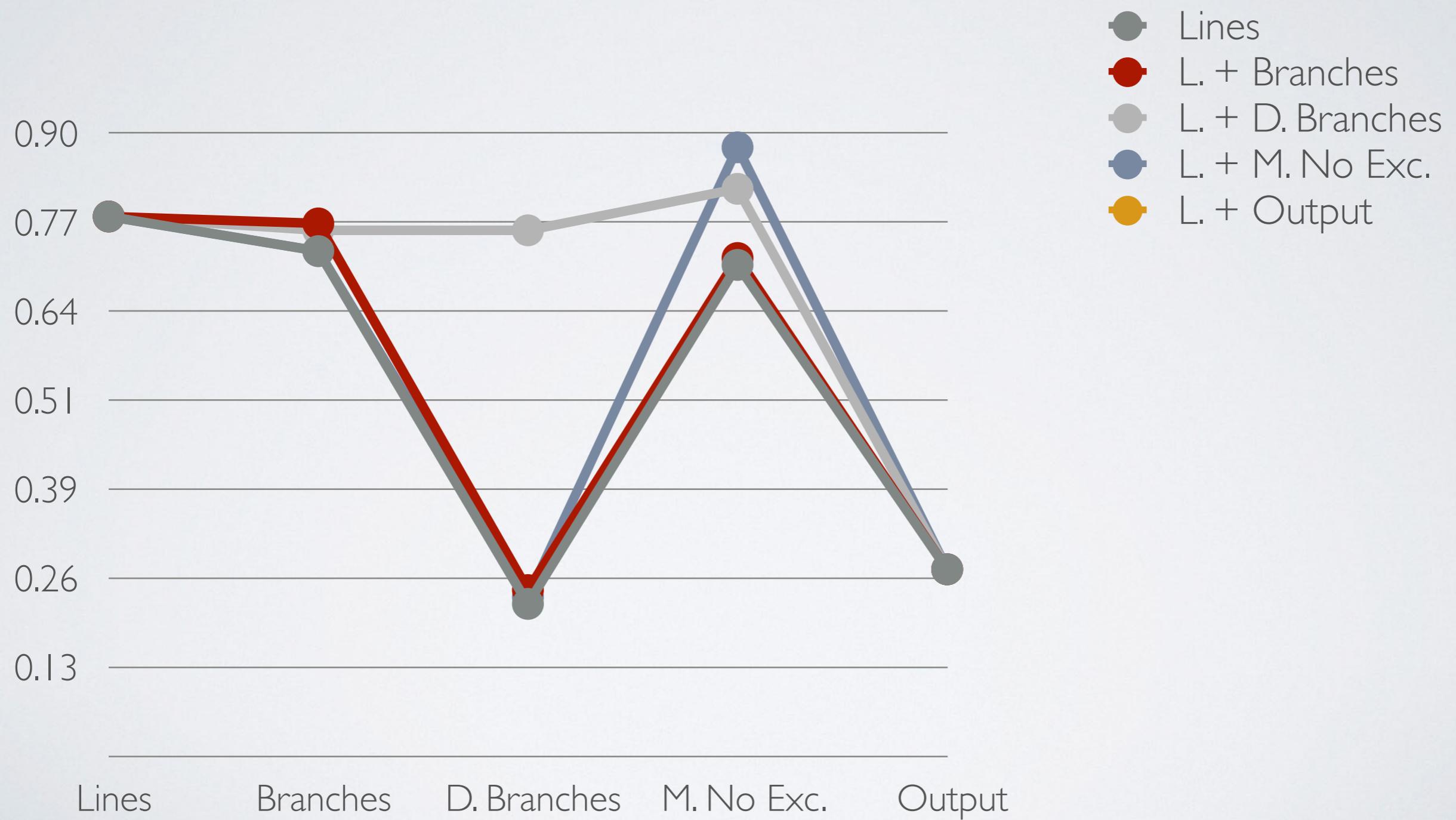
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



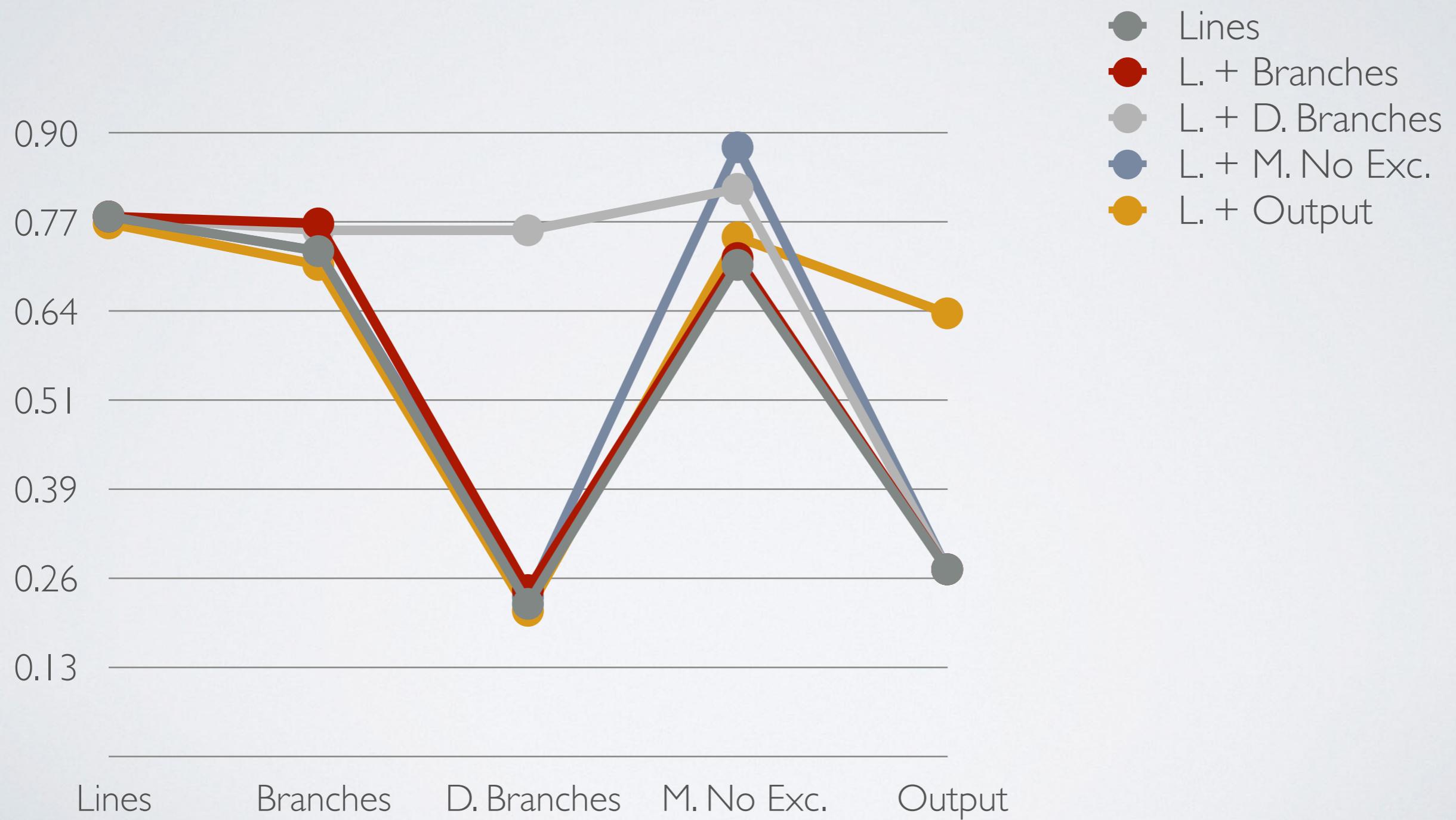
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



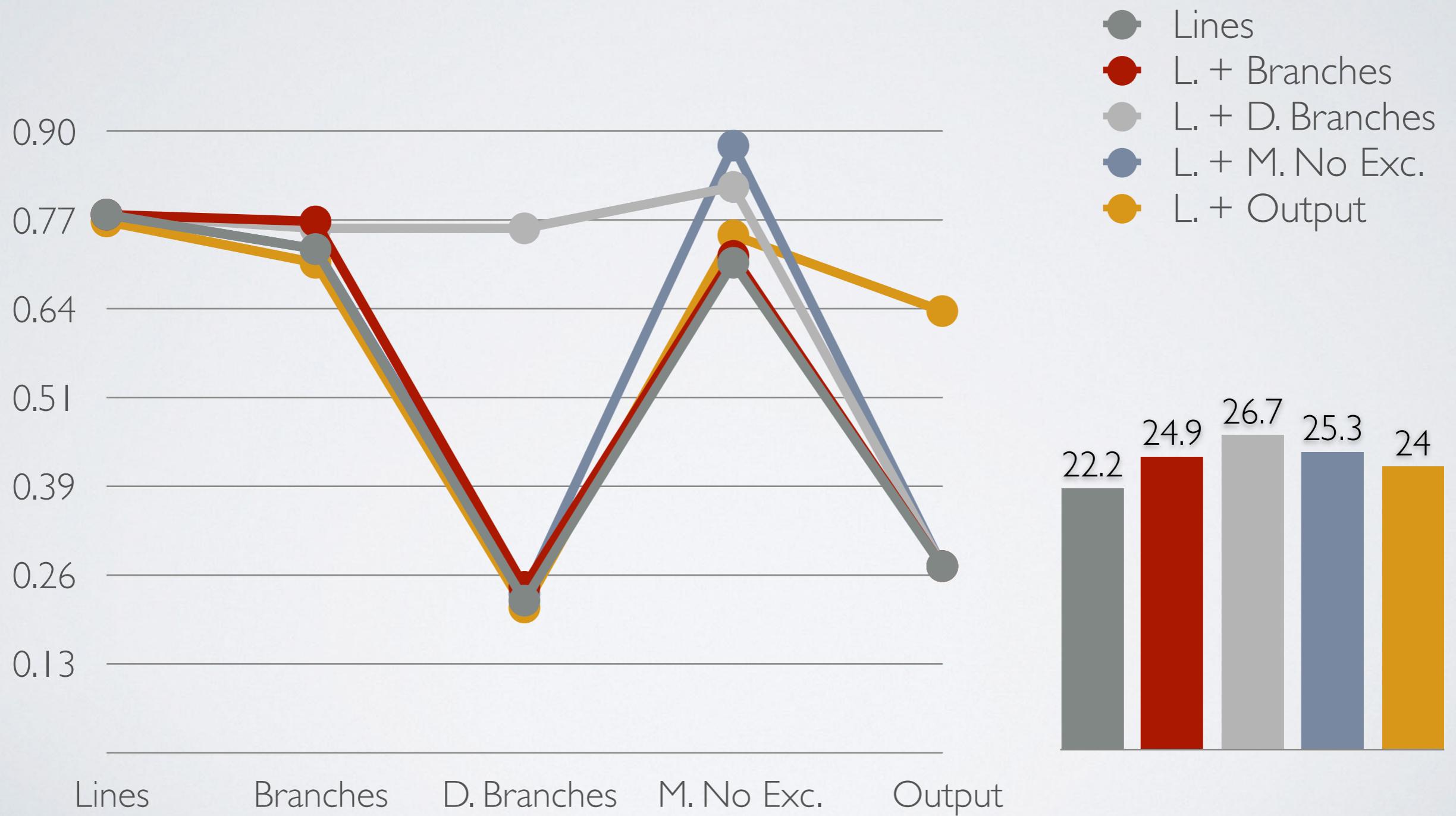
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



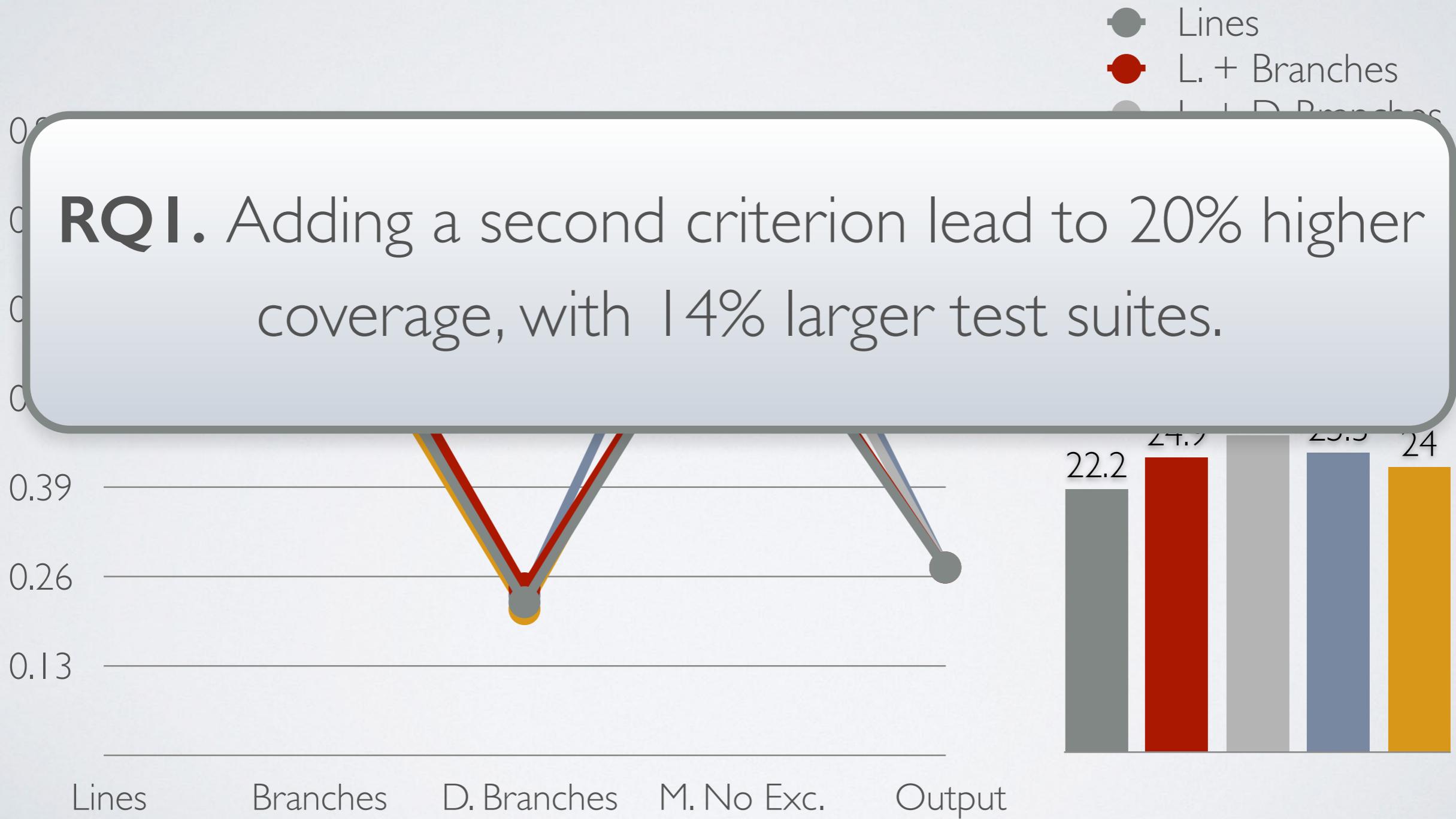
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



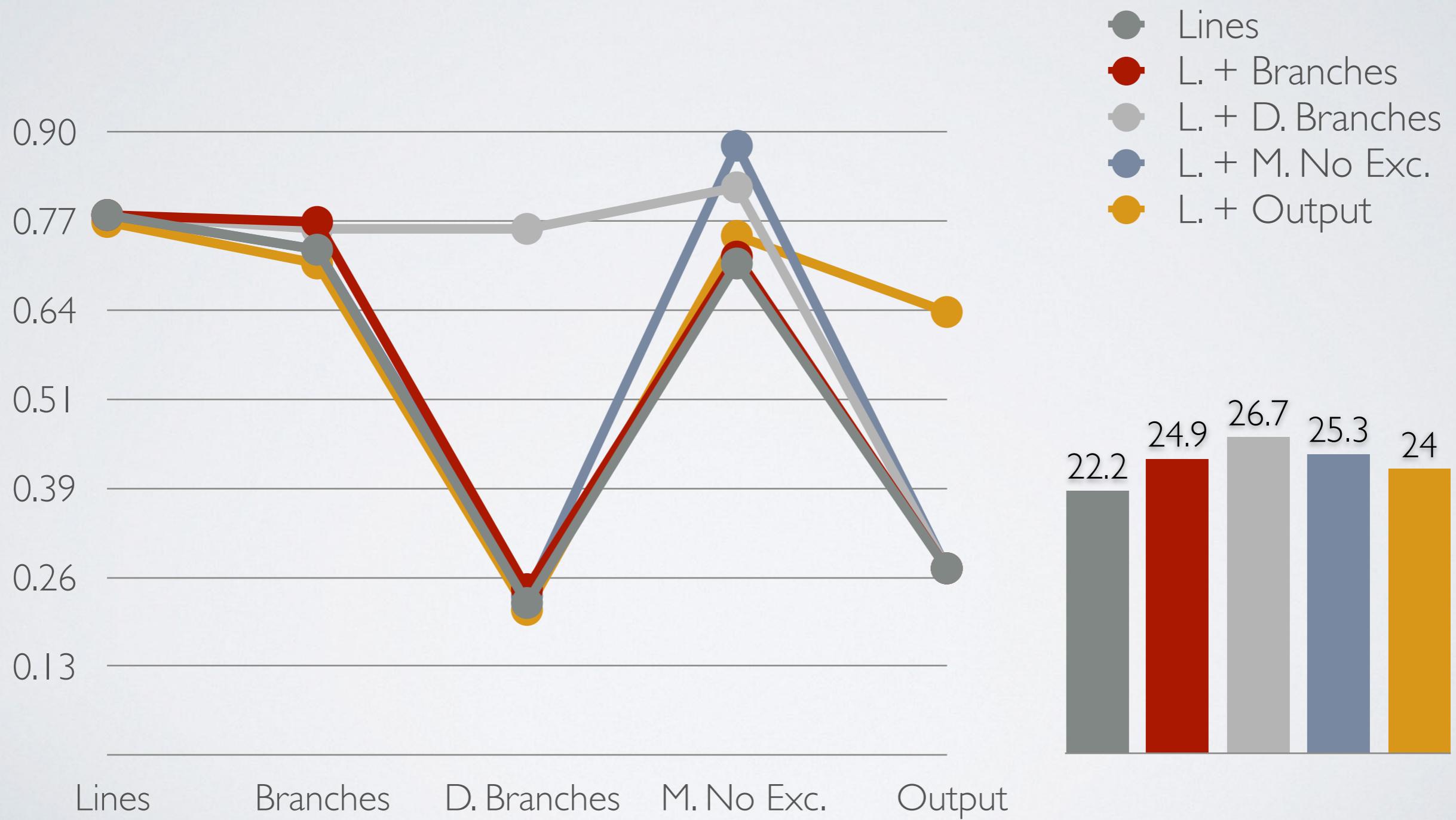
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



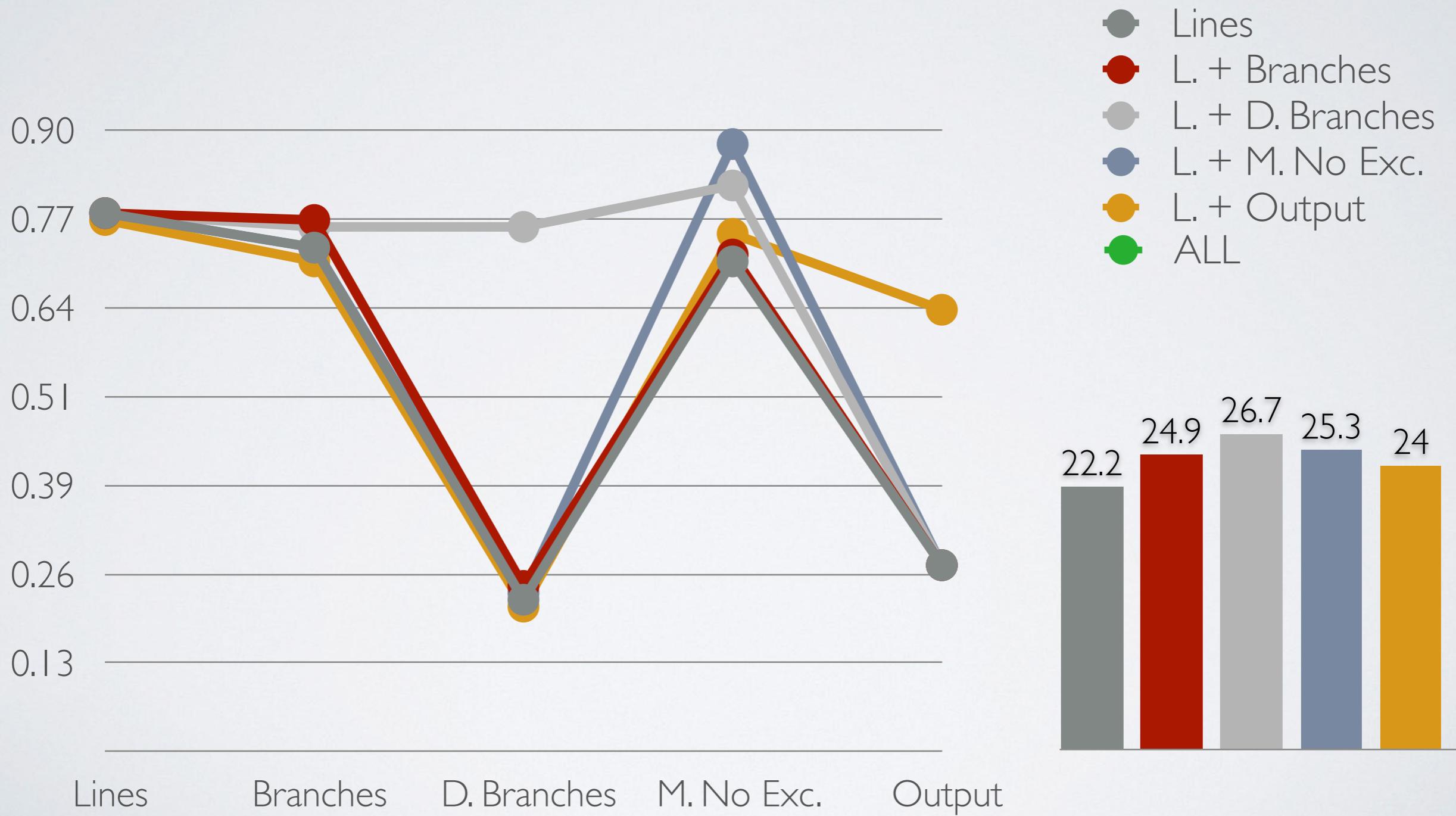
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



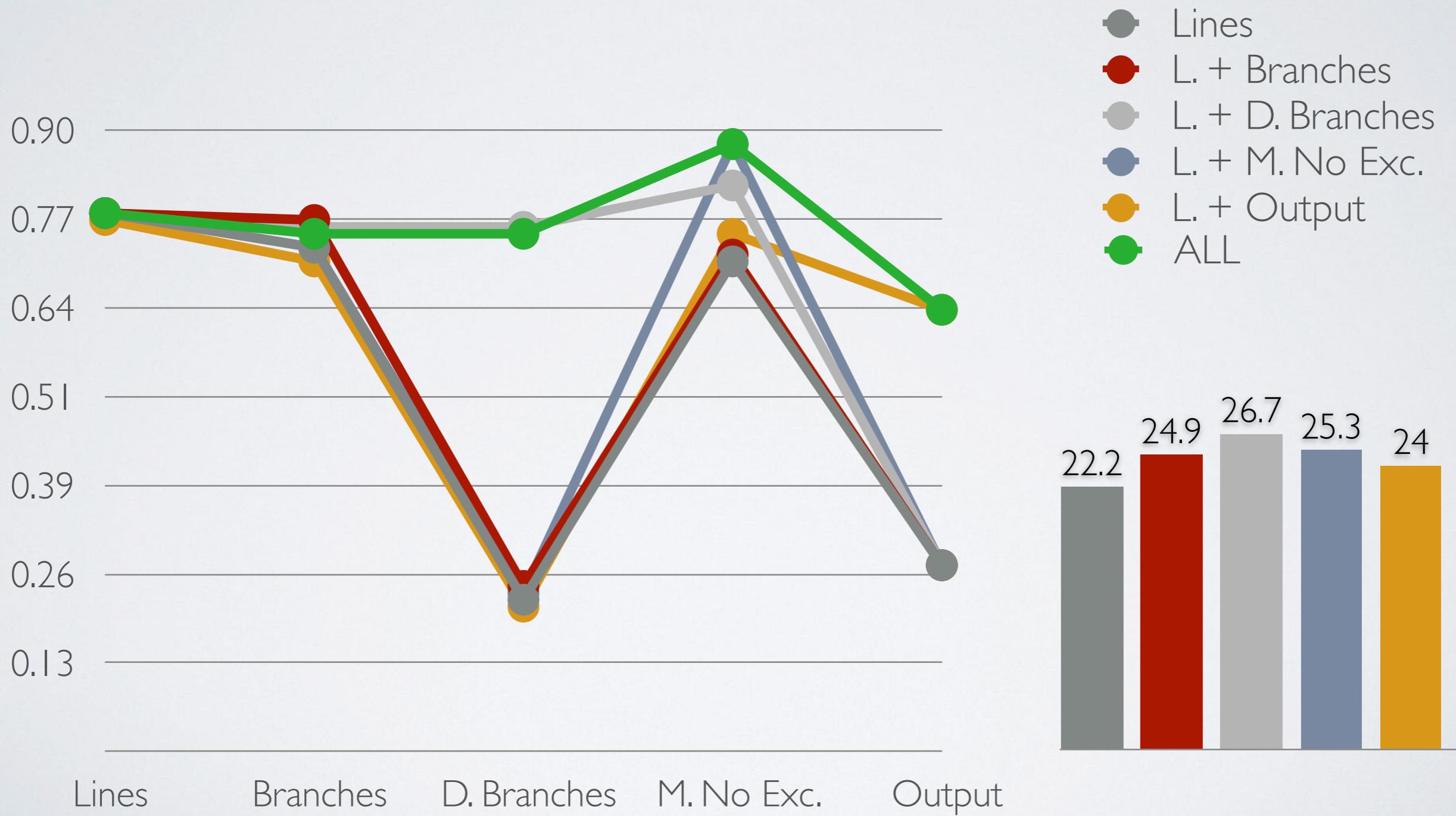
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



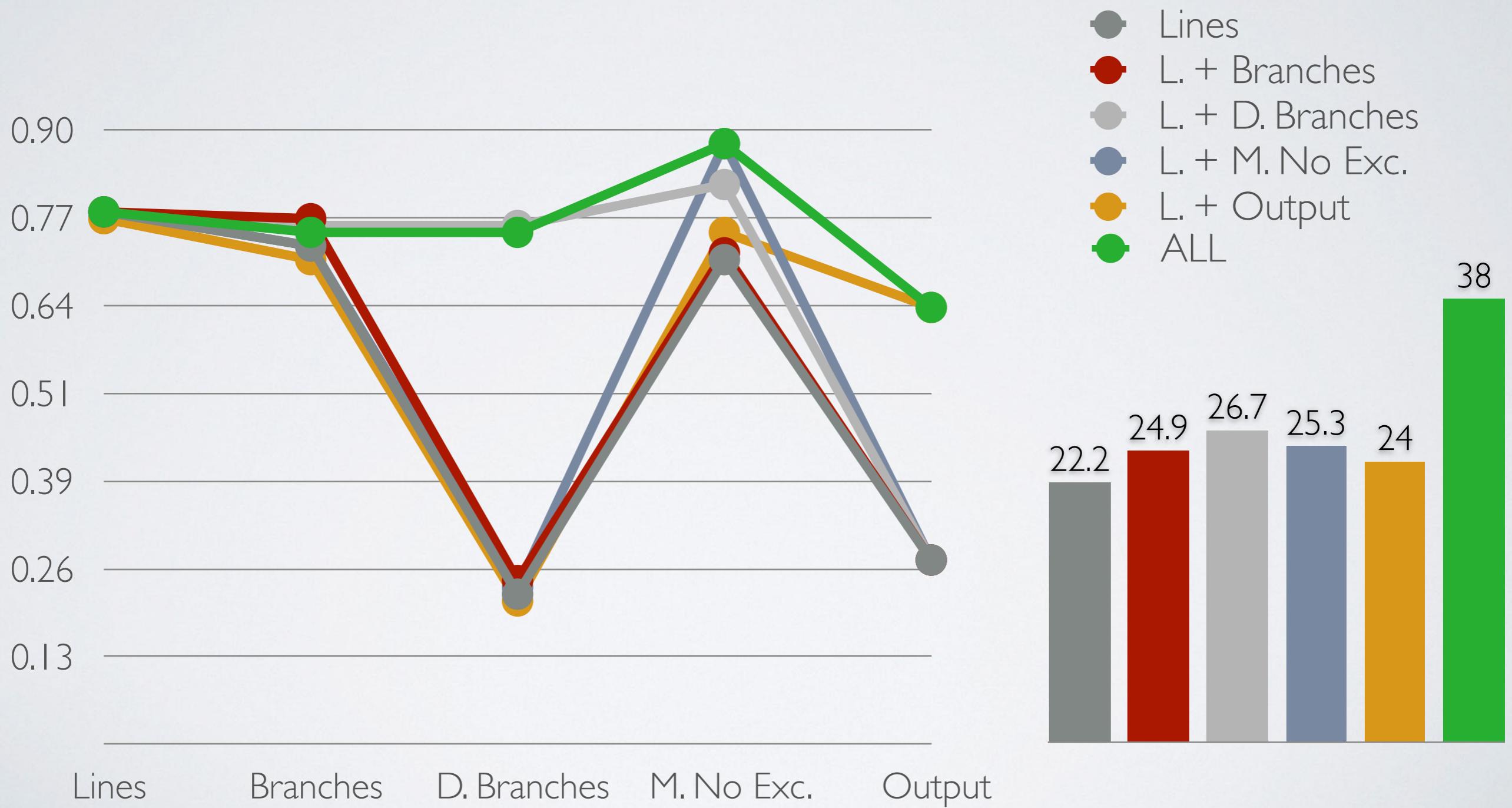
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



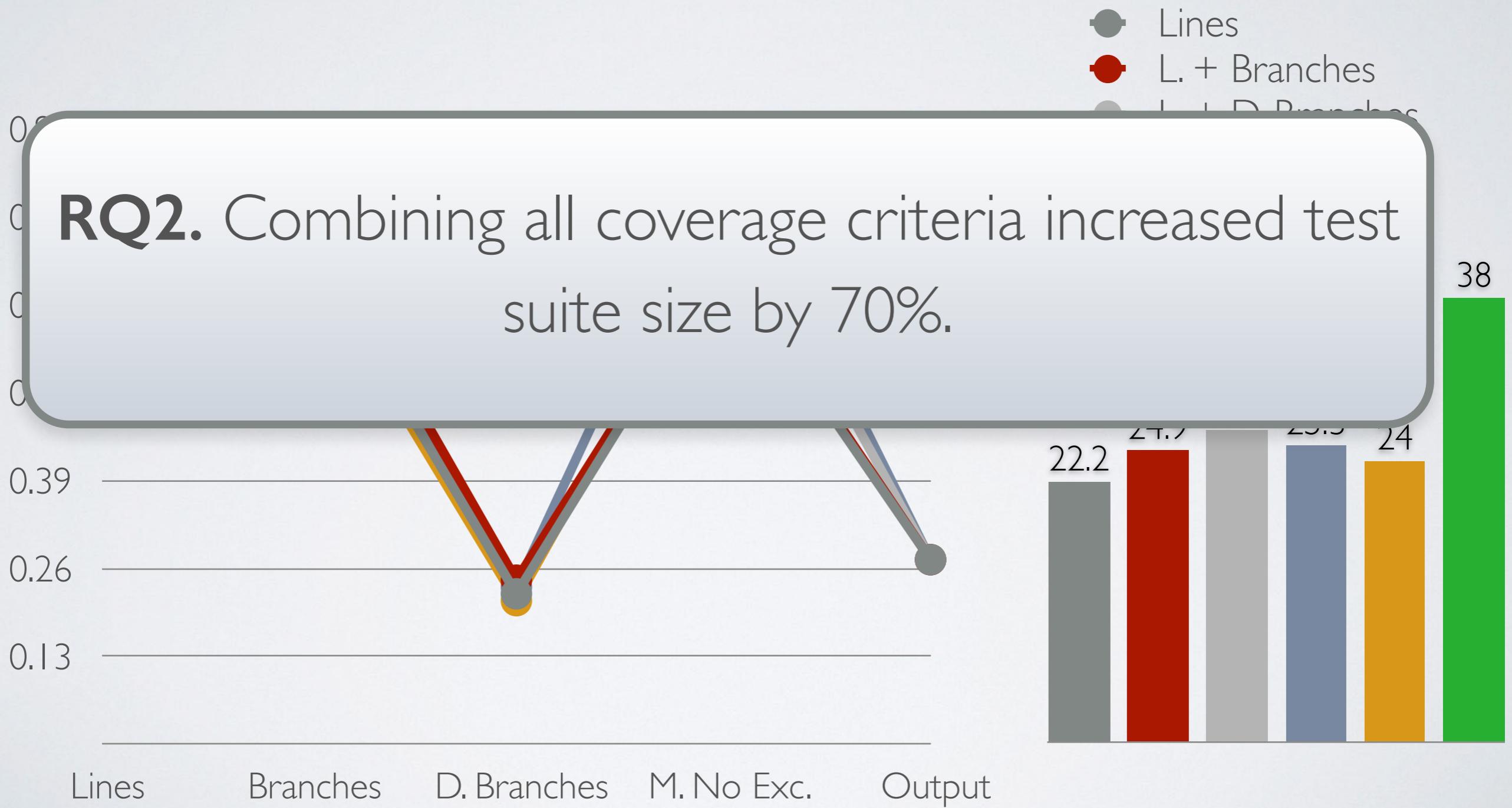
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria

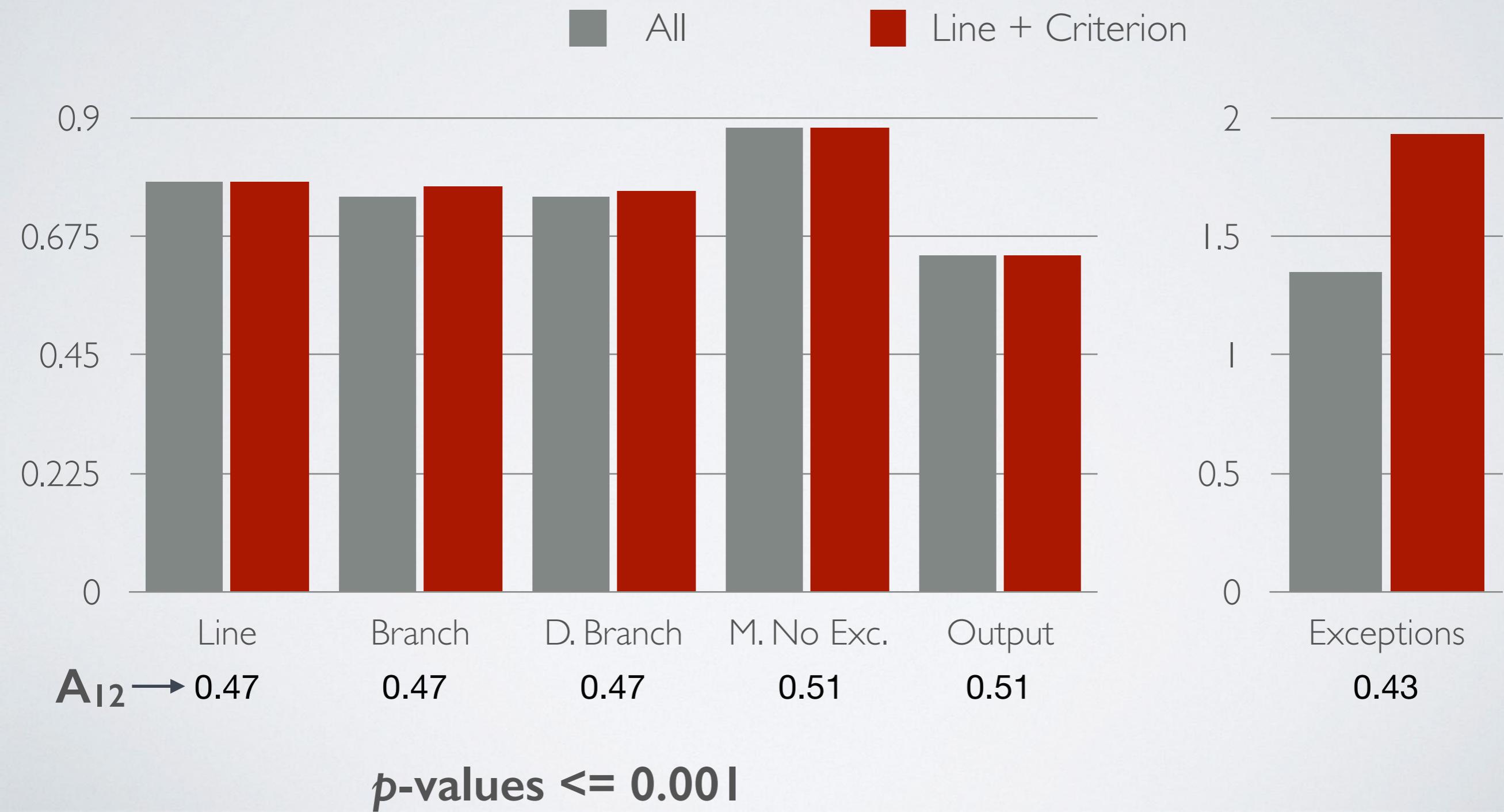


TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



COVERAGE OF CONSTITUENT CRITERIA



COVERAGE OF CONSTITUENT CRITERIA

■ All ■ Line + Criterion

RQ3. Combining multiple criteria lead to a 0.4% coverage decrease in the constituent criteria.

$A_{12} \rightarrow 0.47$

0.47

0.47

0.51

0.51

0.43

p-values <= 0.001

INCREASED SEARCH BUDGET



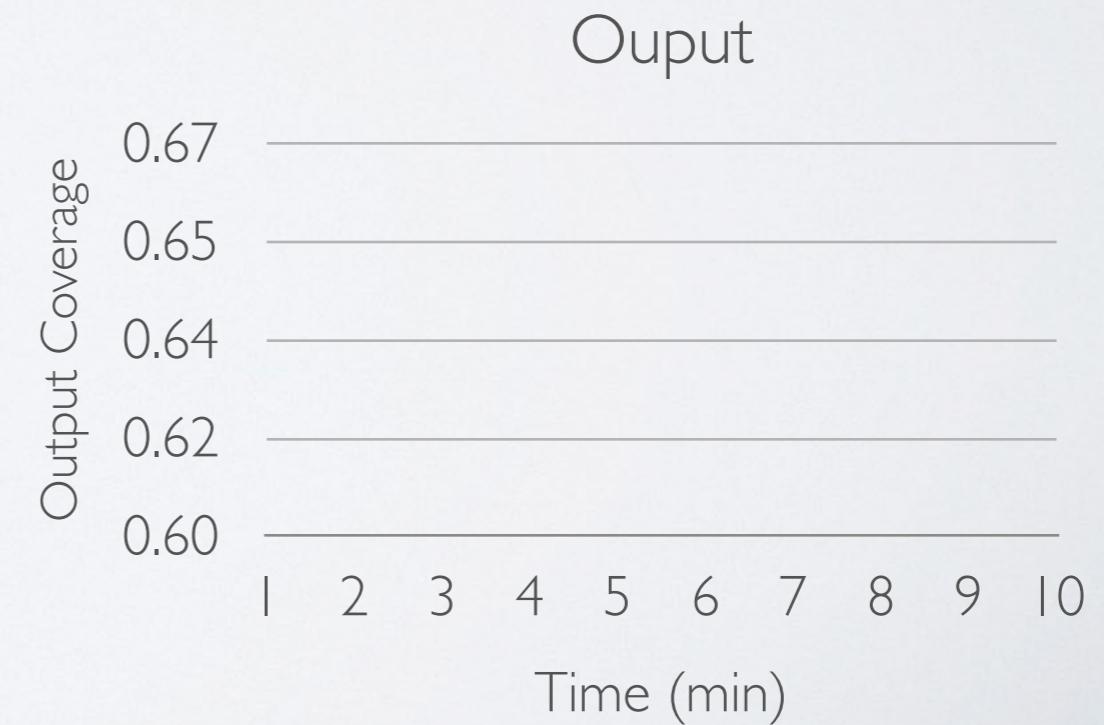
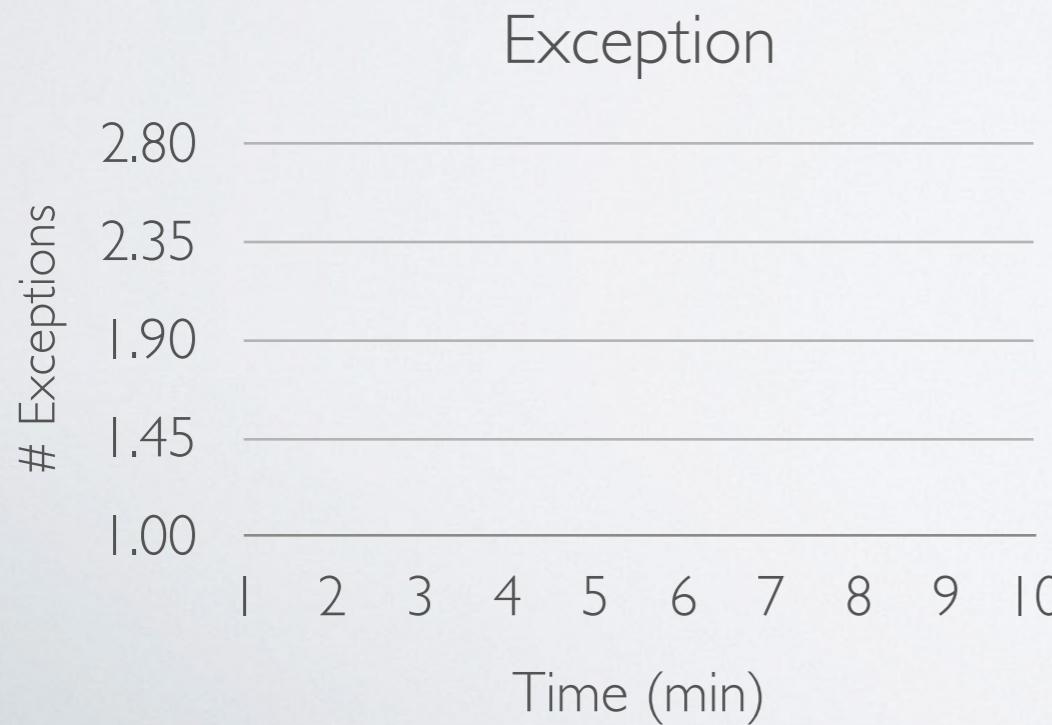
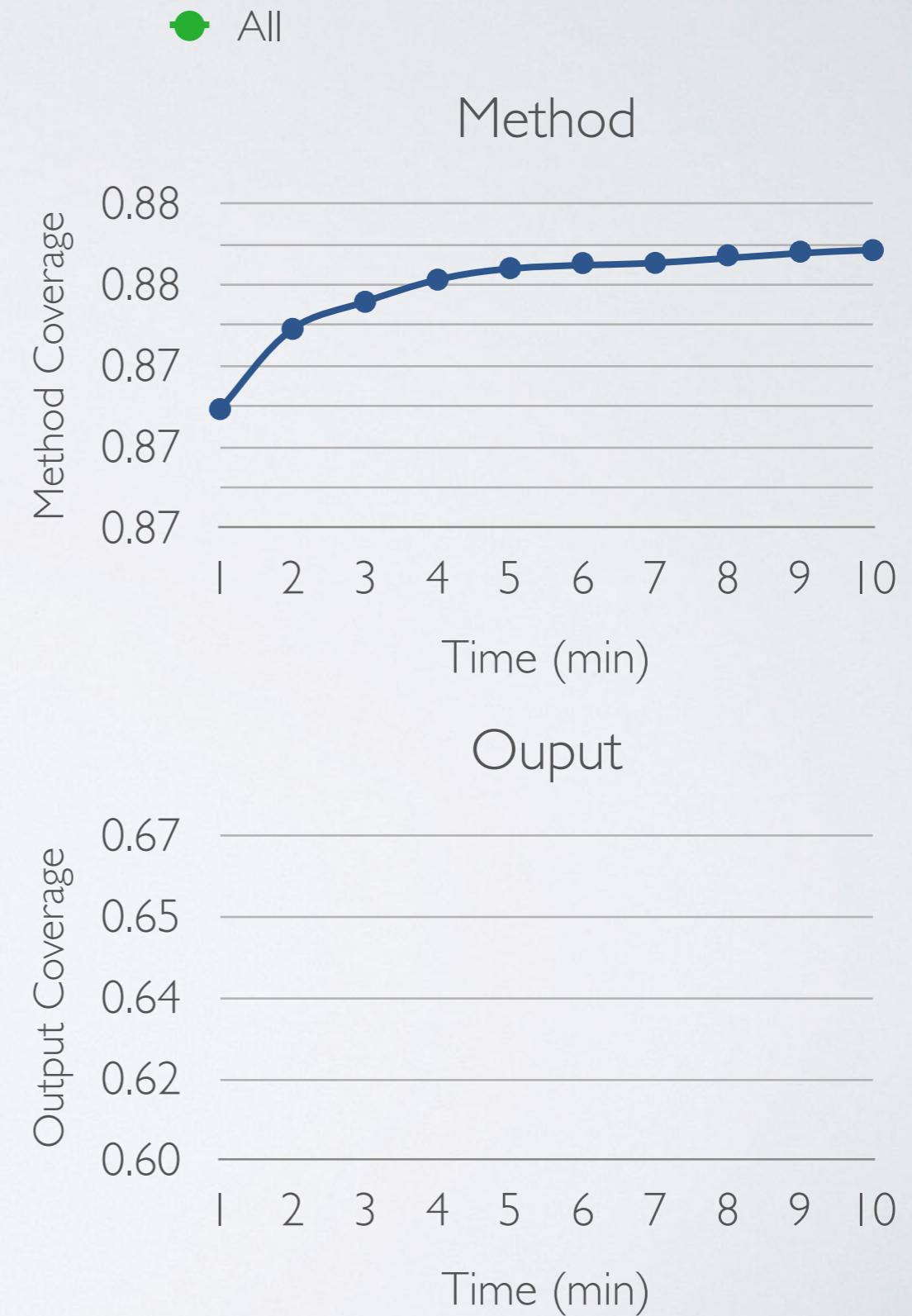
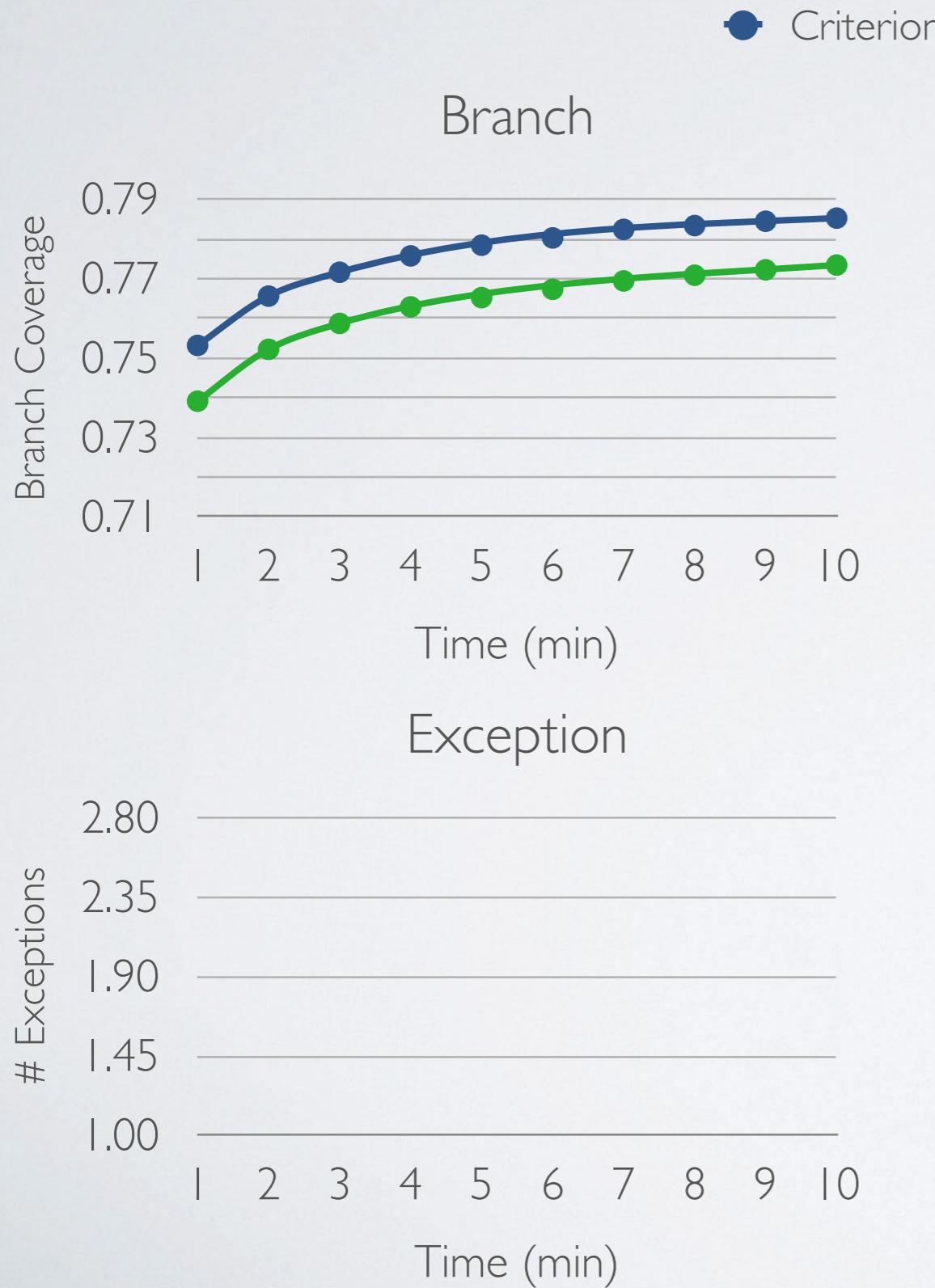
INCREASED SEARCH BUDGET



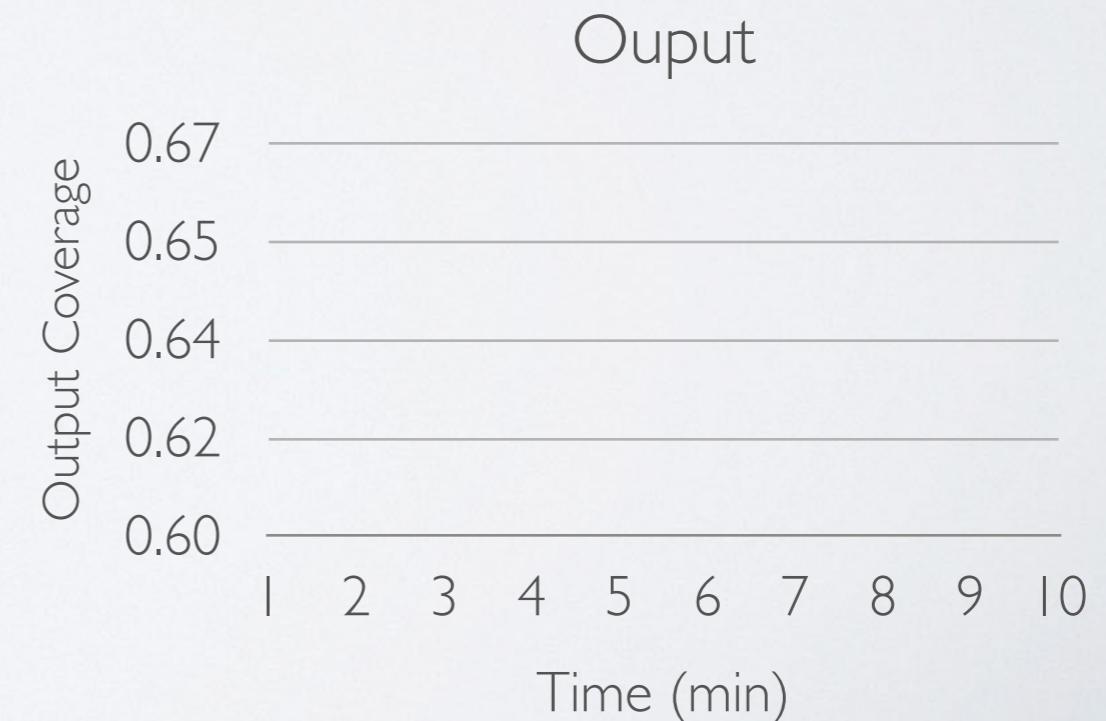
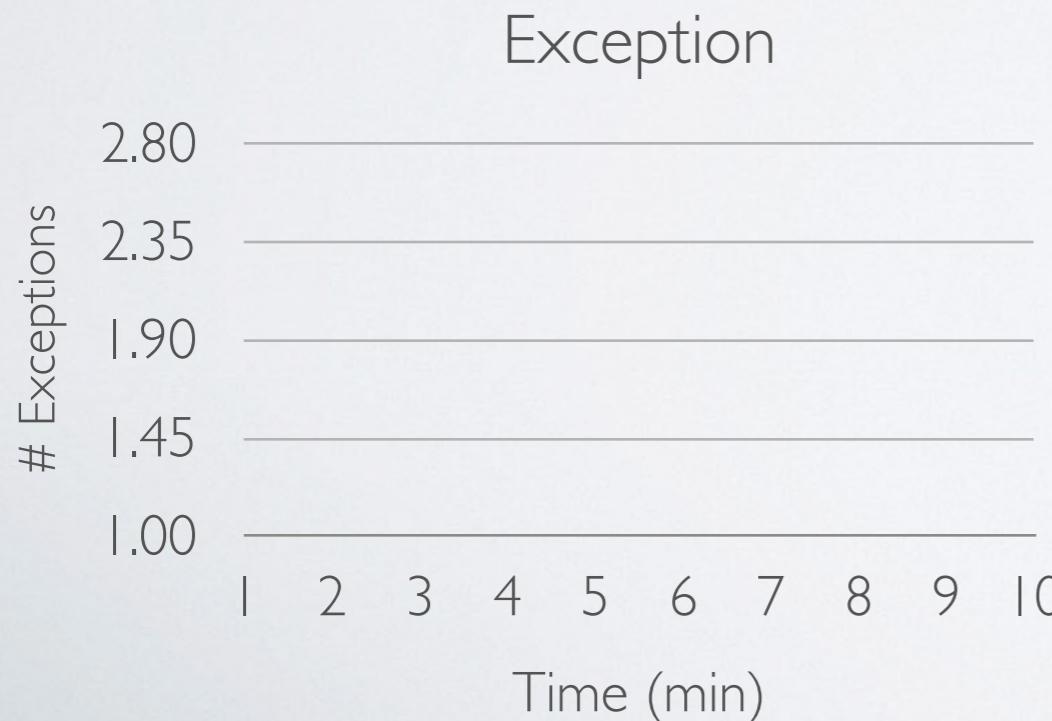
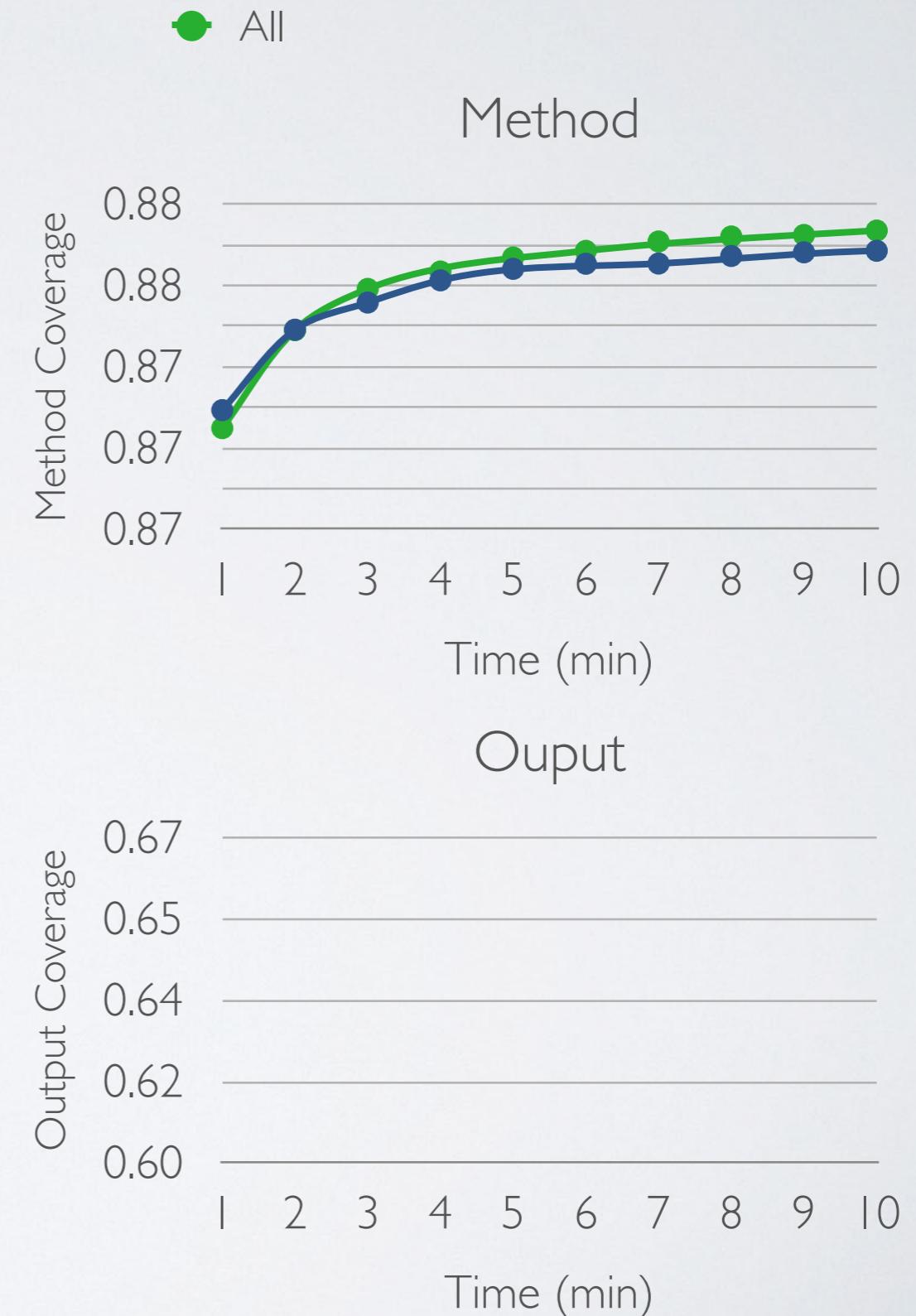
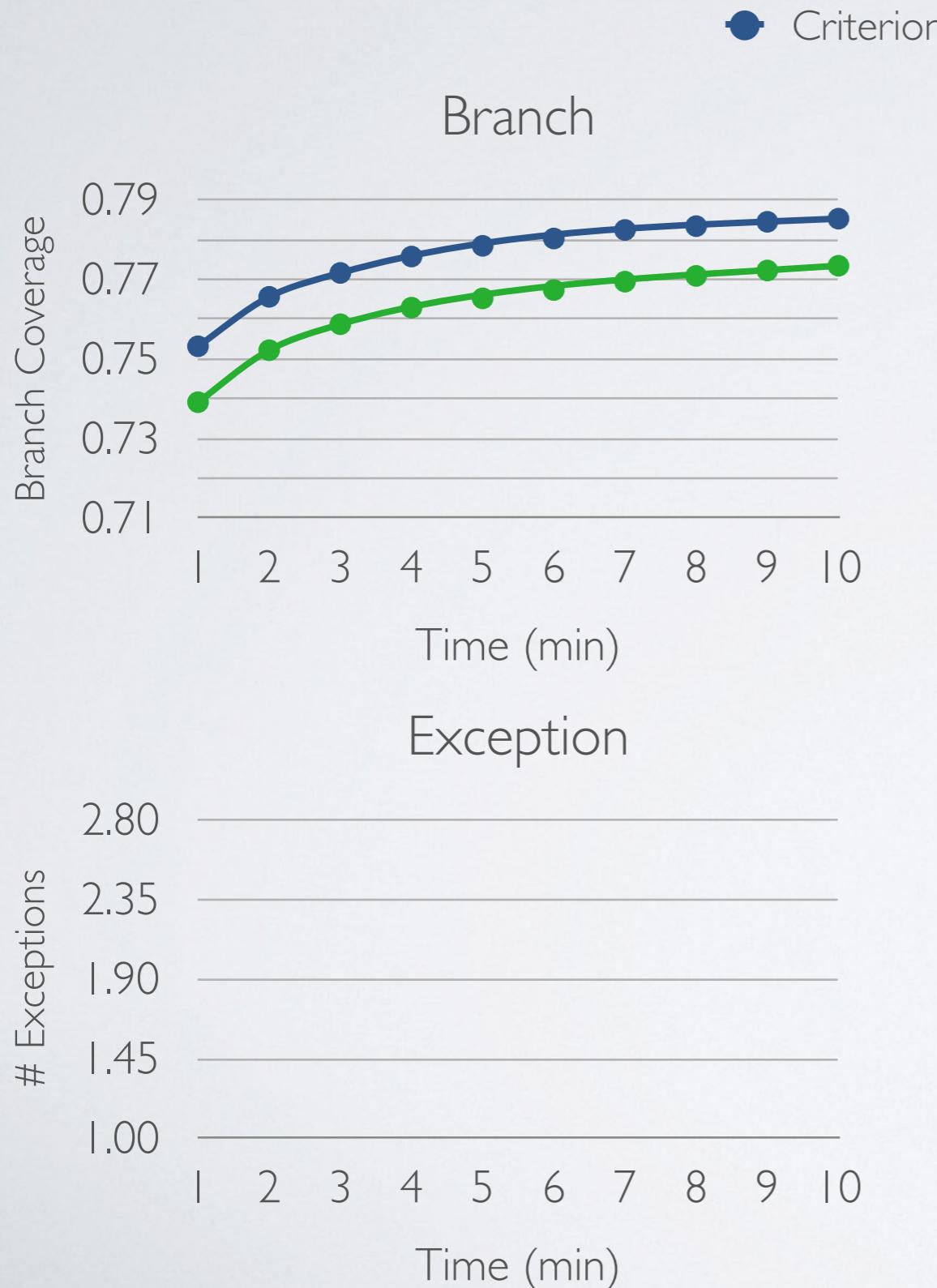
INCREASED SEARCH BUDGET



INCREASED SEARCH BUDGET



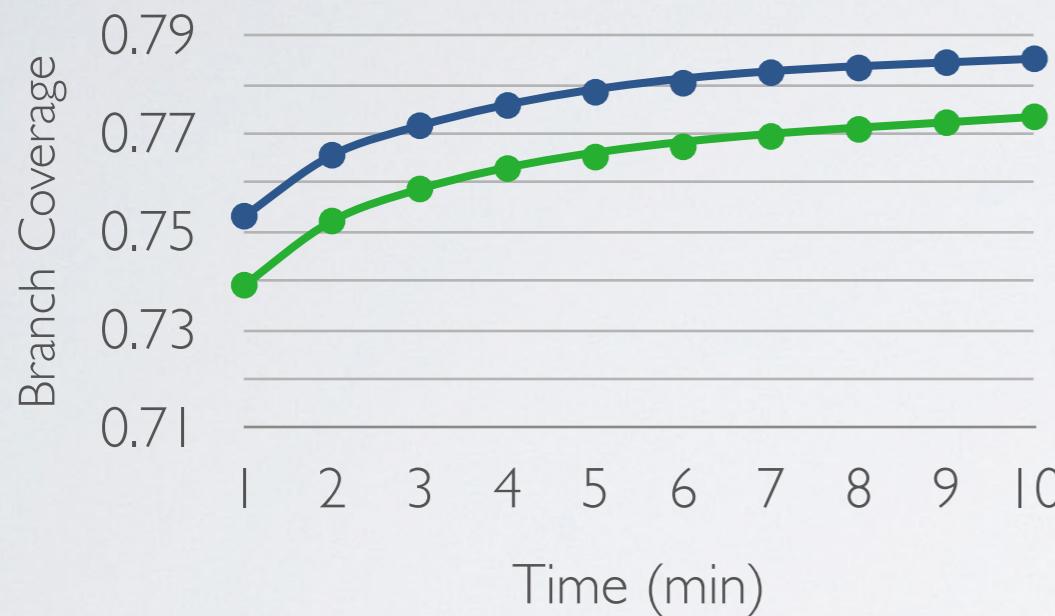
INCREASED SEARCH BUDGET



INCREASED SEARCH BUDGET

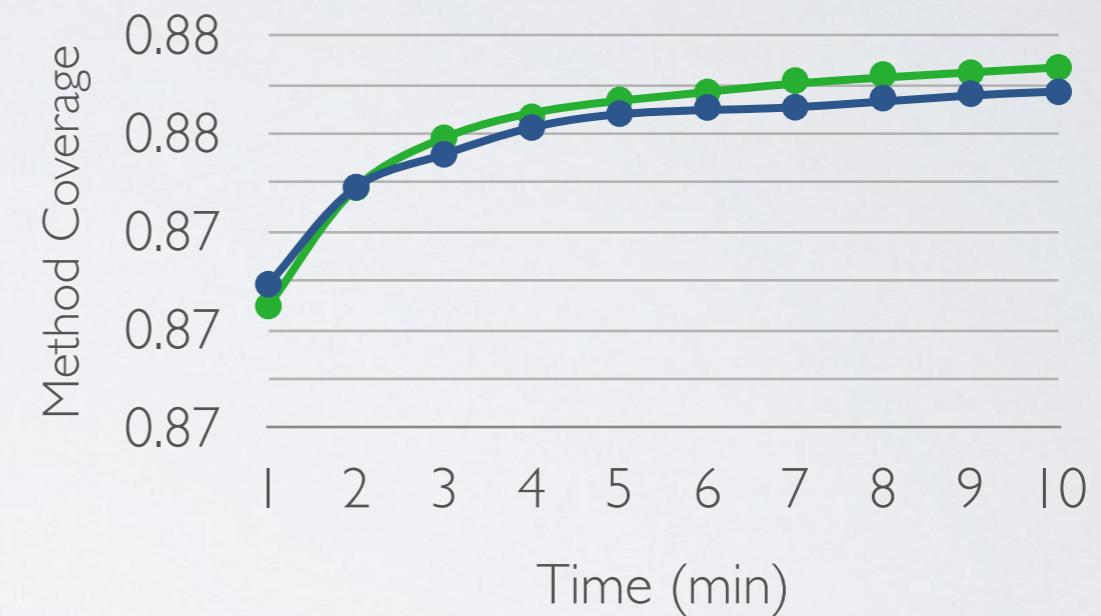
Criterion

Branch

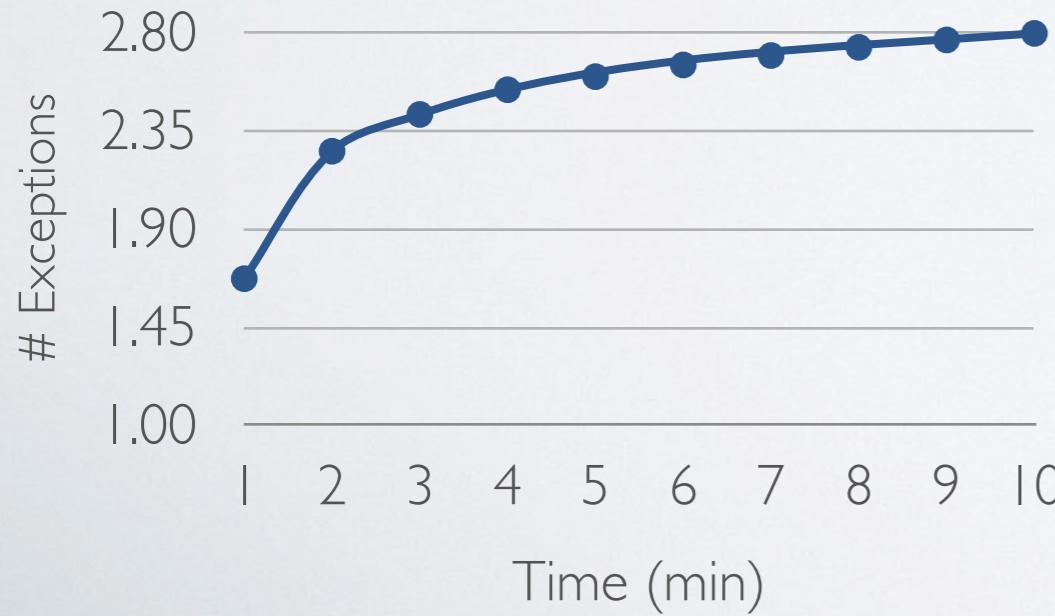


All

Method



Exception



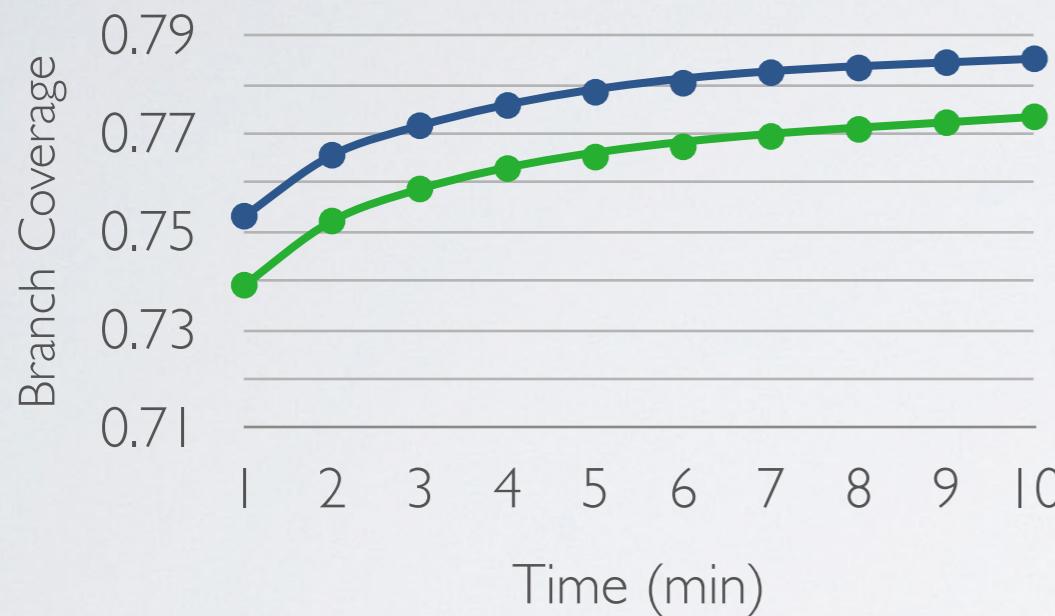
Ouput



INCREASED SEARCH BUDGET

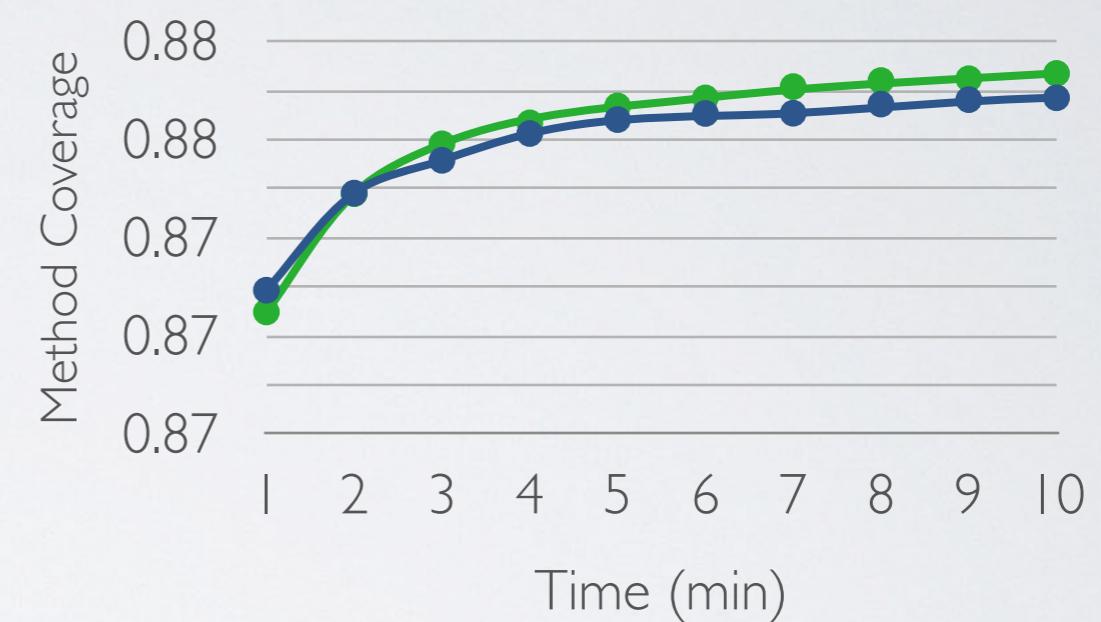
Criterion

Branch

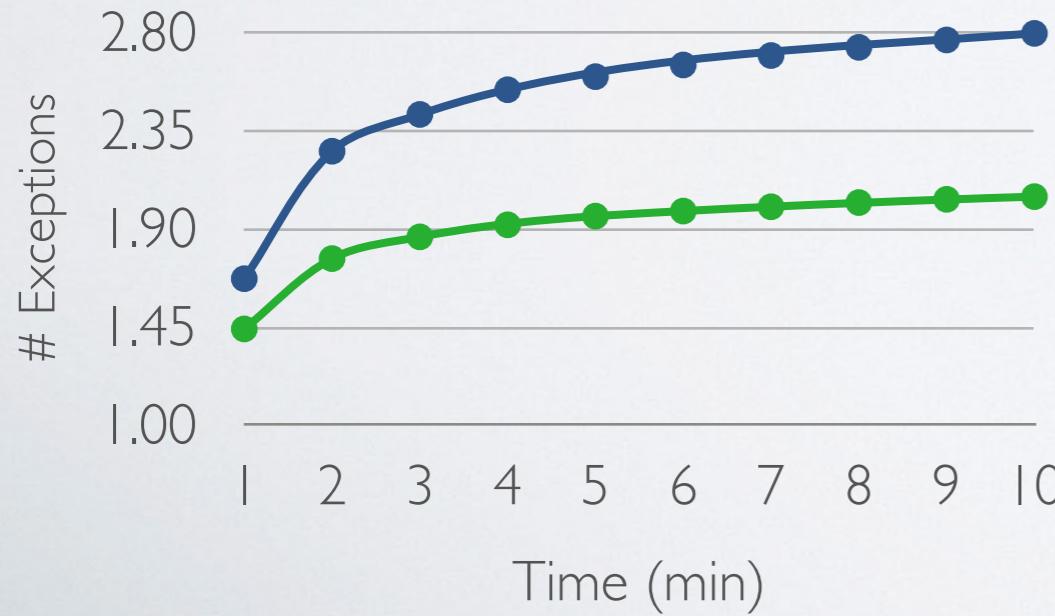


All

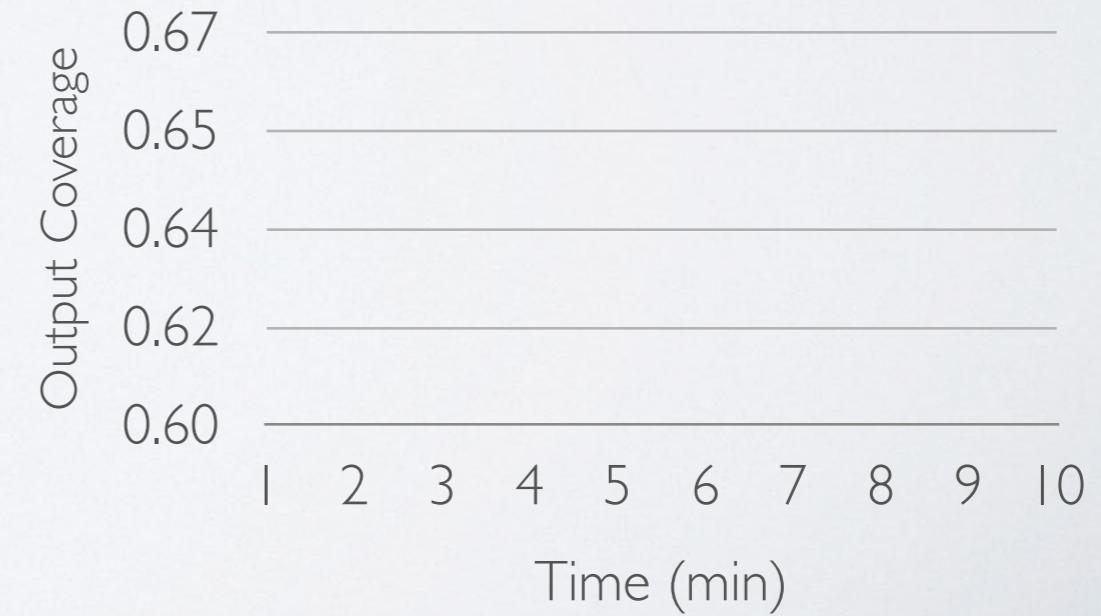
Method



Exception



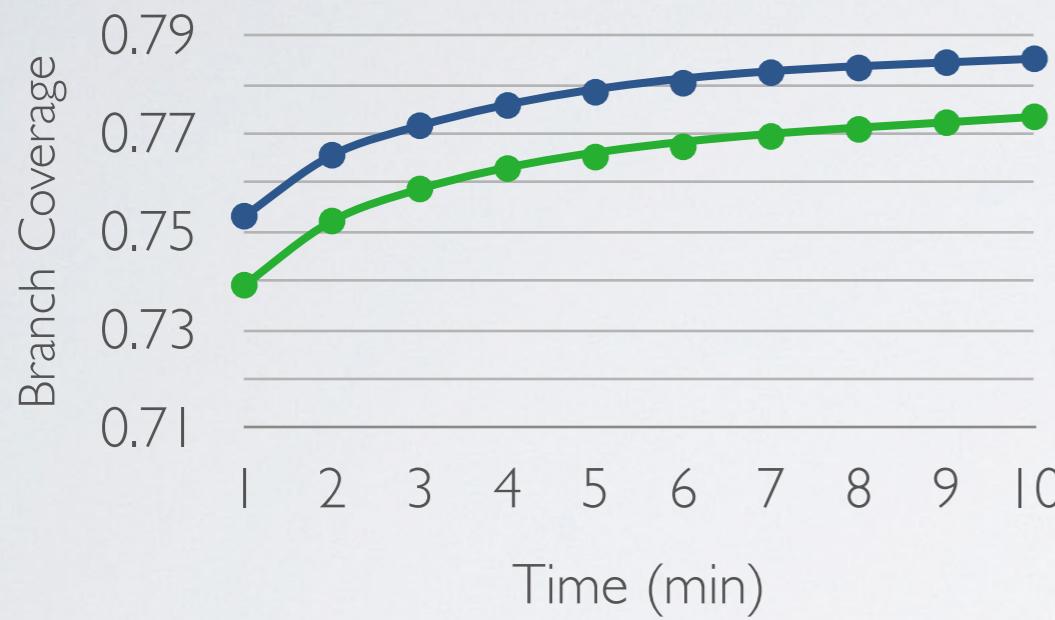
Ouput



INCREASED SEARCH BUDGET

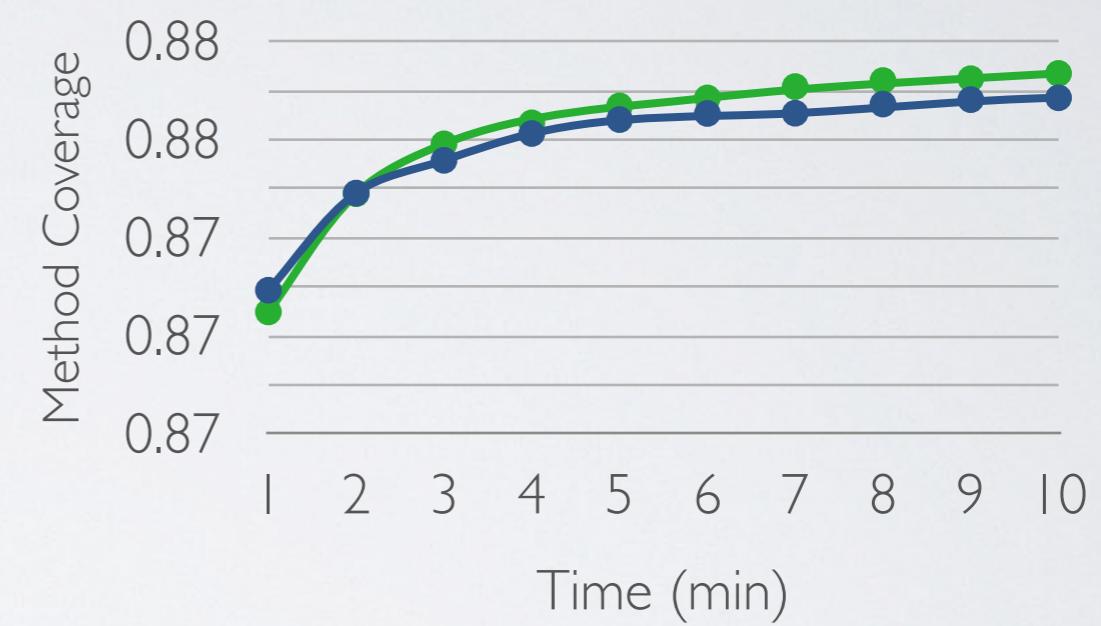
Criterion

Branch

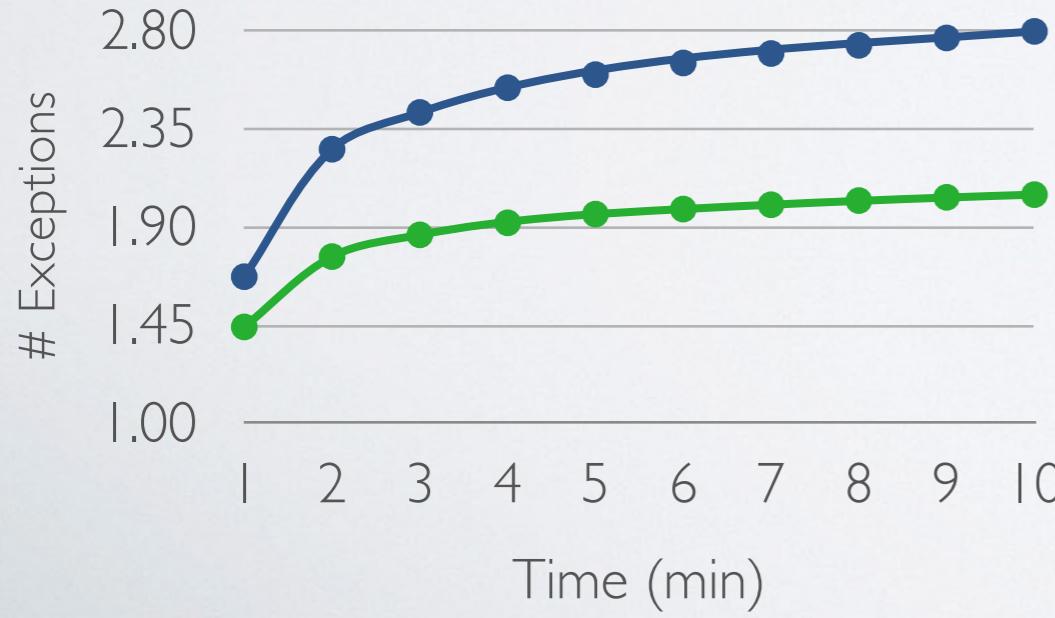


All

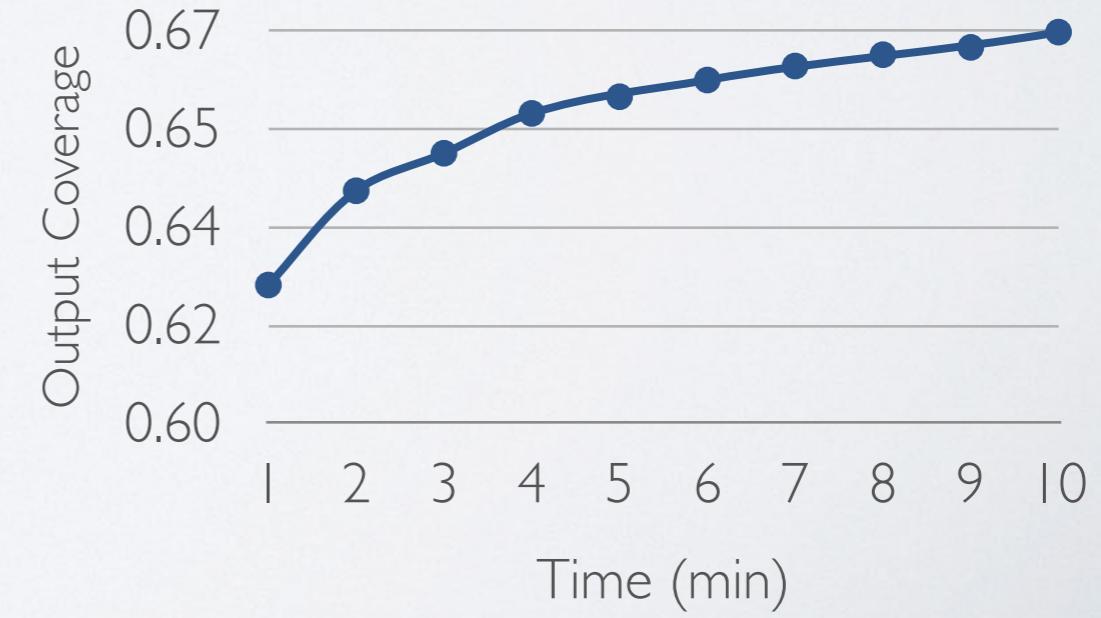
Method



Exception



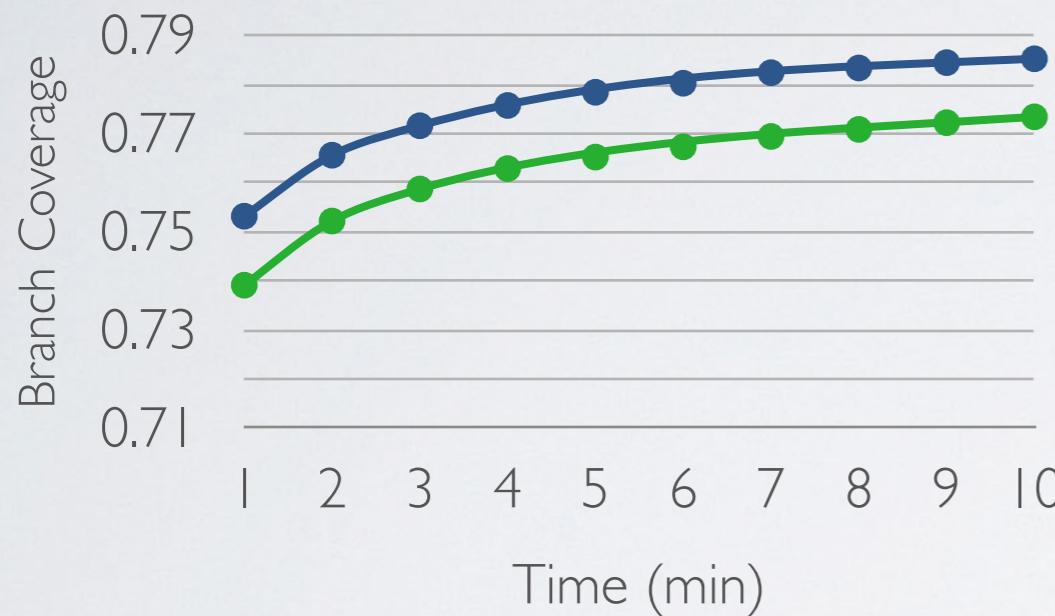
Ouput



INCREASED SEARCH BUDGET

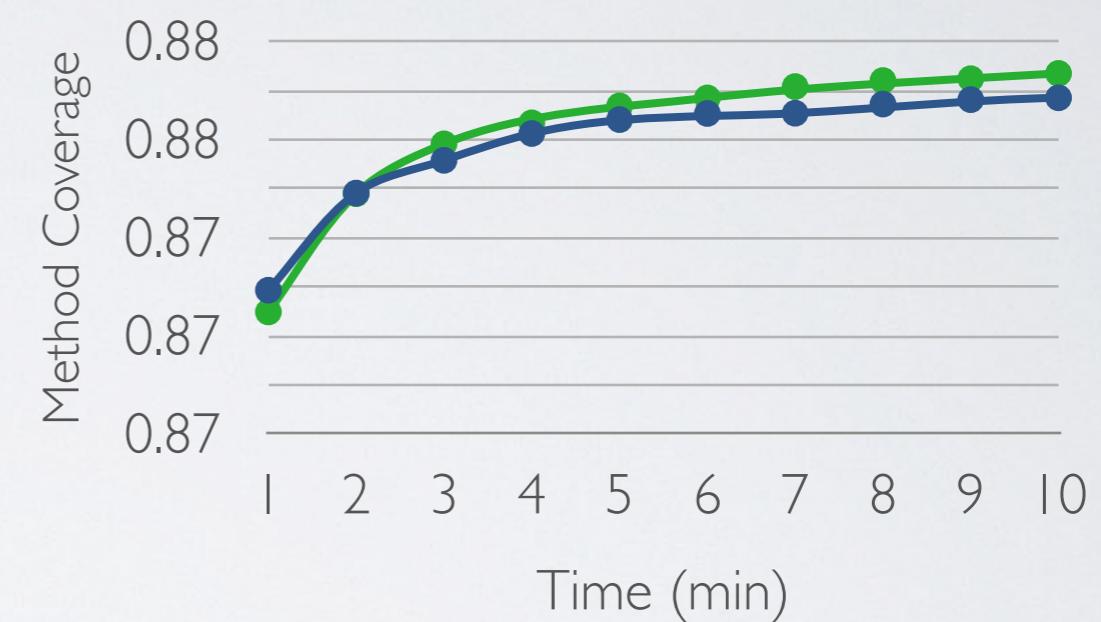
Criterion

Branch

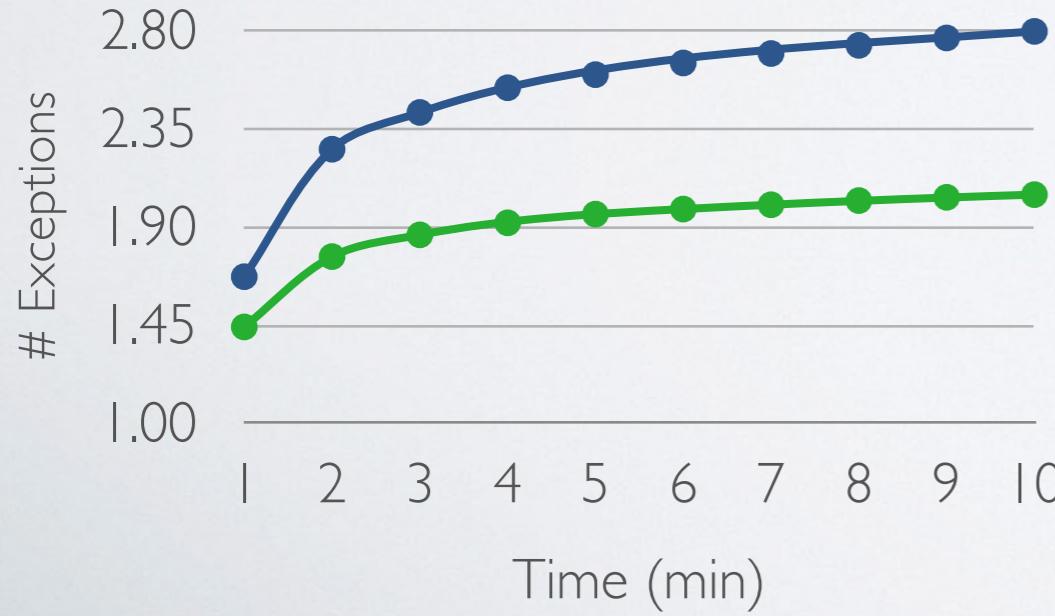


All

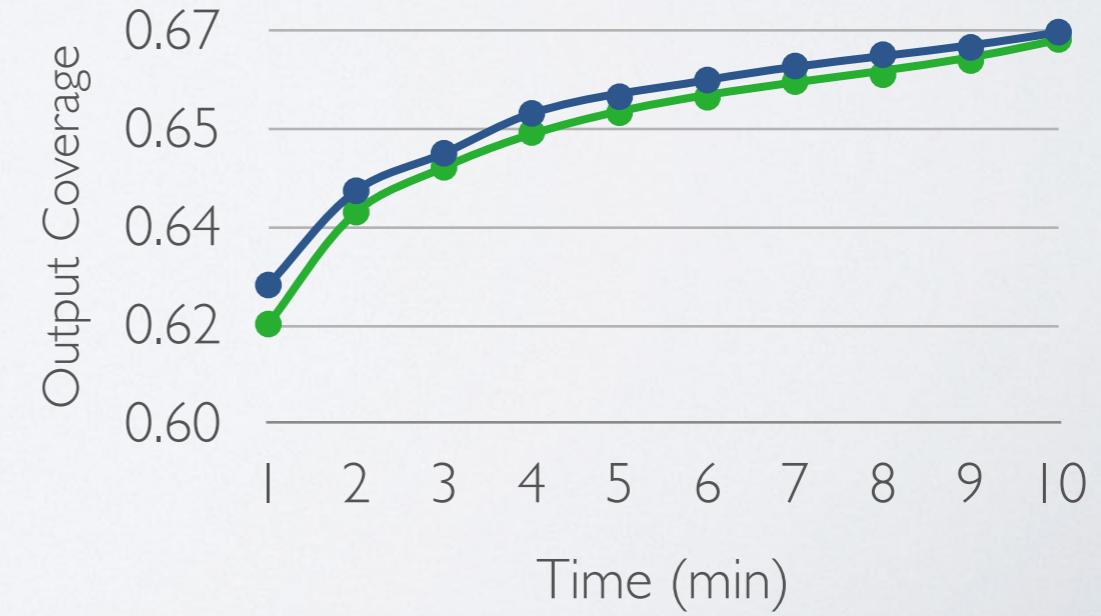
Method



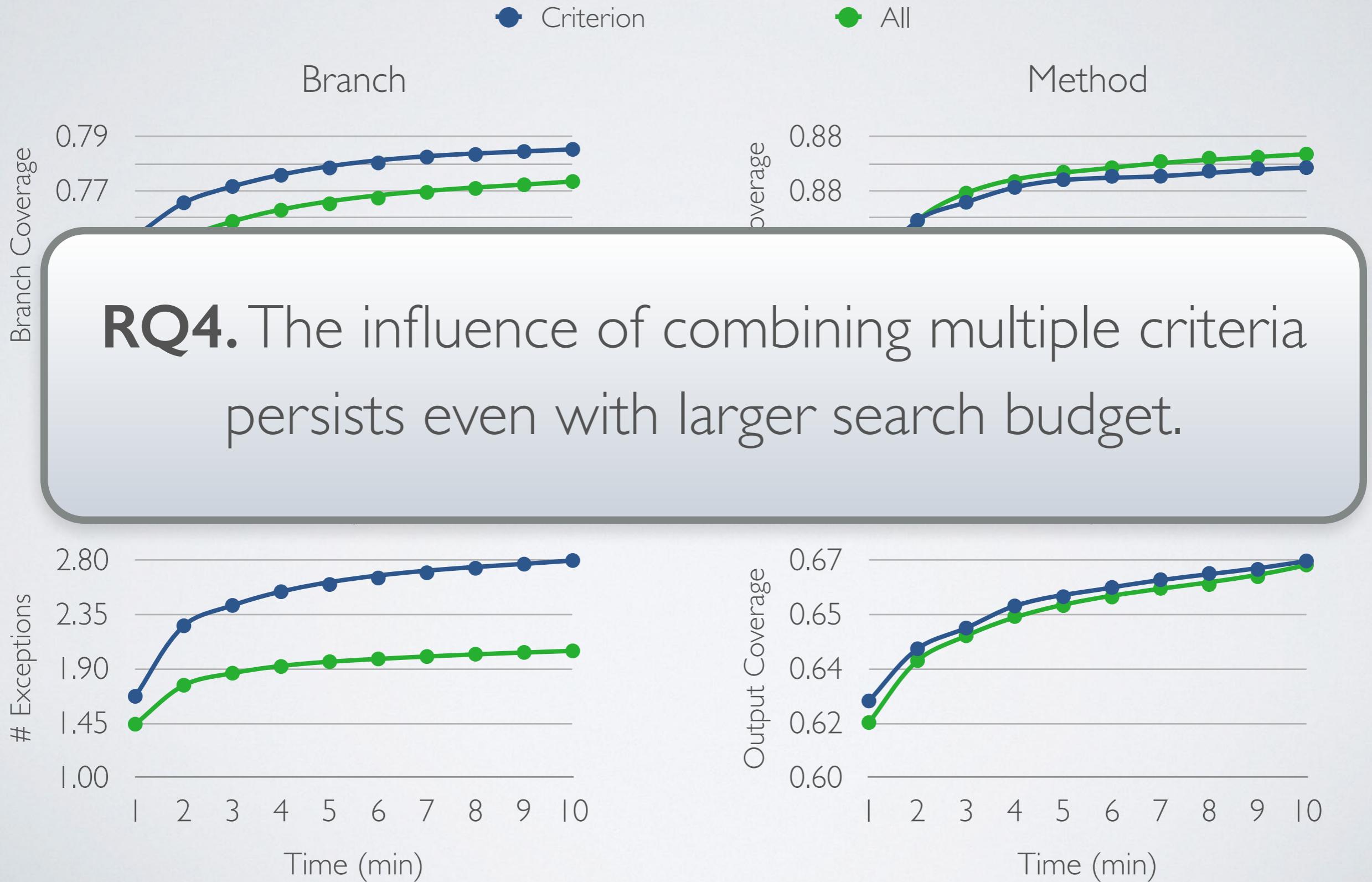
Exception



Ouput



INCREASED SEARCH BUDGET



DEMO

Eclipse plugin update site

<http://www.evosuite.org/update/>

CONCLUSIONS

CONCLUSIONS

- The desired properties of a test suite are multi-faceted

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
 - Combination of multiple criteria is feasible

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
 - Combination of multiple criteria is feasible
 - Really useful for practitioners?

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
 - Combination of multiple criteria is feasible
 - Really useful for practitioners?
- Non-functional criteria: readability, execution time

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
 - Combination of multiple criteria is feasible
 - Really useful for practitioners?
- Non-functional criteria: readability, execution time
 - Conflicting optimisation goals