



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

Java - comparators/src/java/collections/comparators/FixedOrderComparator.java - Eclipse - /Users/jmr/Papers/evosuite-stu...

```
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 be
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 " + t
169      }
170      Object result = map.put(newObj, position);
171      return (result == null);
172  }
173
174  // Comparator methods
175  //-----
```

Writable | Smart Insert | 230 : 33

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(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 be
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 " + t
169      }
170      Object result = map.put(newObj, position);
171      return (result == null);
172  }
173
174  // Comparator methods
175  //-----
```

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(Object unknownObjectBehavior)  
149     checkLocked();  
150     if (unknownObjectBehavior == null  
151         && unknownObjectBehavior != null  
152         && unknownObjectBehavior != null  
153         throw new IllegalArgumentException("unknownObjectBehavior  
154     }  
155     this.unknownObjectBehavior = unknownObjectBehavior;  
156 }  
157  
158 public boolean add(Object object)  
159     checkLocked();  
160     Object position = object;  
161     return (position == null ? true : false);  
162 }  
163  
164 public boolean addA(Object object)  
165     checkLocked();  
166     Integer position = object;  
167     if (position == null  
168         throw new IllegalArgumentException("position  
169     }  
170     Object result = object;  
171     return (result == null ? true : false);  
172 }  
173  
174 // Comparator method  
175 //-----
```

```
20 * This file was automatically generated by EvoSuite.  
5  
6 package collections.comparators;  
7  
8 import static org.junit.Assert.assertEquals;  
20  
21 @RunWith(EvoRunner.class) @EvoRunnerParameters(useVNET = true)  
22 public class FixedOrderComparator_ESTest extends FixedOrderComparator_ESTest_sca  
23  
24 @Test  
25 public void test00() throws Throwable {  
26     Object[] objectArray0 = new Object[5];  
27     Object object0 = new Object();  
28     objectArray0[1] = object0;  
29     FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(object  
30     fixedOrderComparator0.compare(objectArray0[1], object0);  
31     boolean boolean0 = fixedOrderComparator0.isLocked();  
32     assertTrue(boolean0);  
33 }  
34  
35 @Test  
36 public void test01() throws Throwable {  
37     Object[] objectArray0 = new Object[5];  
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     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>unknownObjectBeha
146 * is not valid.
147 */
148 public void setUnkn
149 checkLocked();
150 if (unknownObjec
151     && unknown0
152     && unknown0
153     throw new I
154 }
155 this.unknownObj
156 }
157
158 public boolean add(
159 checkLocked();
160 Object position
161 return (positio
162 }
163
164 public boolean addA
165 checkLocked();
166 Integer positio
167 if (position ==
168     throw new I
169 }
170 Object result =
171 return (result
172 }
173
174 // Comparator metho
175 //-----
```

Optimised for Branch Coverage

```
20 * This file was automatically generated by EvoSuite.
21 package collections.comparators;
22 import static org.junit.Assert.assertEquals;
23
24 @RunWith(EvoRunner.class) @EvoRunnerParameters(useVNET = true)
25 public class FixedOrderComparator_ESTest extends FixedOrderComparator_ESTest_sca
26
27 @Test
28 public void test00() throws Throwable {
29     Object[] objectArray0 = new Object[5];
30     Object object0 = new Object();
31     objectArray0[1] = object0;
32     FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objec
33     fixedOrderComparator0.compare(objectArray0[1], object0);
34     boolean boolean0 = fixedOrderComparator0.isLocked();
35     assertTrue(boolean0);
36 }
37
38 @Test
39 public void test01() throws Throwable {
40     Object[] objectArray0 = new Object[5];
41     Object object0 = new Object();
42     Object object1 = new Object();
43     objectArray0[3] = object1;
44     objectArray0[4] = object0;
45     FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(objec
46     int int0 = fixedOrderComparator0.compare(object1, object0);
47     assertTrue(fixedOrderComparator0.isLocked());
48     assertEquals((-1), int0);
49 }
50
51 @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 pe
145 * @throws IllegalArgumentException if the parameter <code>unknownObjectBeha
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 b
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 //-----
```

Optimised for Branch Coverage

```
FixedOrderComparator_ESTest.java - Eclipse - /Users/jmr/Paper...
...
ly generated by EvoSuite.
...
tors;
ert.assertEquals();
EvoRunnerParameters(useVNET = true)
arator_ESTest extends FixedOrderComparator_ESTest_sca
...
ows Throwable {
= new Object[5];
Object();
ect0;
fixedOrderComparator0 = new FixedOrderComparator(obje
.compare(objectArray0[1], object0);
xedOrderComparator0.isLocked();
...
ows Throwable {
= new Object[5];
Object object0 = new Object();
Object object1 = new Object();
objectArray0[3] = object1;
objectArray0[4] = object0;
FixedOrderComparator fixedOrderComparator0 = new FixedOrderComparator(obje
int int0 = fixedOrderComparator0.compare(object1, object0);
assertTrue(fixedOrderComparator0.isLocked());
assertEquals((-1), int0);
}
@Test
...
Writable Smart Insert 2 : 1
```

Automated Unit Test Generation

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

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

@Test
void test9(){
    ArrayIntList l =
        new ArrayIntList();
    // 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 ArrayIntList
public int set(int idx,
               int element){
    checkRange(idx);
    incrModCount();
    int oldval = _data[idx];
    _data[idx] = element;
    return oldval;
}

@Test
void test9(){
    ArrayIntList l =
        new ArrayIntList();
    // 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();
}
```

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

IS 100% BRANCH COVERAGE ENOUGH?

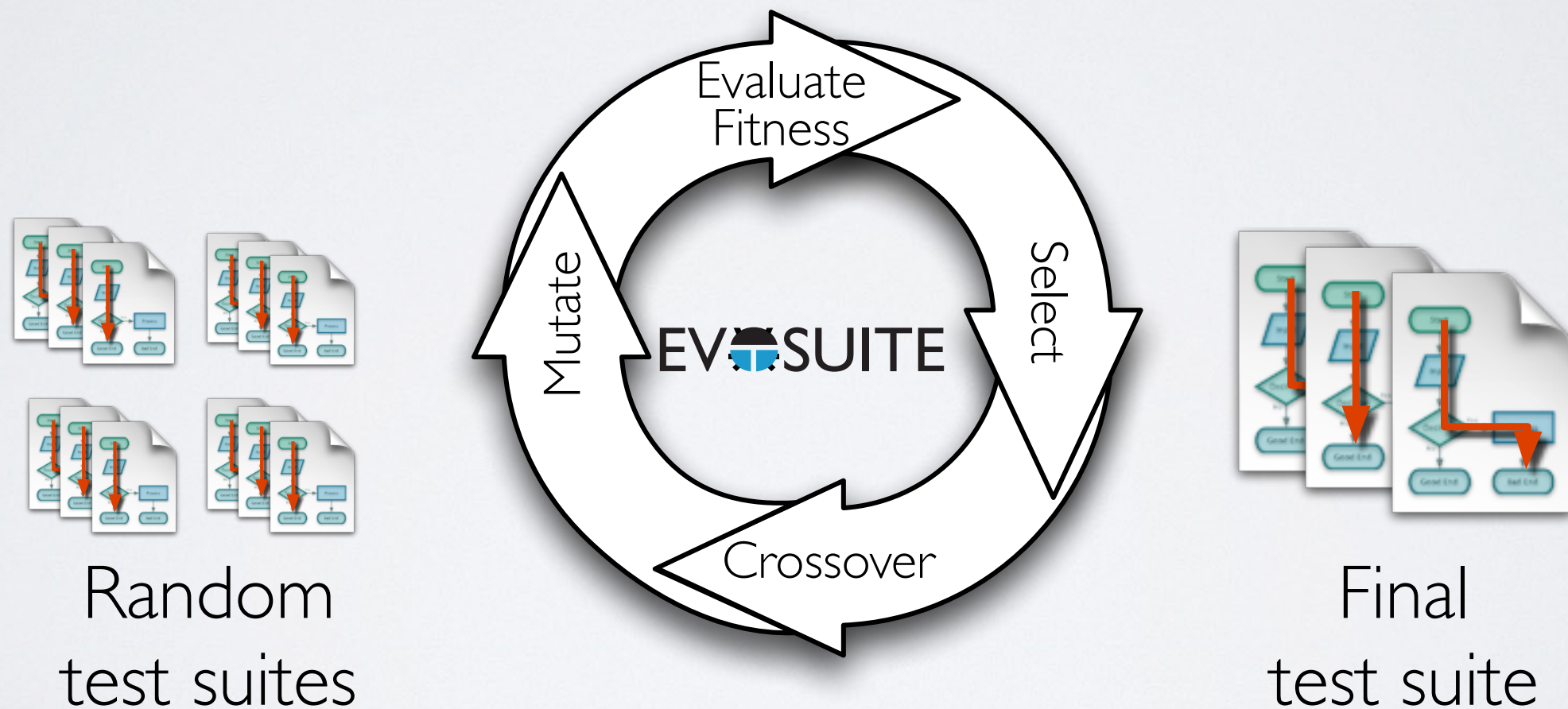
```
// Class Complex
public Complex log(){
    if (isNaN) return NaN;
    double r = log(abs());
    double i;
    i = atan2(imaginary, real);
    return createComplex(r, i);
}
public Complex pow(double x){
    Complex c = this.log();
    return c.multiply(x).exp();
}
```

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

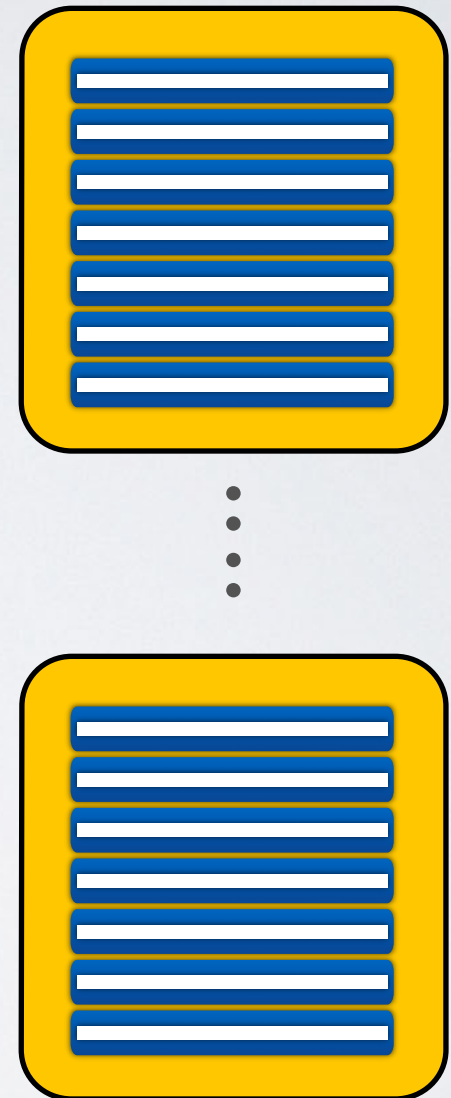
WHOLE TEST SUITE GENERATION



FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

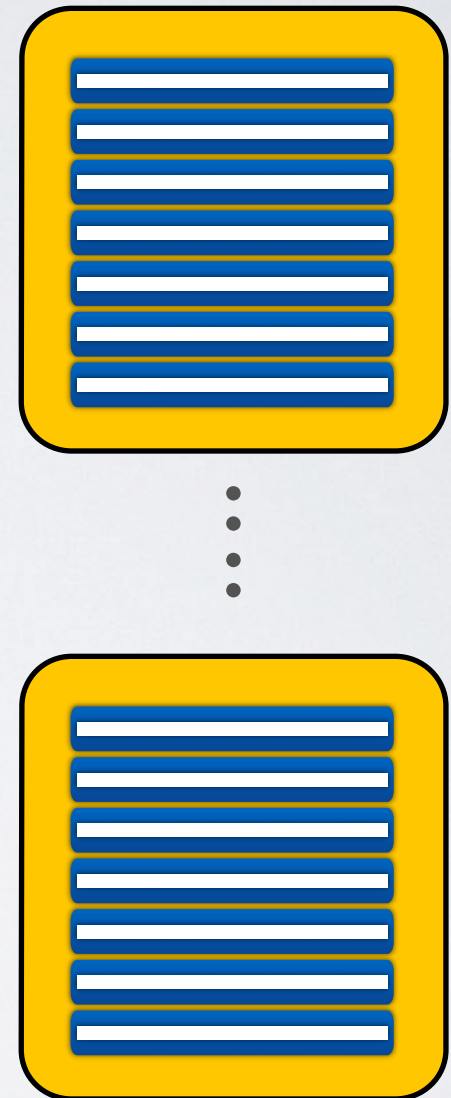
```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
    throws NullArgumentException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



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();
}
```

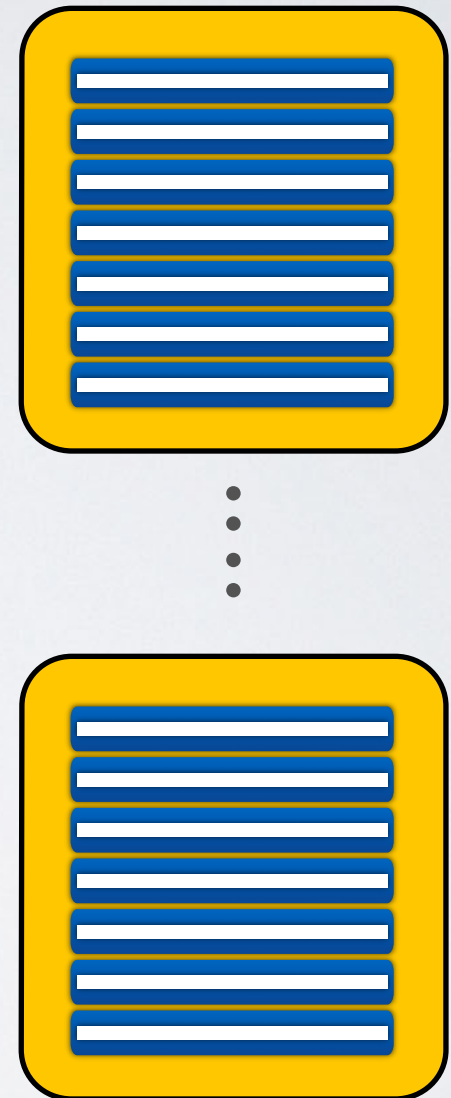


Top-level Method Coverage No exception

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

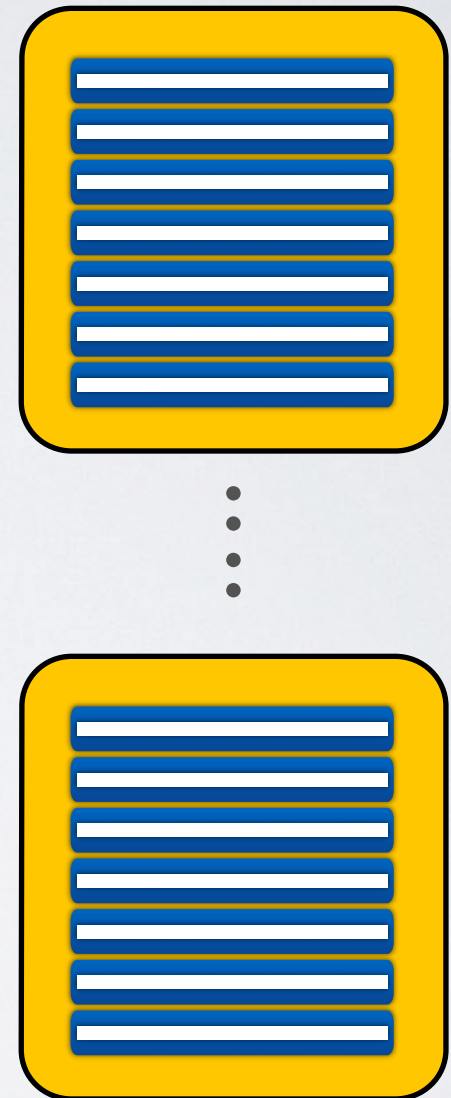


Line Coverage

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();
}
```

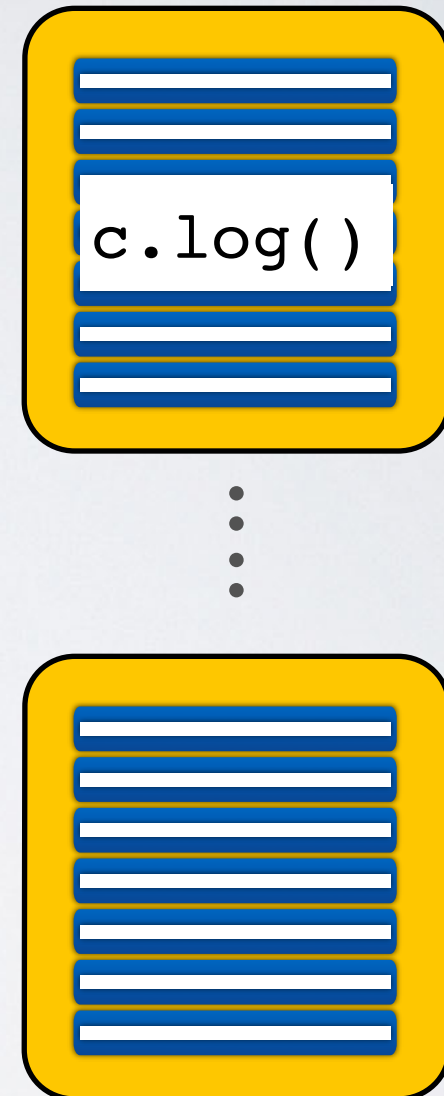


Branch Coverage

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

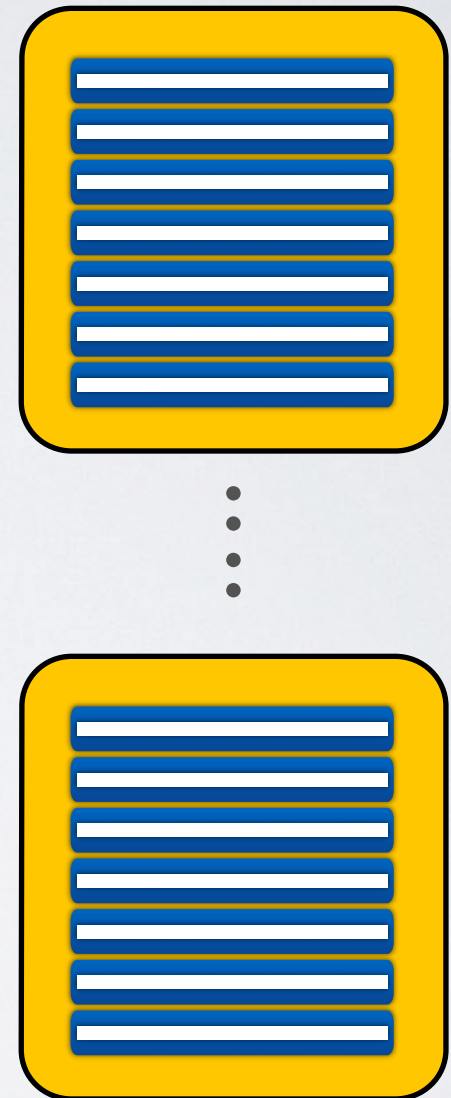


Direct Branch Coverage

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();
}
```



Weak Mutation

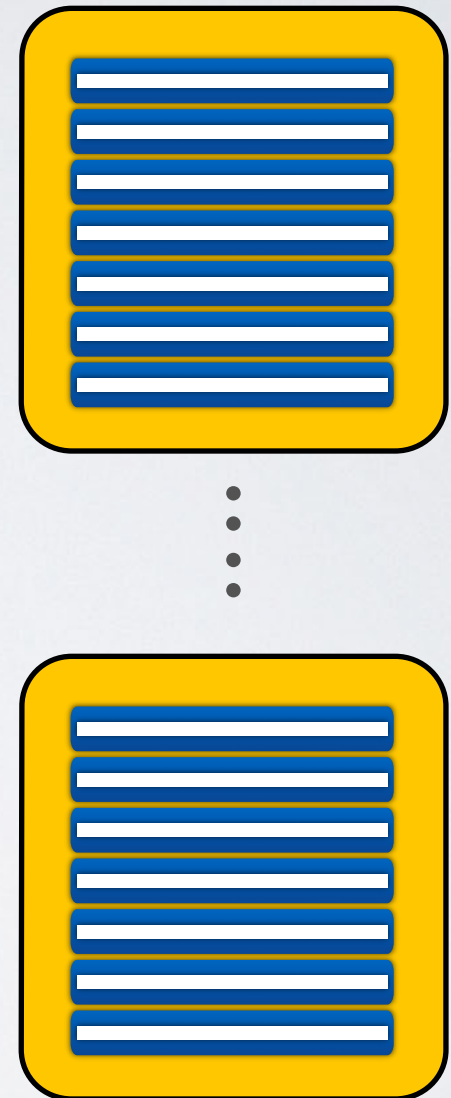
FITNESS FUNCTIONS

(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();
}
```

$\left\{ \begin{array}{l} - \\ 0 \\ + \end{array} \right.$

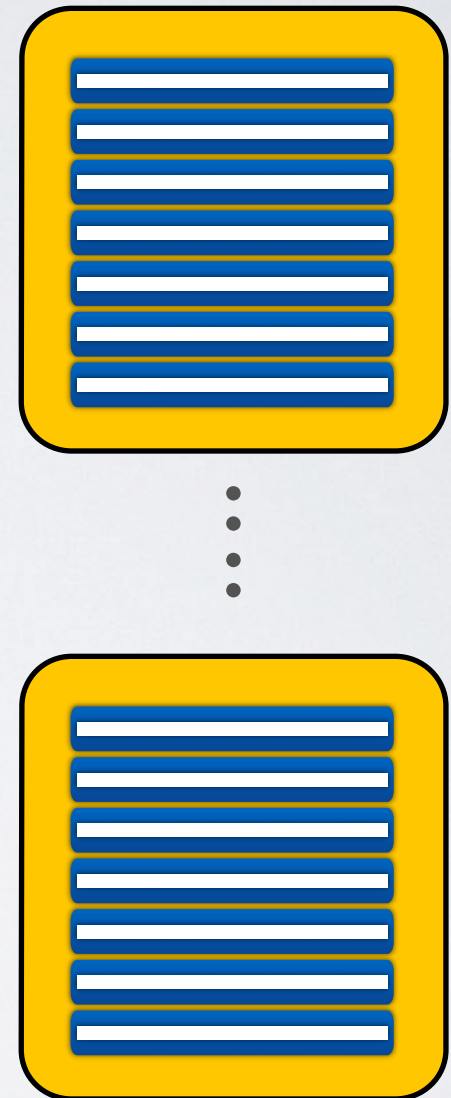
$\left\{ \begin{array}{l} \text{null} \\ \text{nonnull} \end{array} \right.$



FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

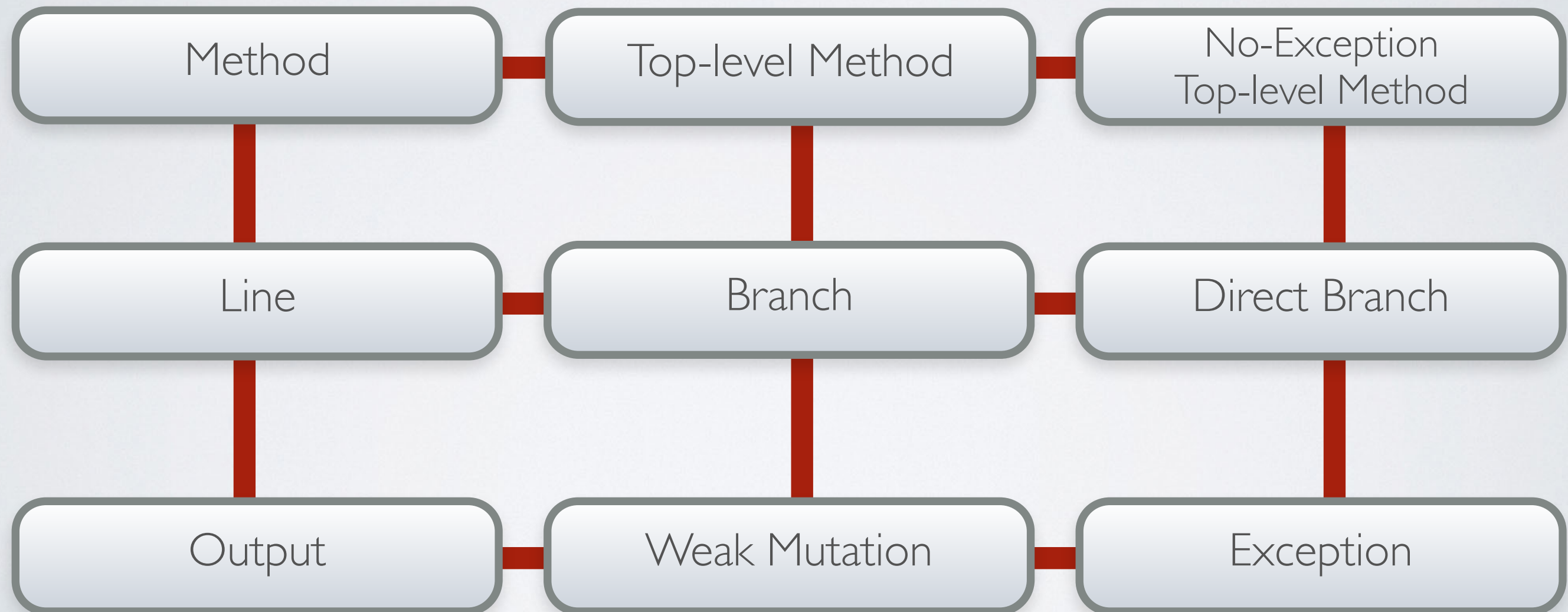
```
// Class Complex
public Complex log(){
1  if (isNaN) return NaN;
2  double r = log(abs());
3  double i;
4  i = atan2(imaginary, real);
5  return createComplex(r, i);
}
public Complex pow(double x)
  throws NullPointerException{
6  Complex c = this.log();
7  return c.multiply(x).exp();
}
```



Number of exceptions

FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA



FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA

Method

Top-level Method

No-Exception
Top-level Method

$$fitness(Suite) = \sum_{i=1}^n w_i \times f_i$$

bench

Output

Weak Mutation

Exception

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

EV**SUITE**

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

EXPERIMENTAL SETUP

EVOSUITE

SFI10 Corpus

| 110 Projects
+20,000 Classes

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

| 110 Projects
+20,000 Classes



650

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

| 110 Projects
+20,000 Classes



650



10

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects
+20,000 Classes



650



10



2/10

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus | 110 Projects
+20,000 Classes



650



10



2/10



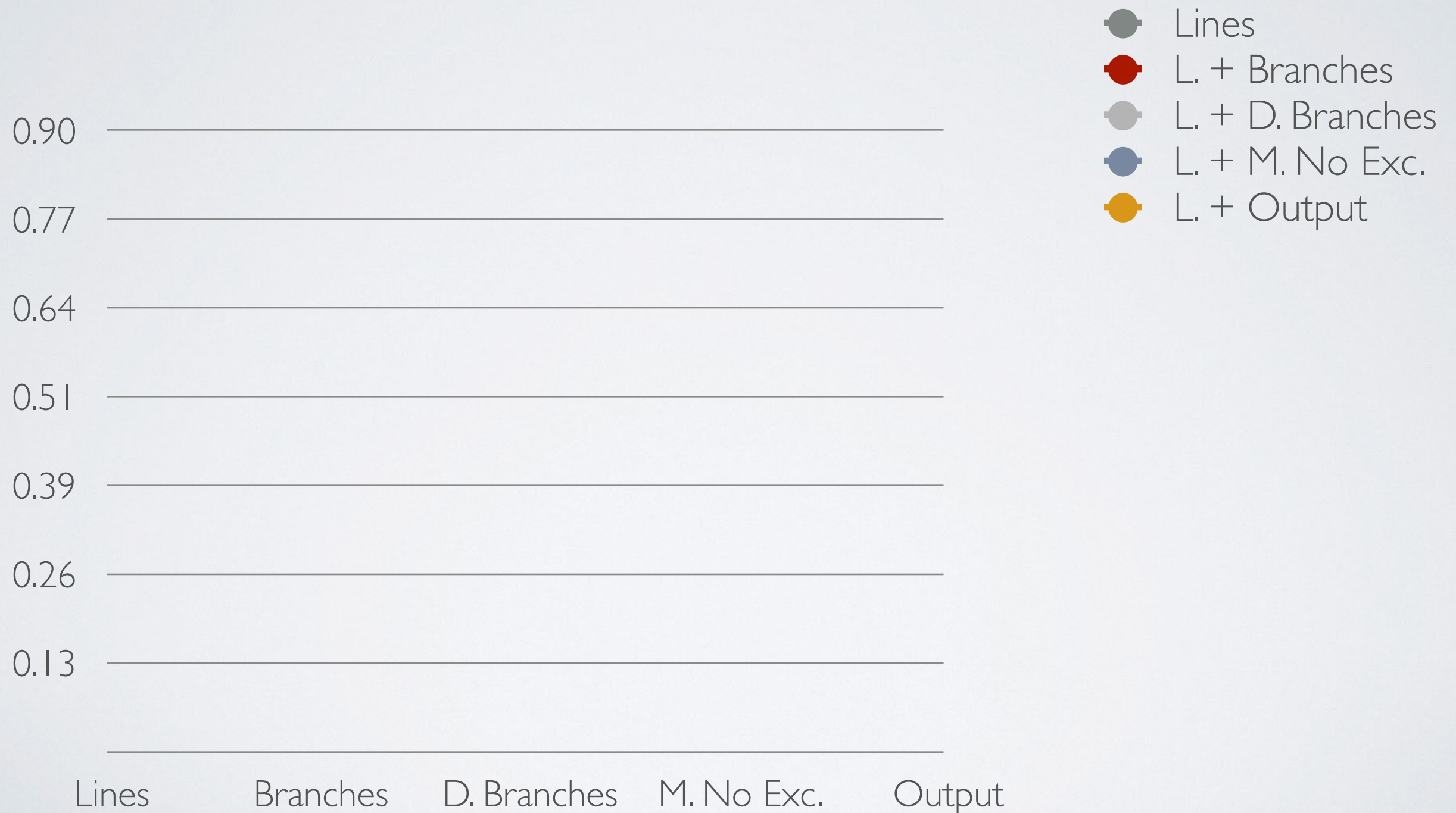
40/5

TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion

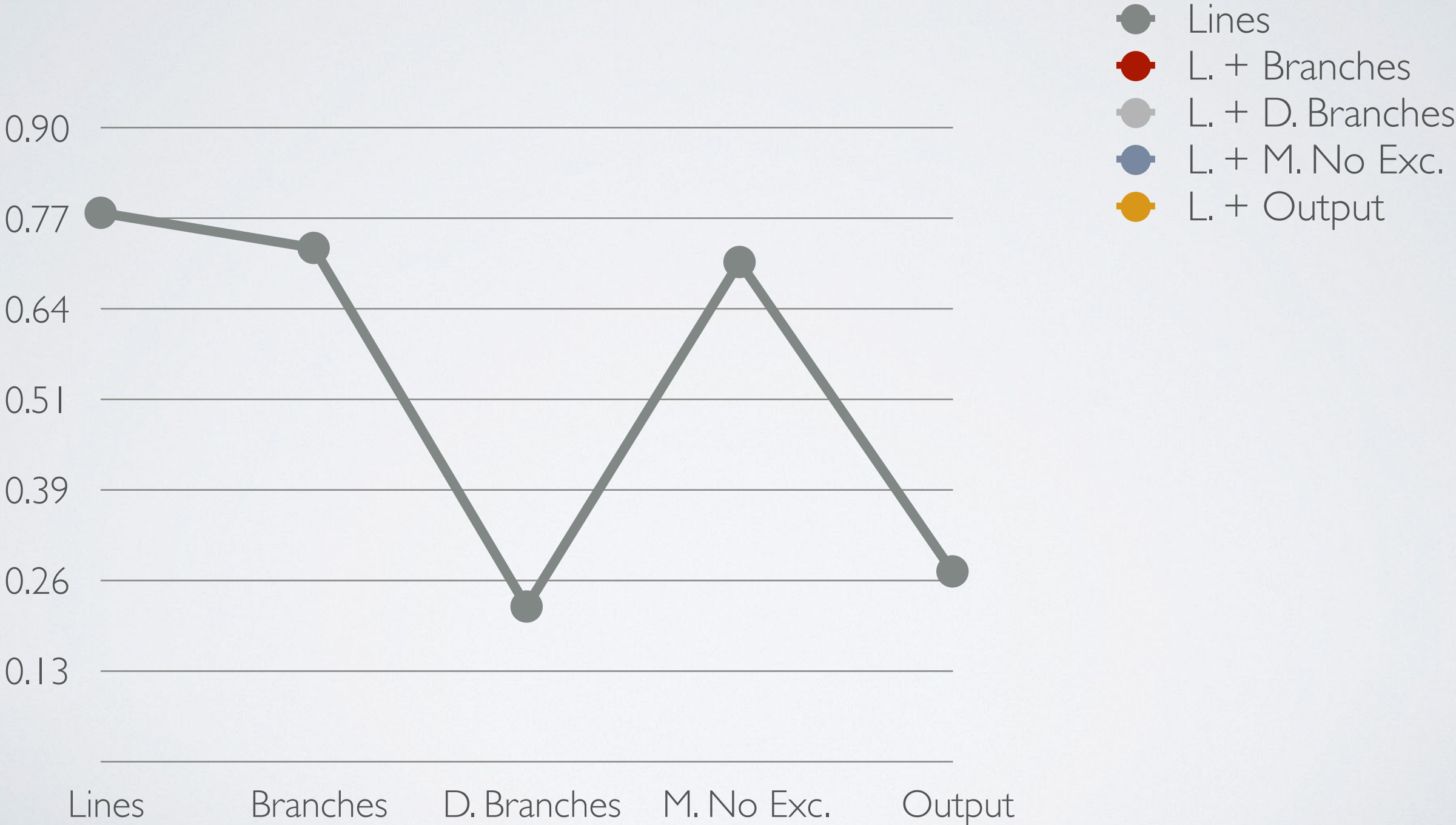
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



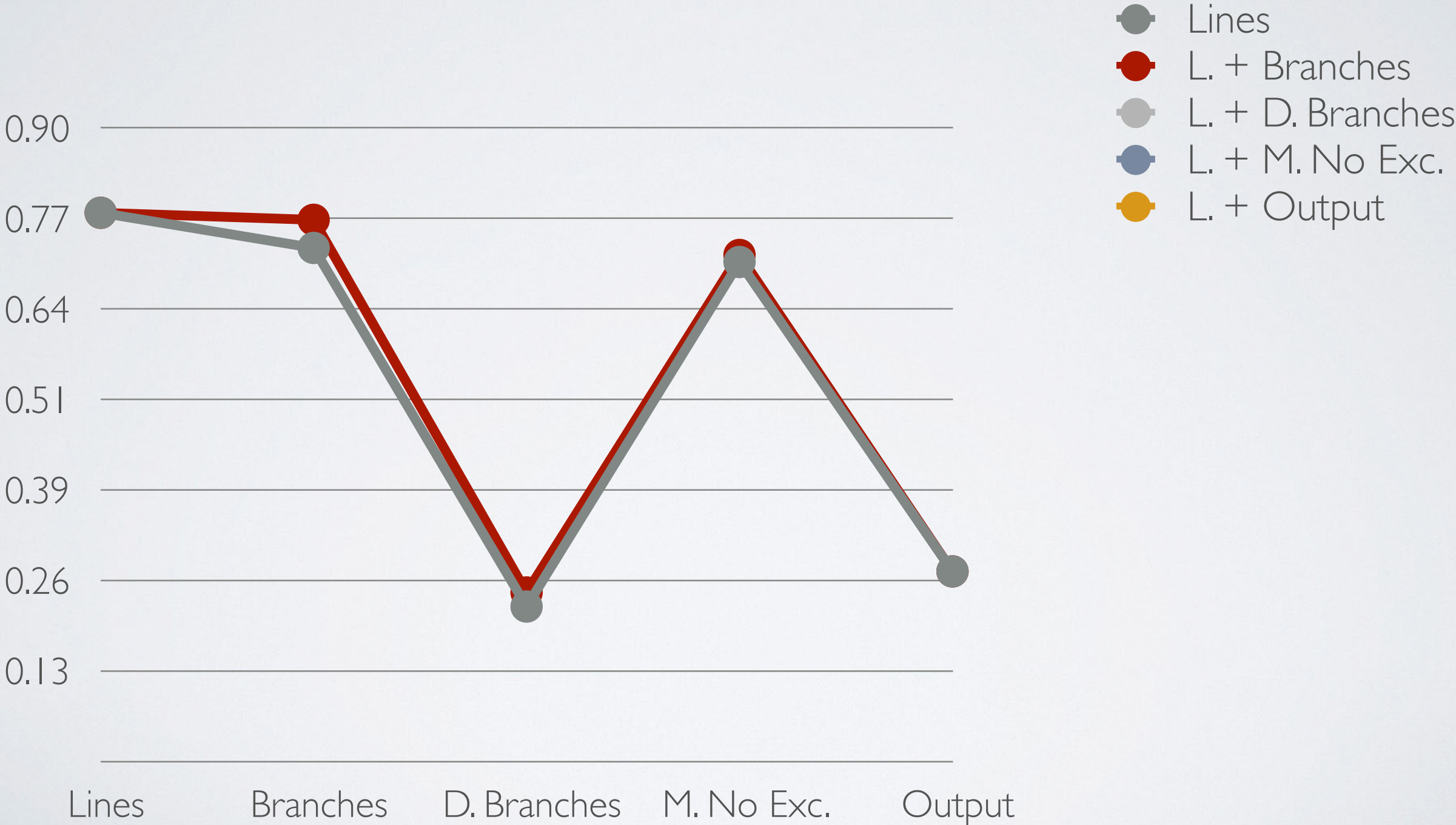
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



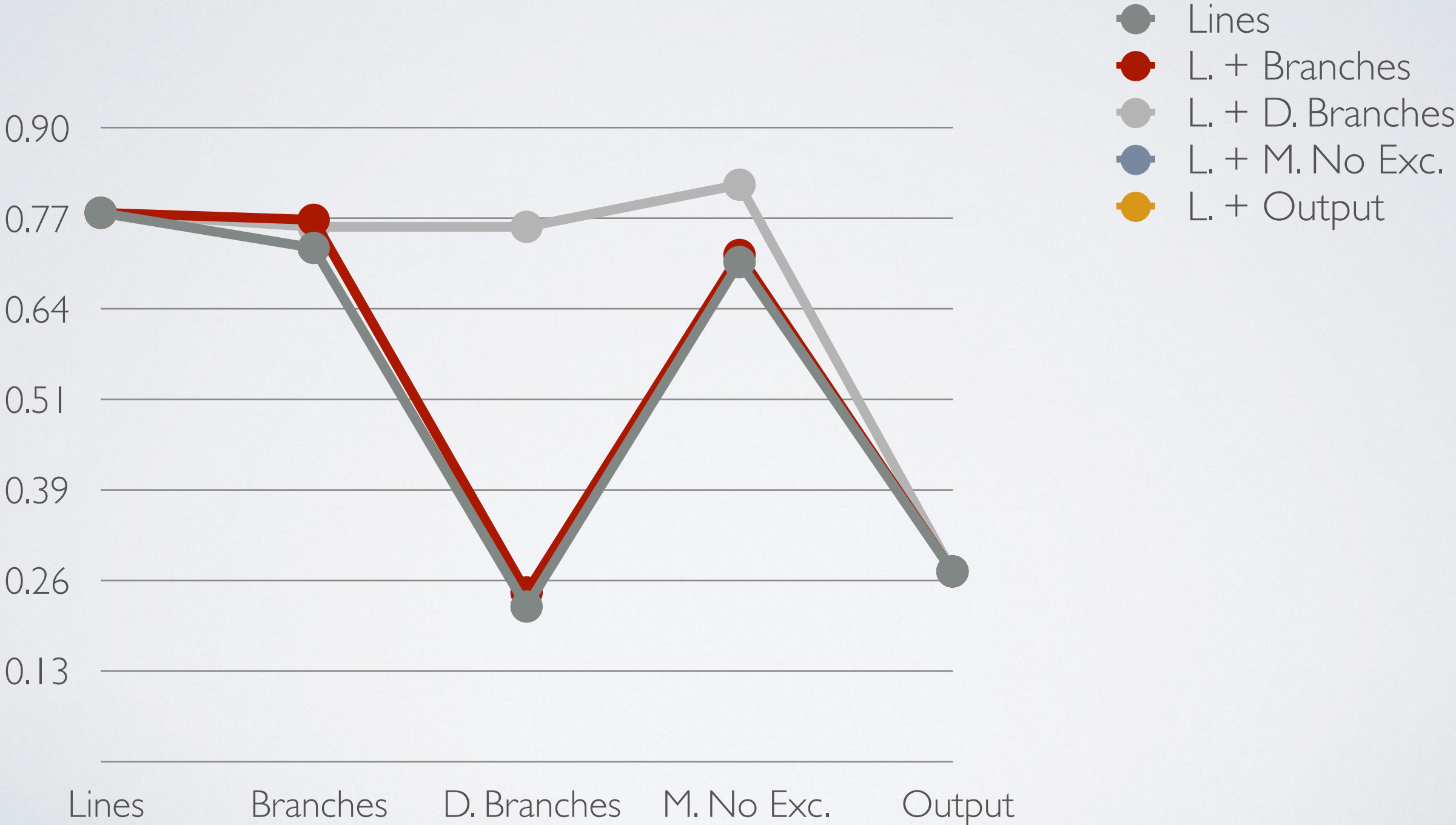
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



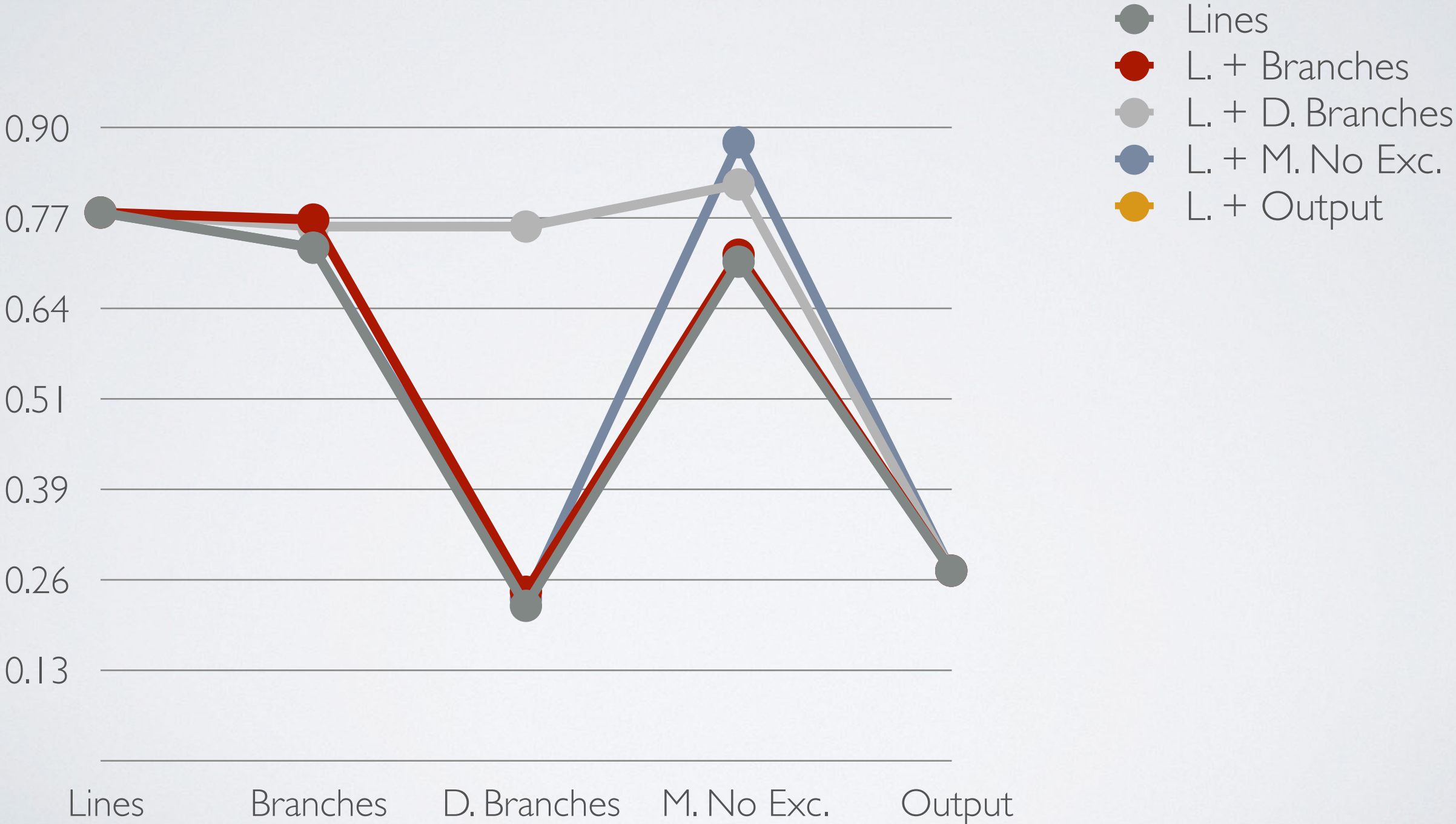
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



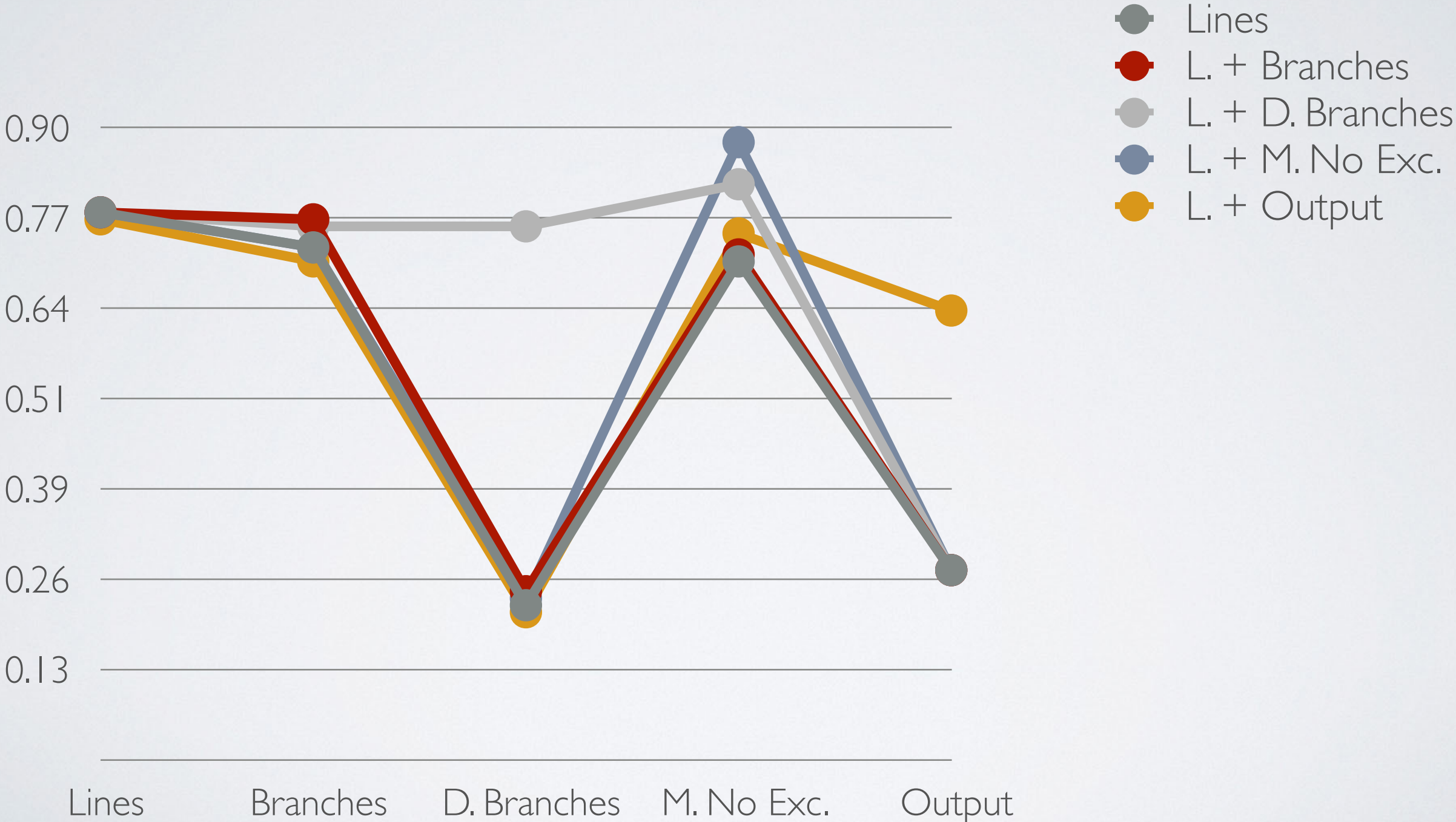
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



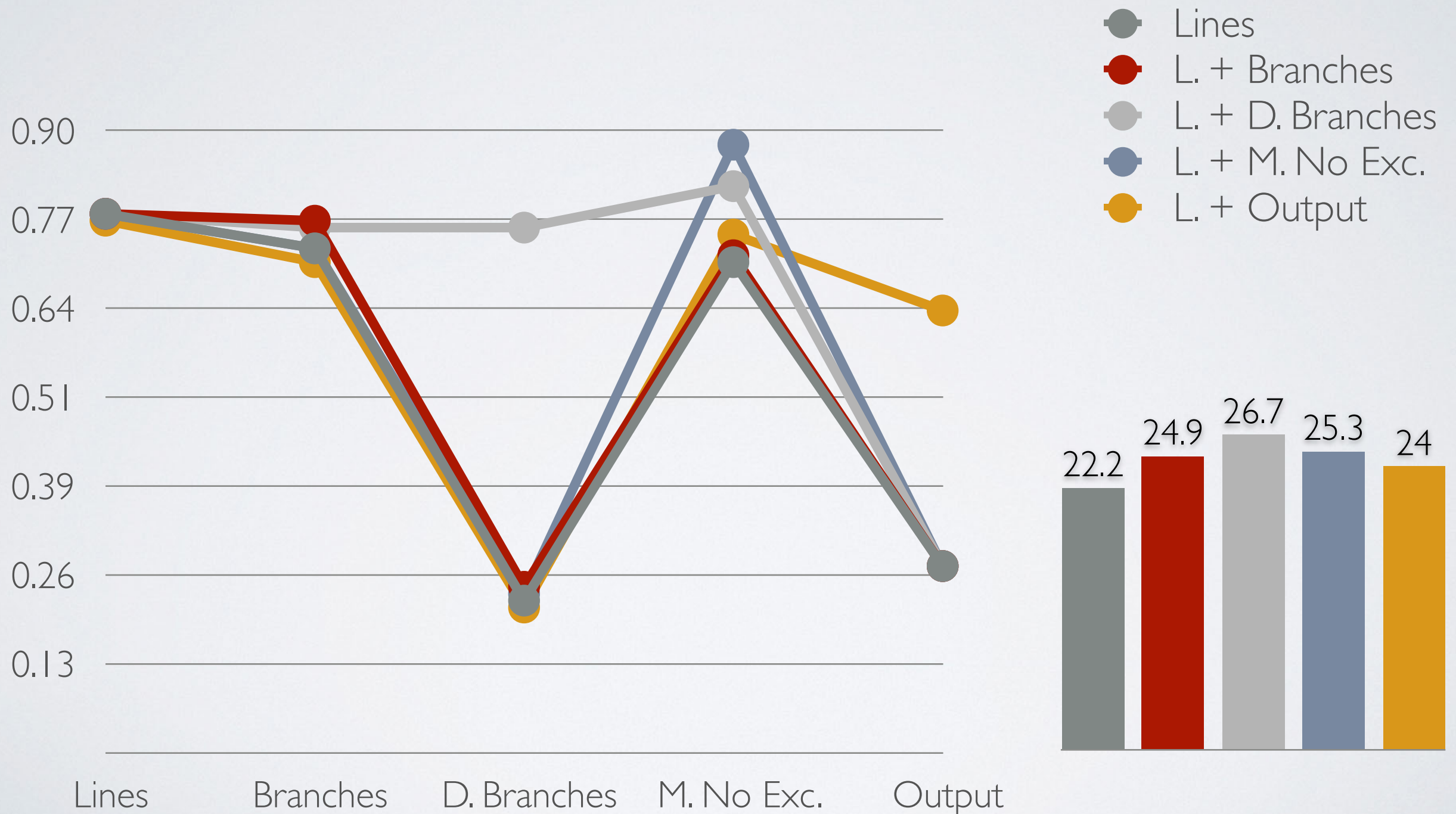
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion

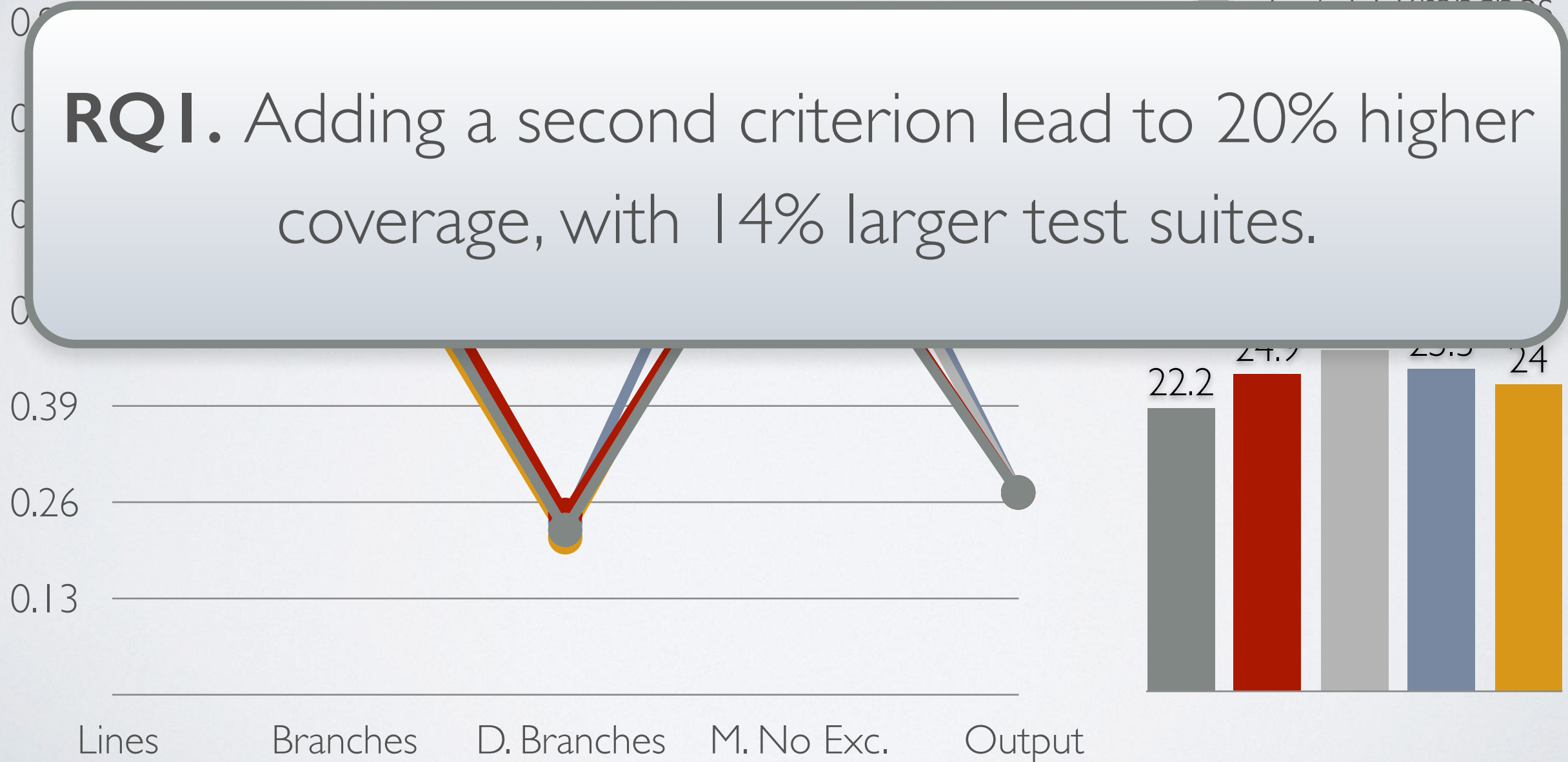


TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion

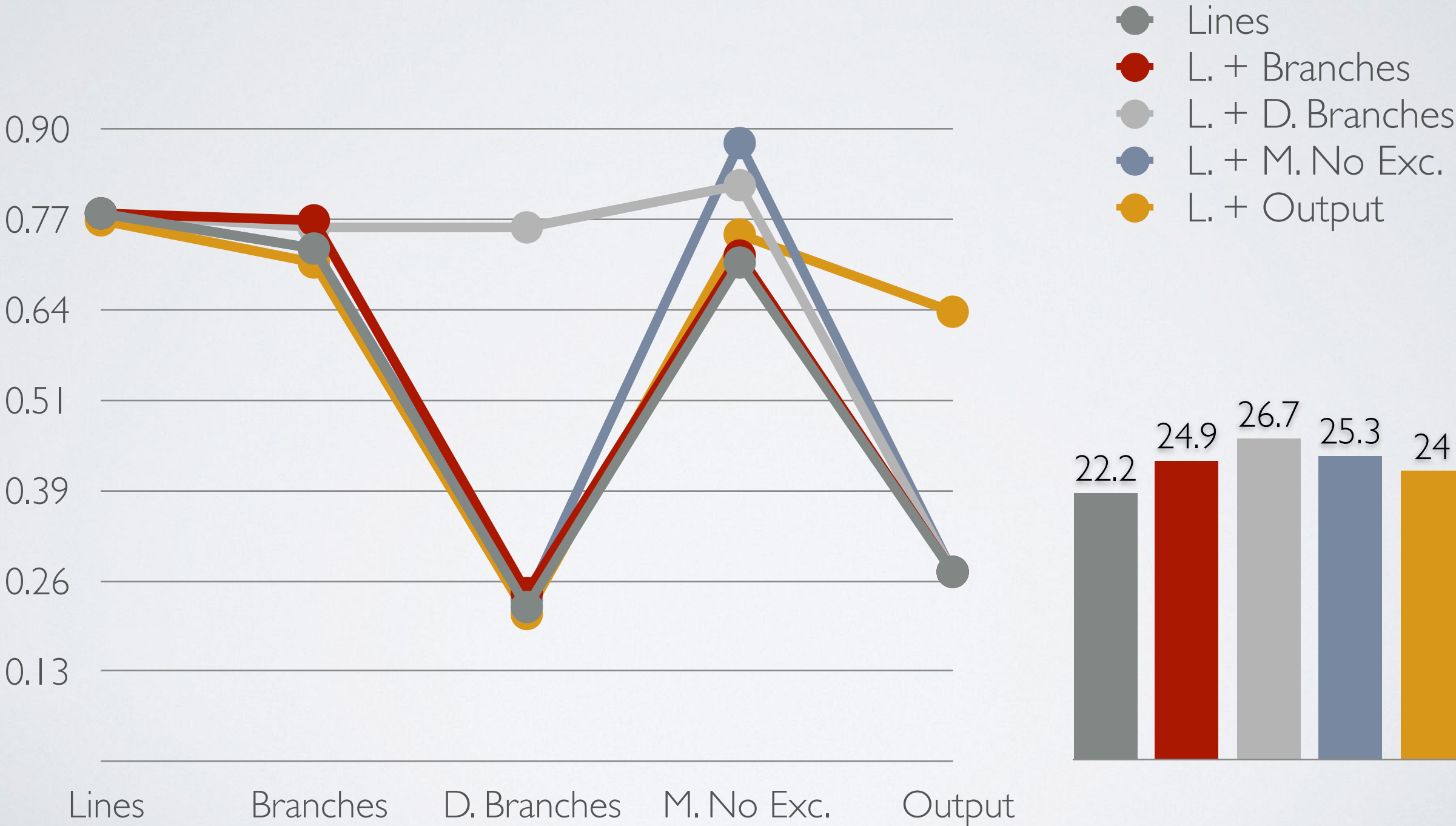
- Lines
- L. + Branches
- L. + D. Branches

RQI. Adding a second criterion lead to 20% higher coverage, with 14% larger test suites.



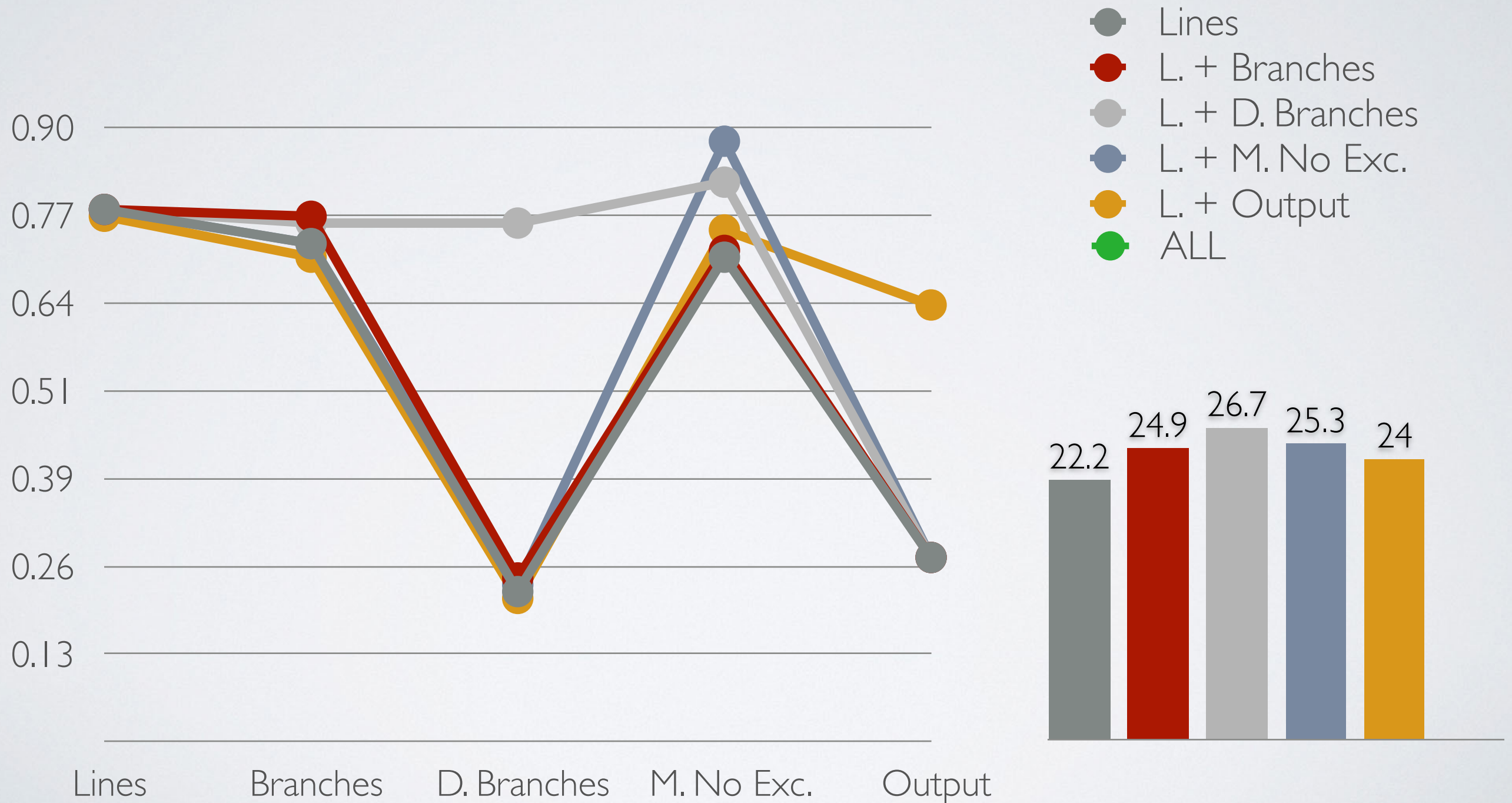
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



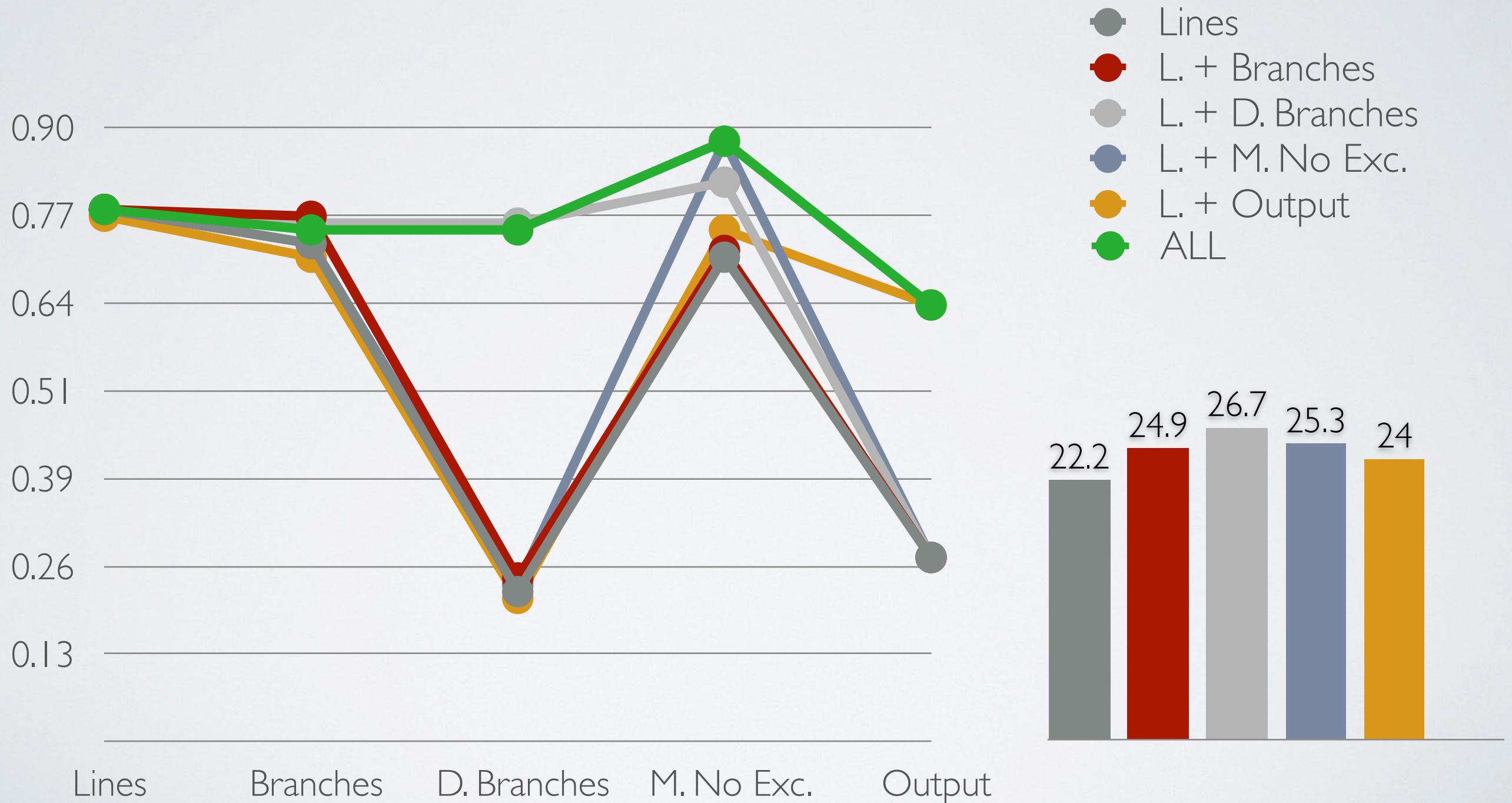
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



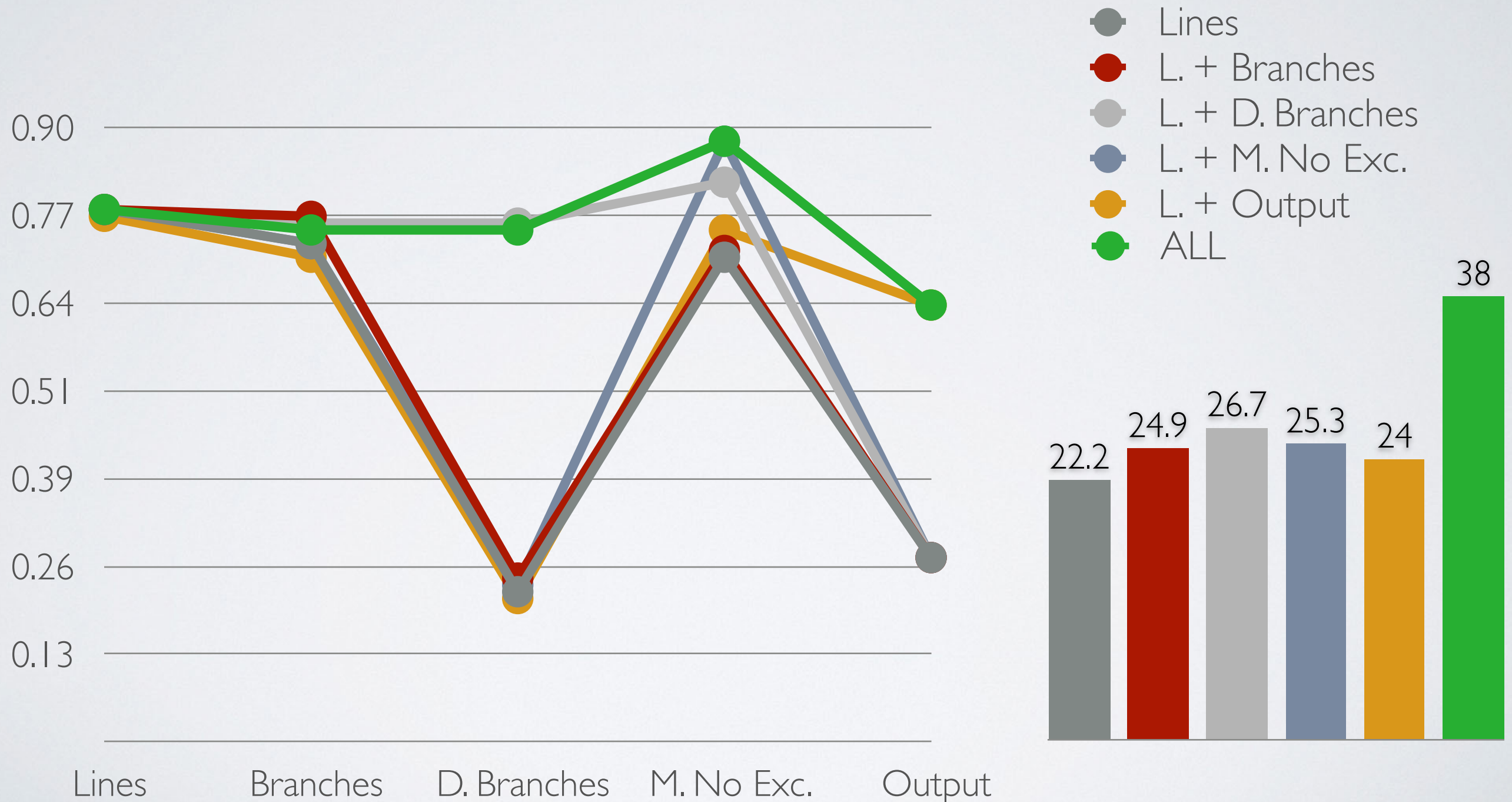
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria

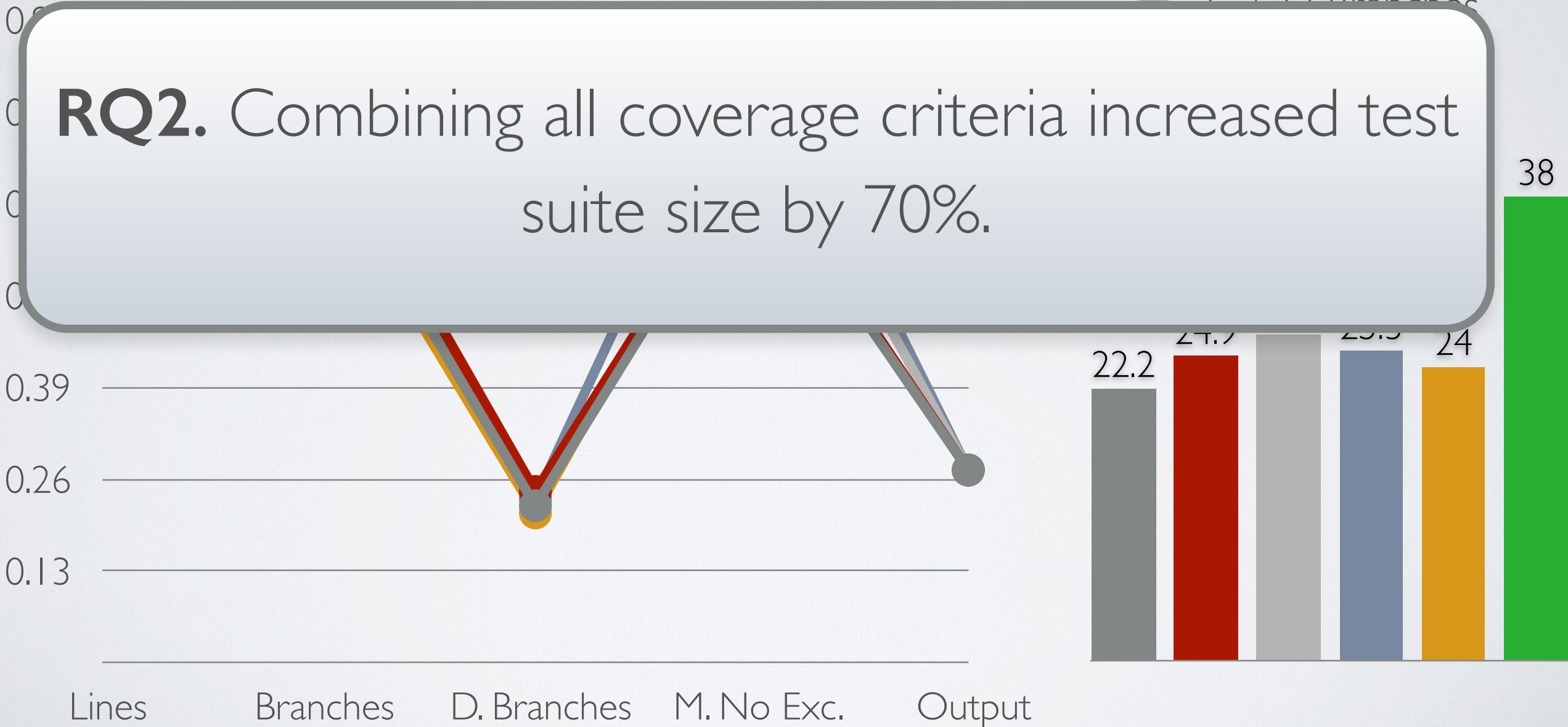


TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria

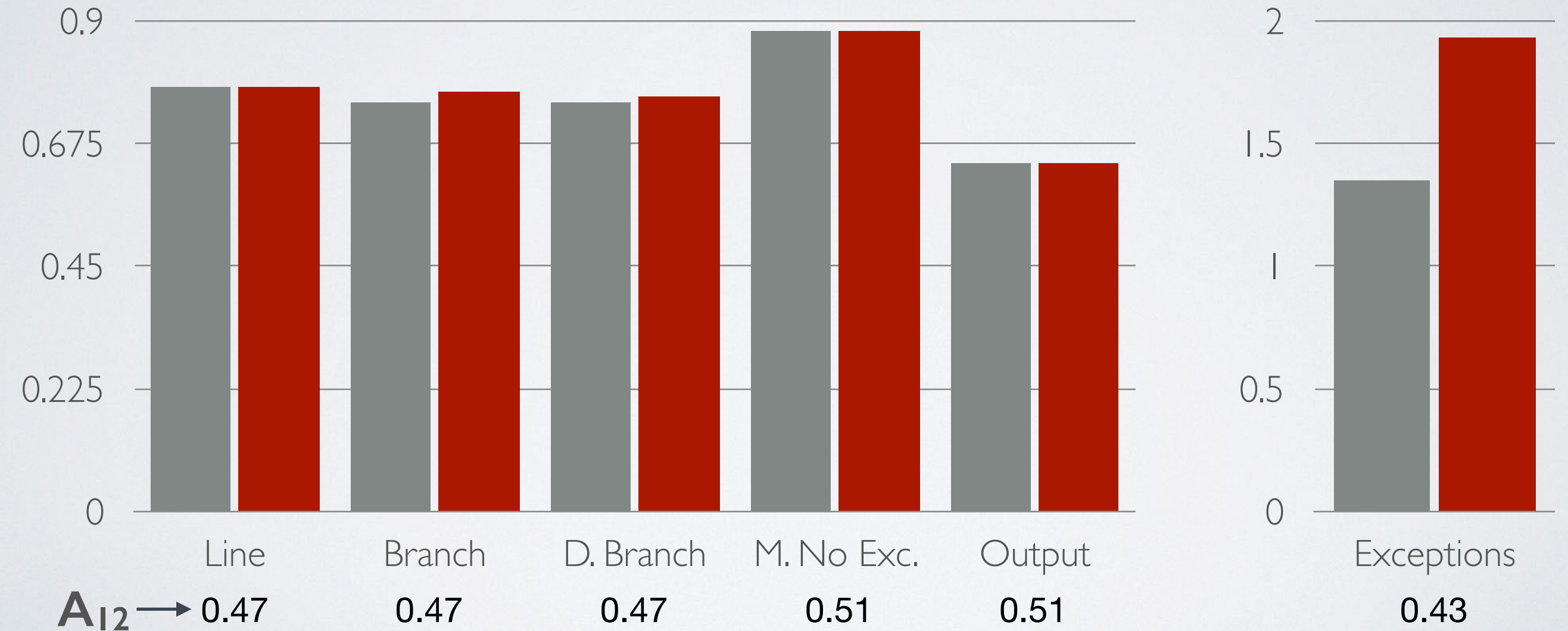
- Lines
- L. + Branches
- L. + D. Branches

RQ2. Combining all coverage criteria increased test suite size by 70%.



COVERAGE OF CONSTITUENT CRITERIA

■ All ■ Line + Criterion

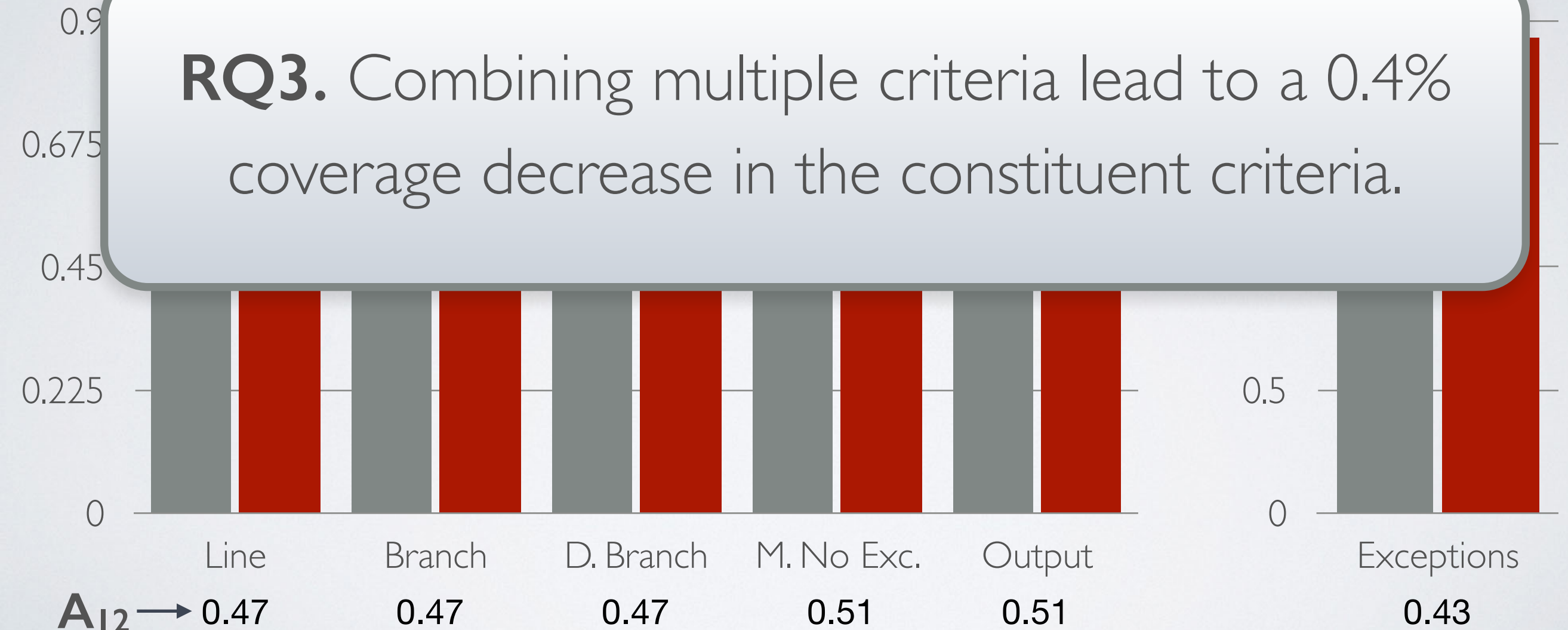


p -values ≤ 0.001

COVERAGE OF CONSTITUENT CRITERIA

■ All ■ Line + Criterion

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



$A_{12} \rightarrow 0.47$

$p\text{-values} \leq 0.001$

INCREASED SEARCH BUDGET

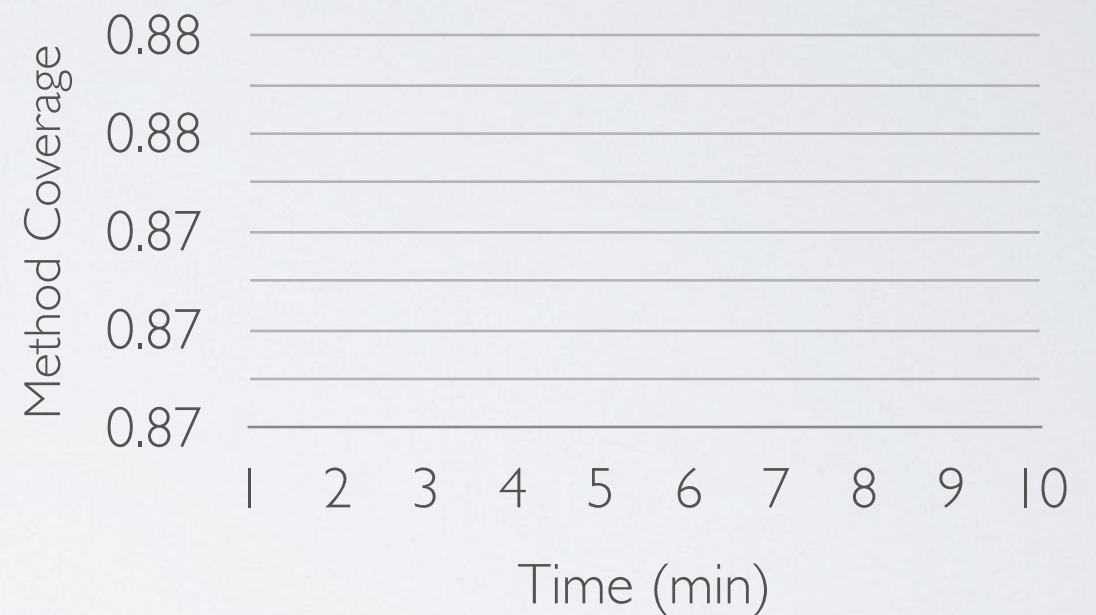
● Criterion

● All

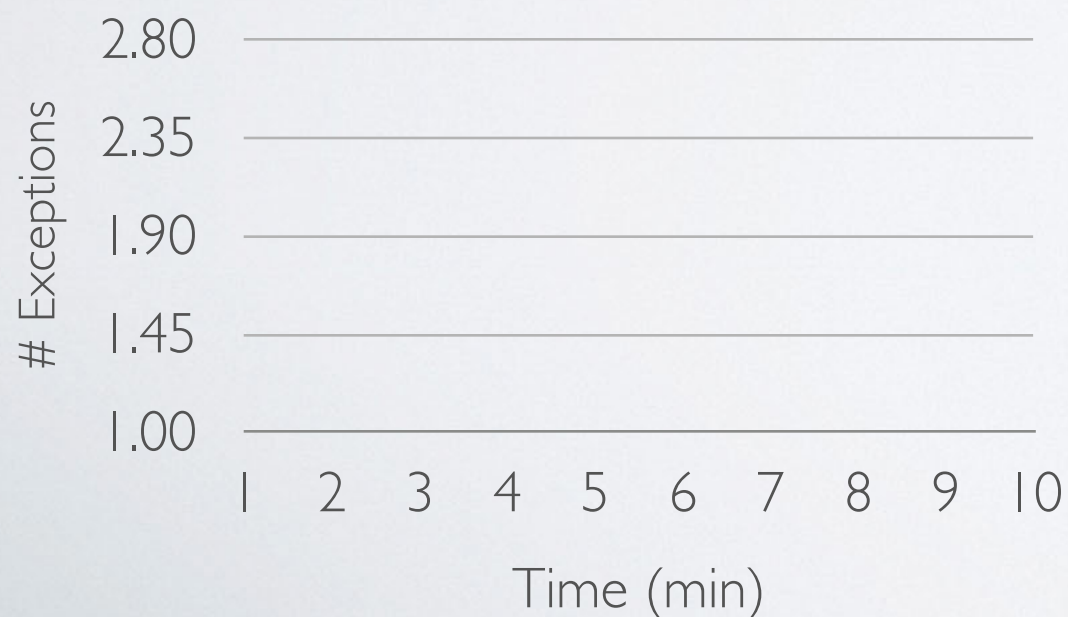
Branch



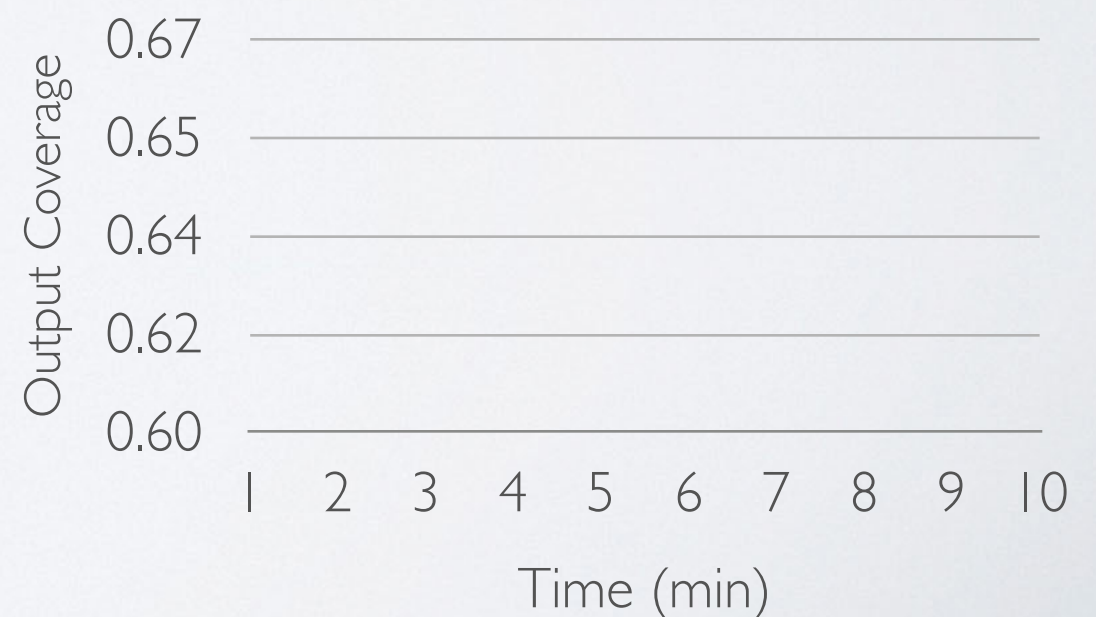
Method



Exception



Ouput

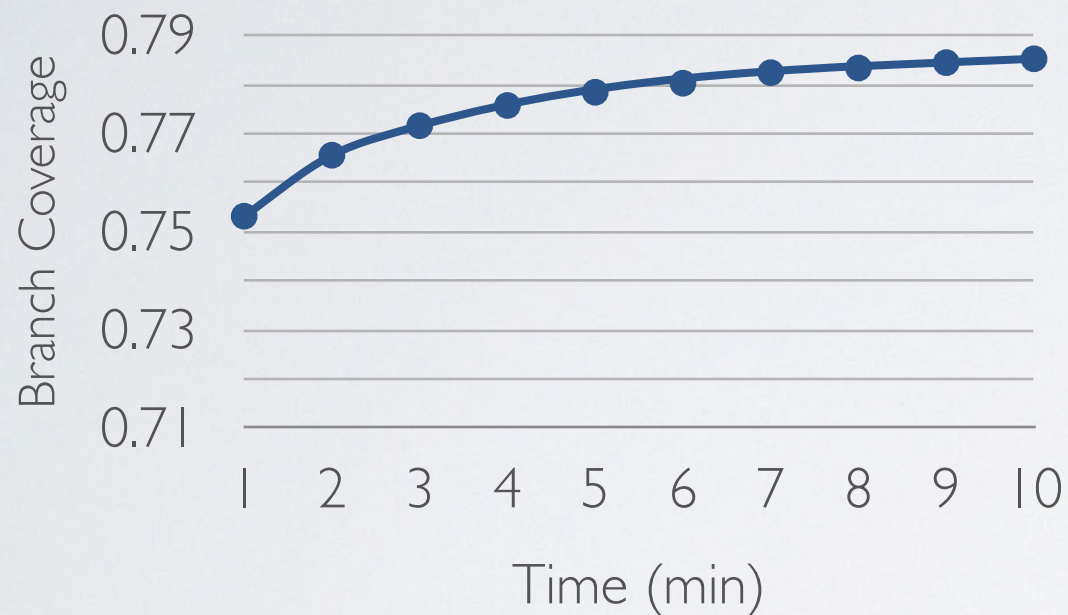


INCREASED SEARCH BUDGET

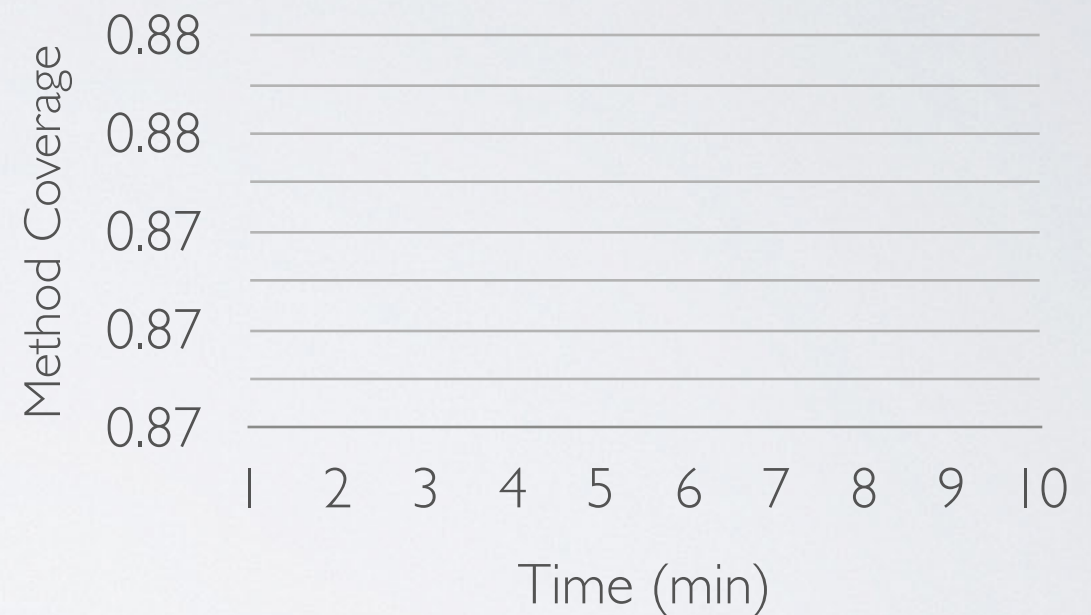
● Criterion

● All

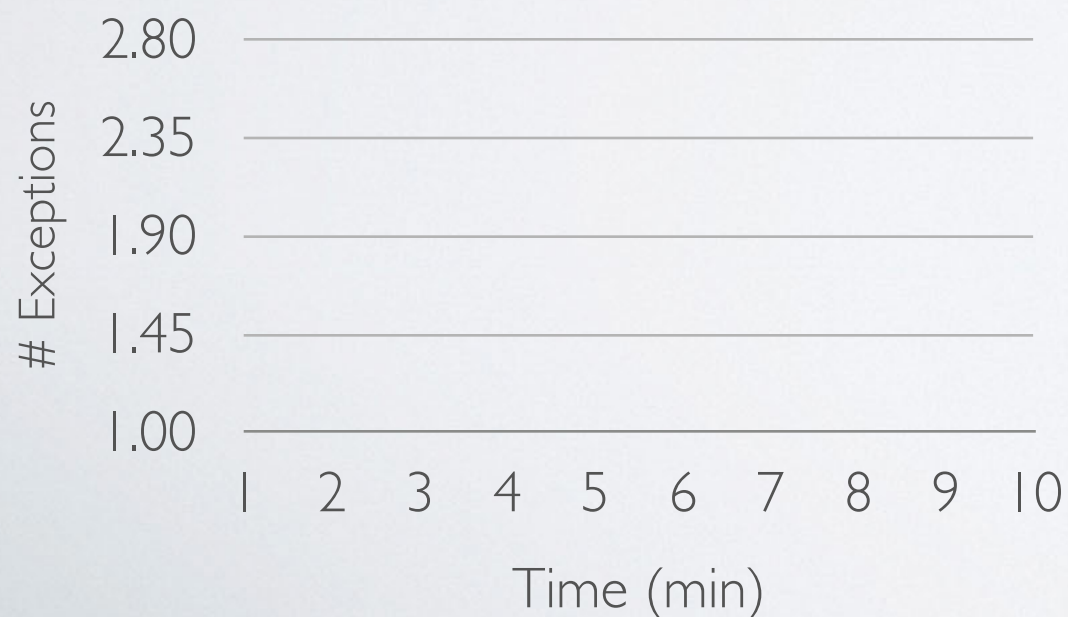
Branch



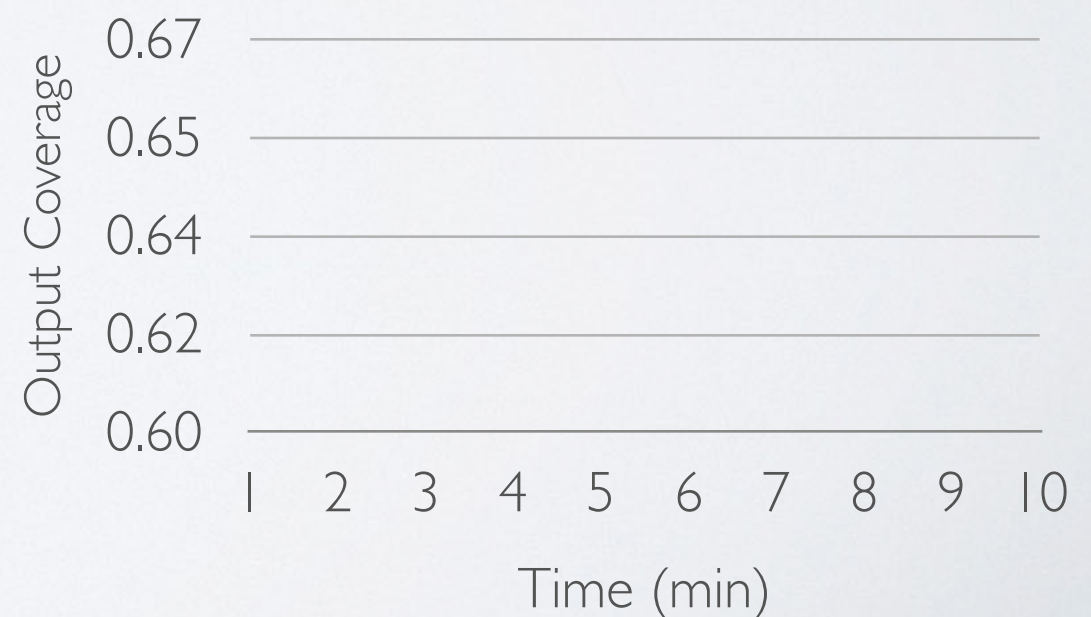
Method



Exception



Ouput

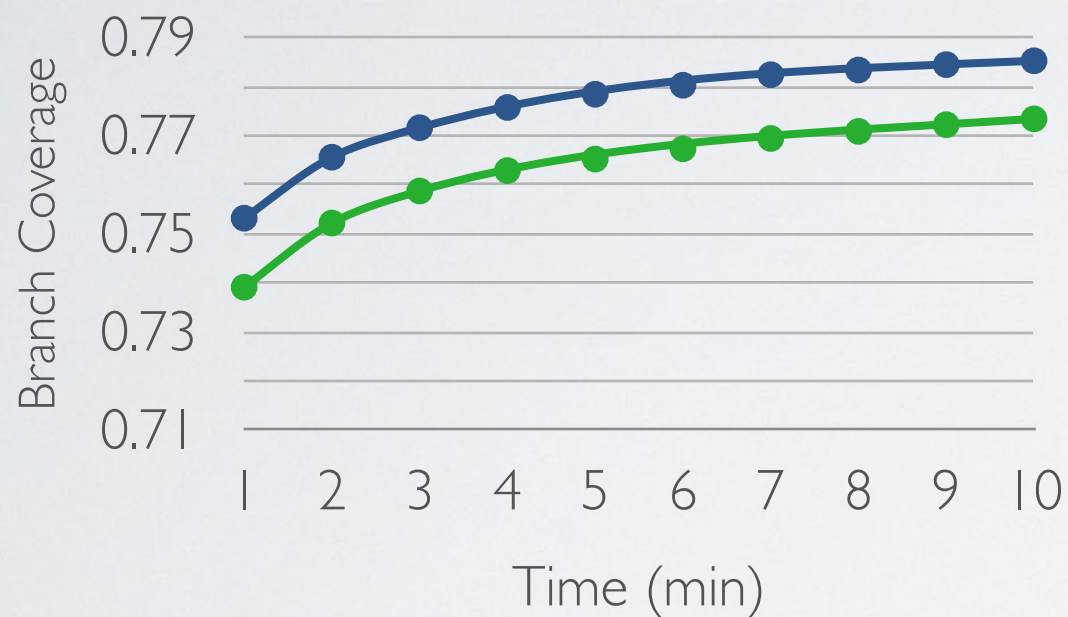


INCREASED SEARCH BUDGET

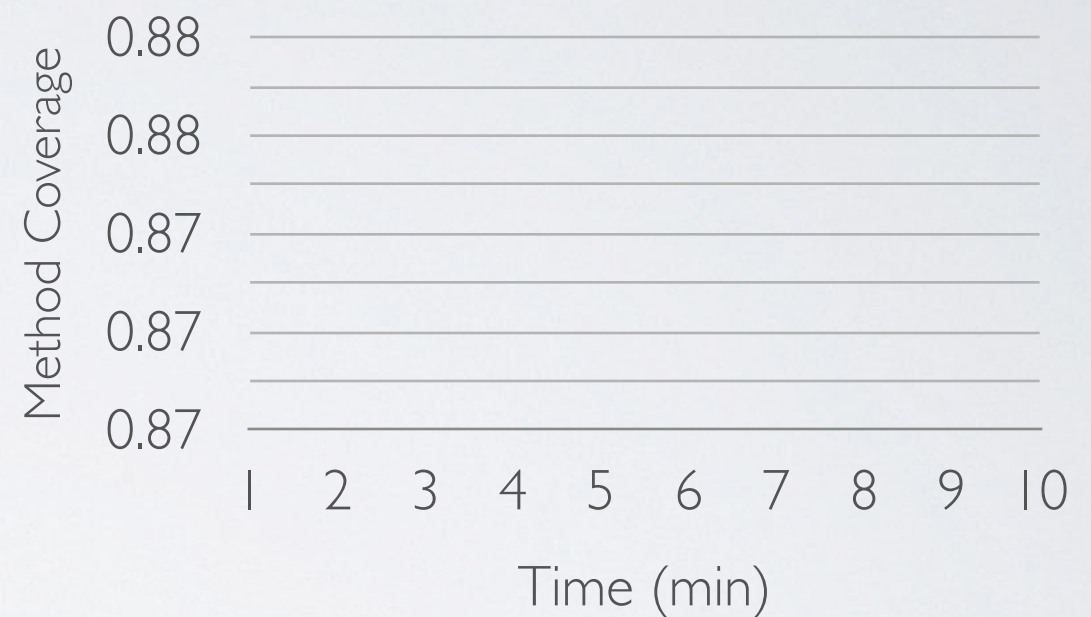
● Criterion

● All

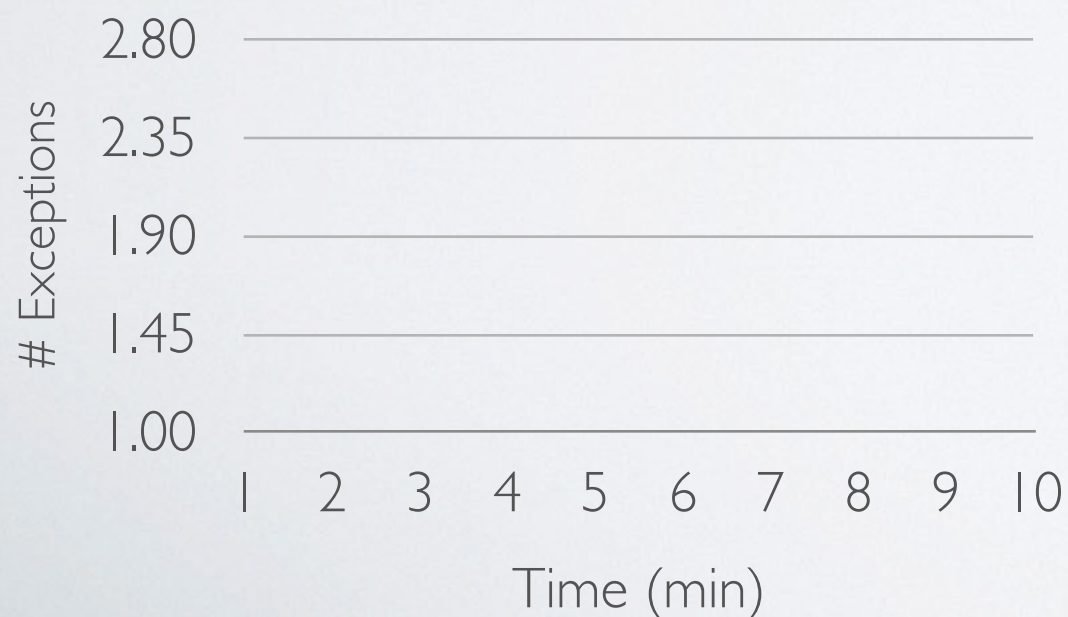
Branch



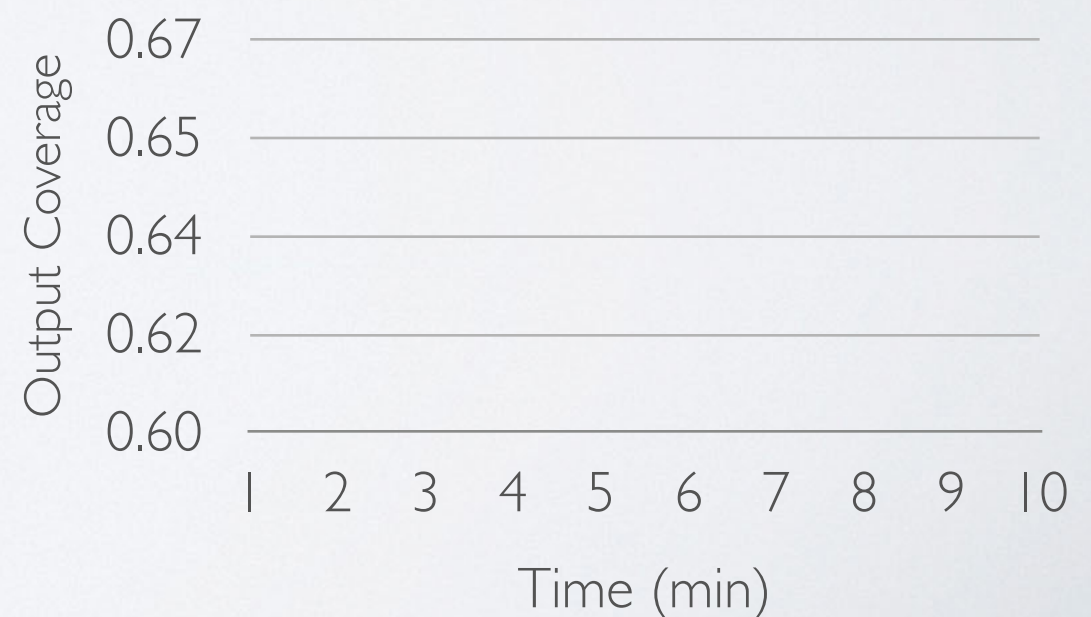
Method



Exception



Ouput

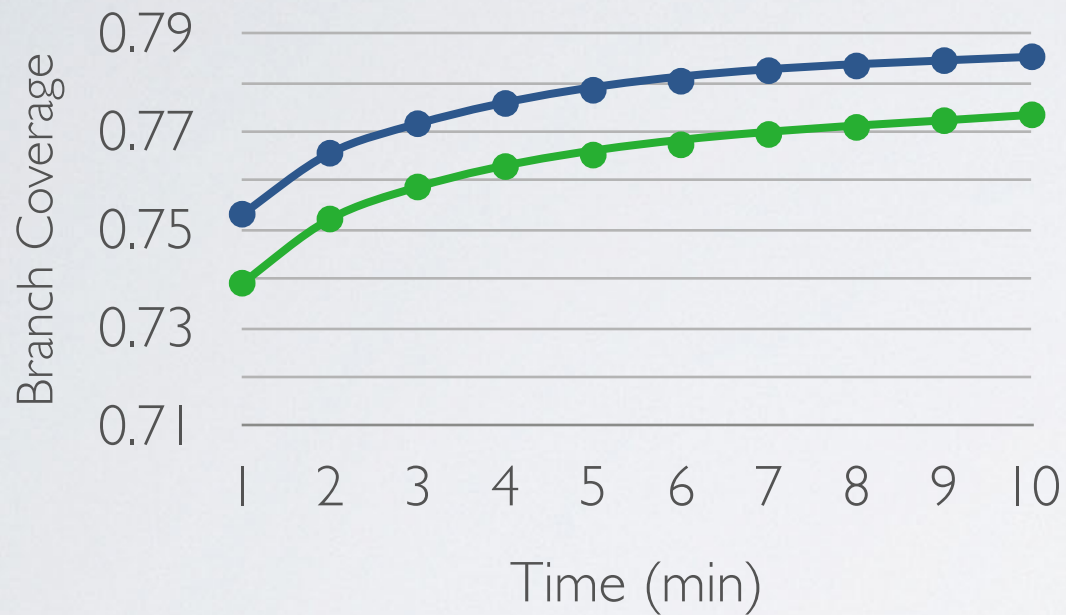


INCREASED SEARCH BUDGET

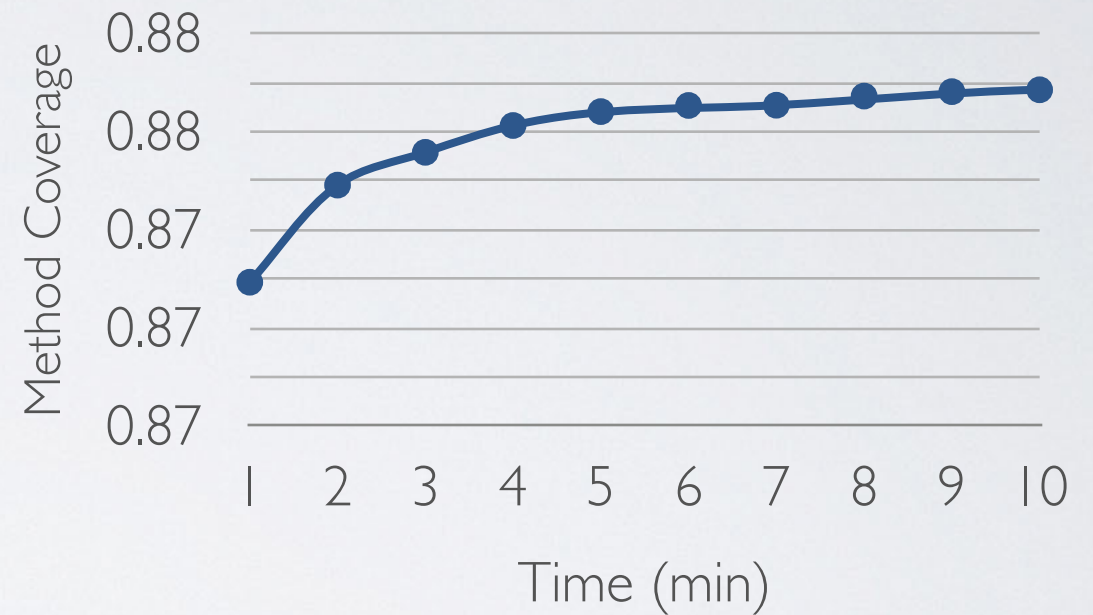
● Criterion

● All

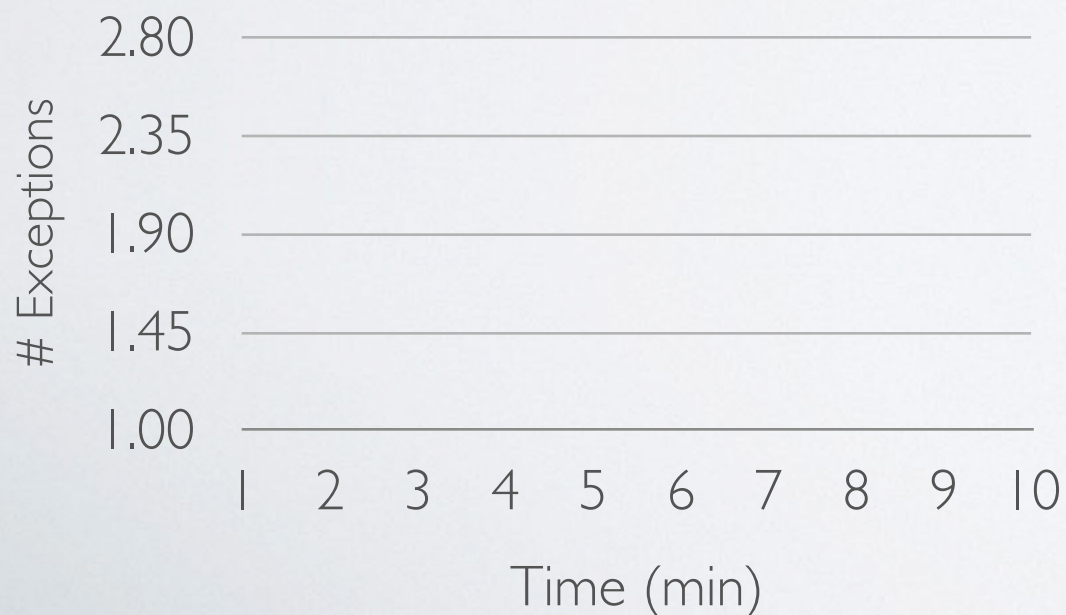
Branch



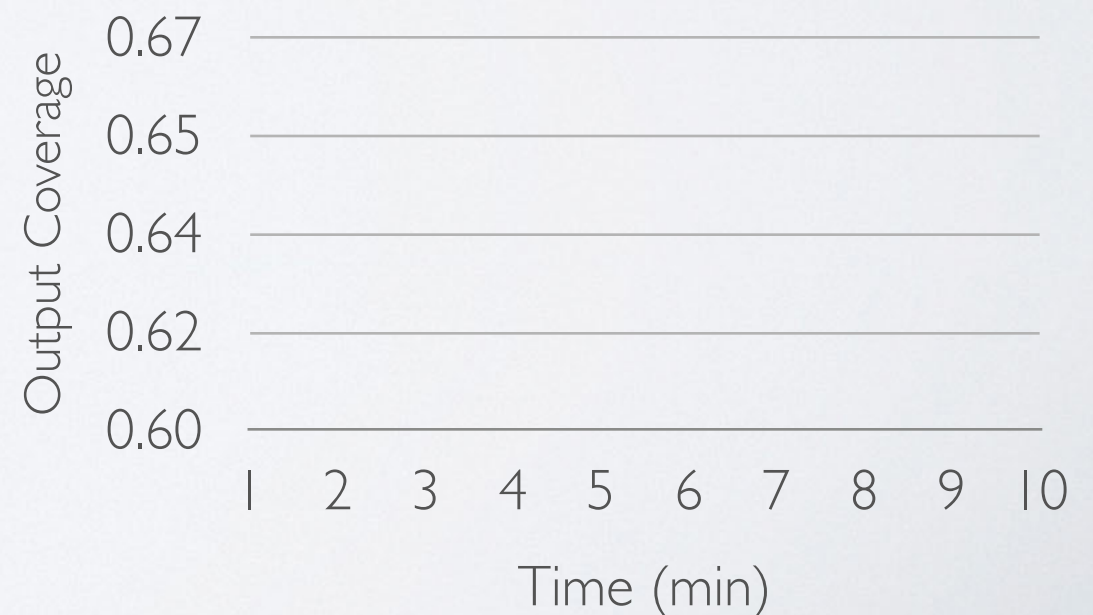
Method



Exception



Ouput

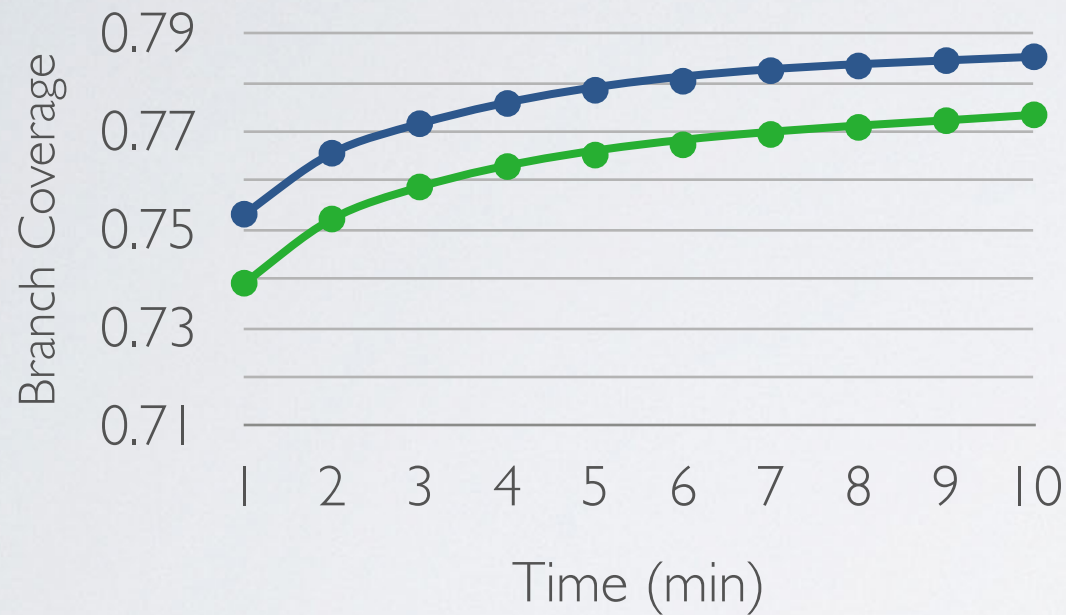


INCREASED SEARCH BUDGET

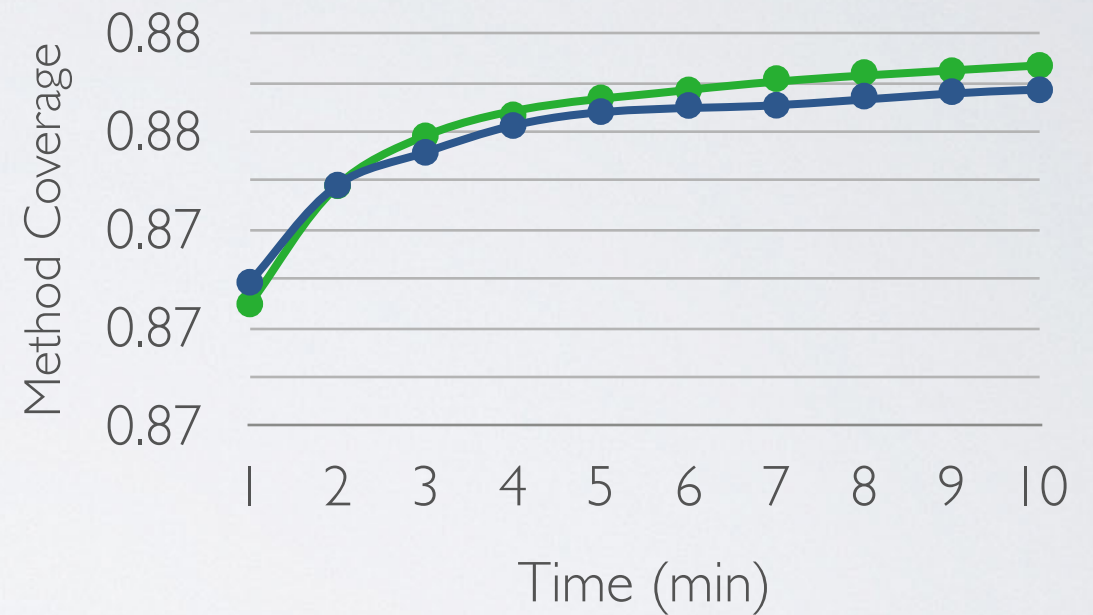
● Criterion

● All

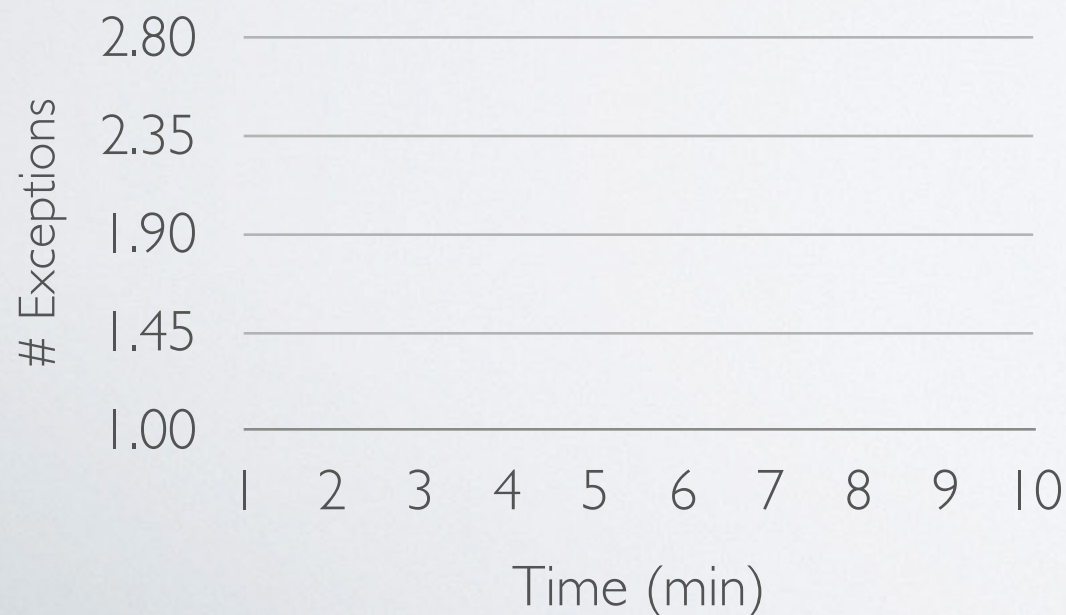
Branch



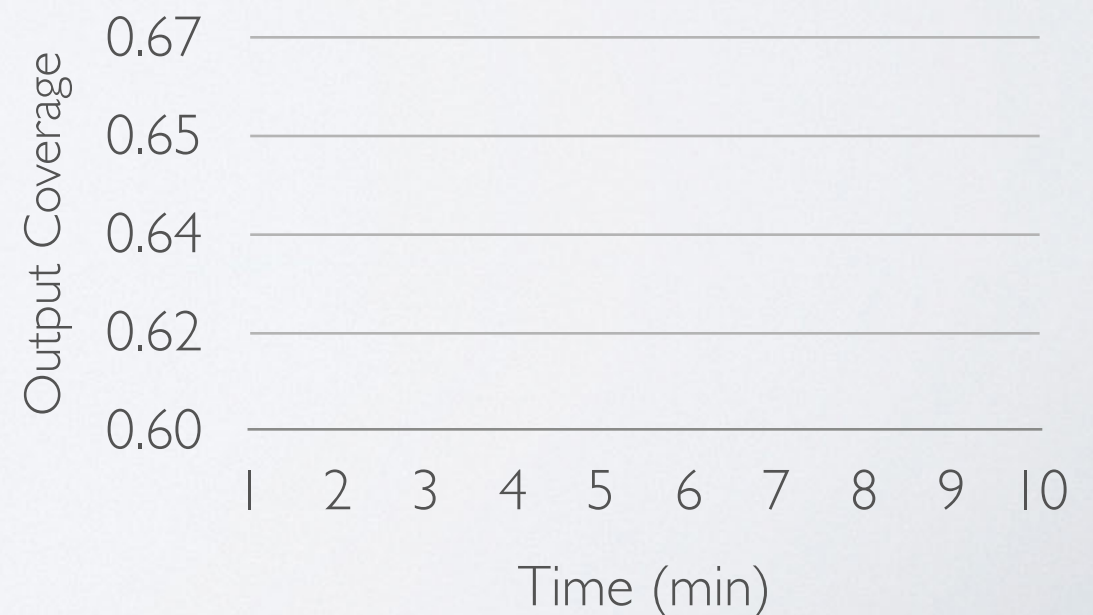
Method



Exception



Ouput

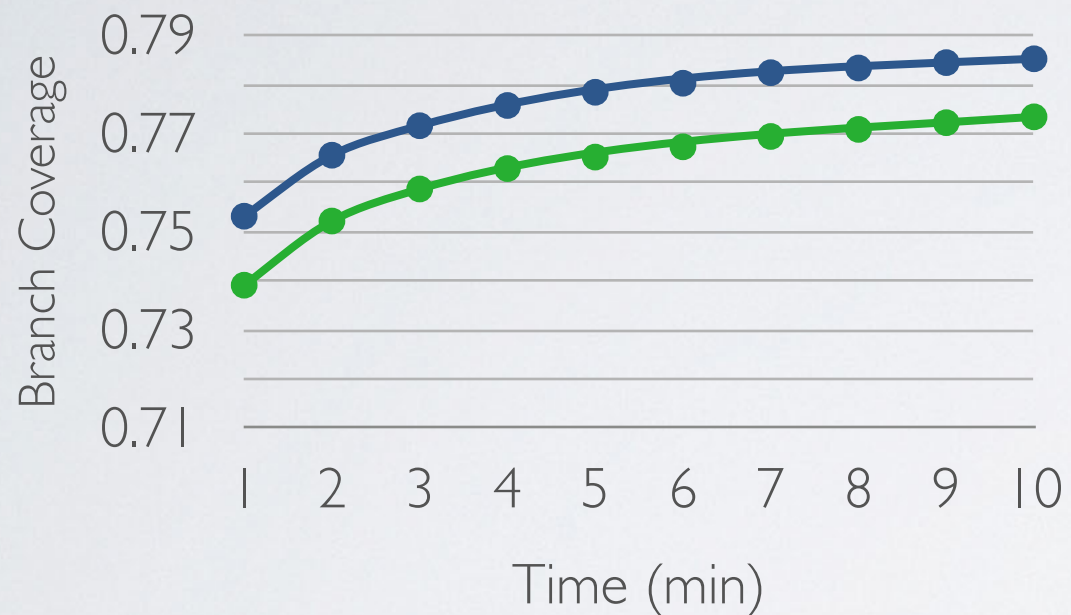


INCREASED SEARCH BUDGET

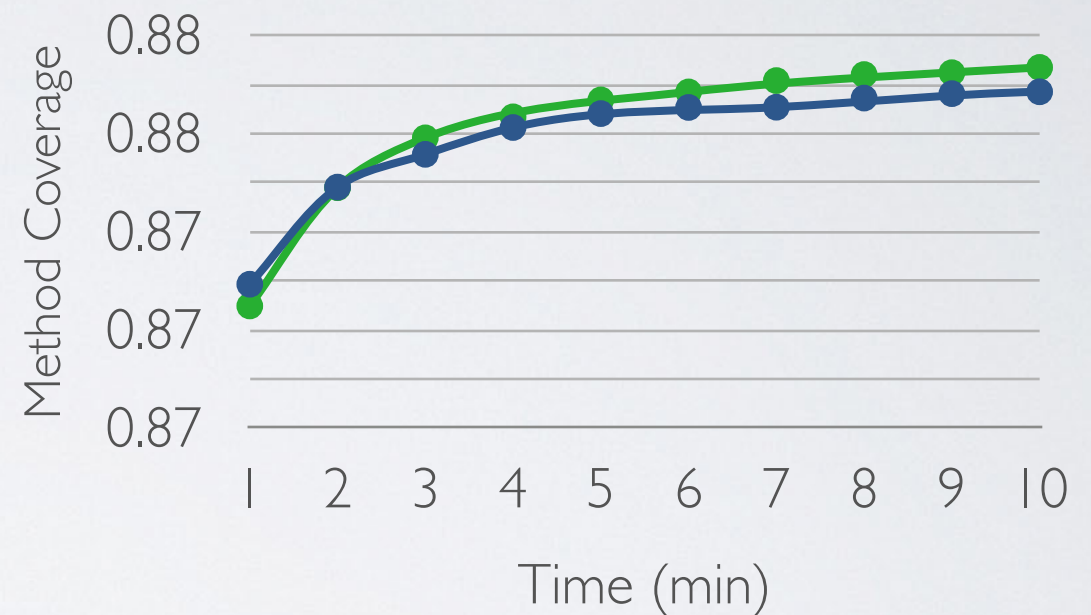
● Criterion

● All

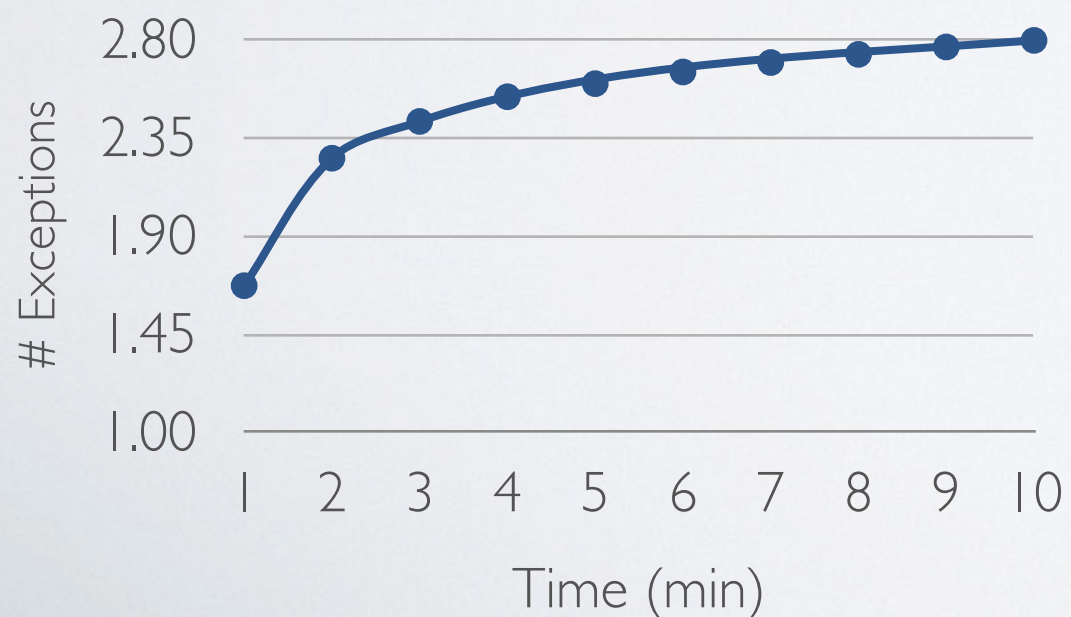
Branch



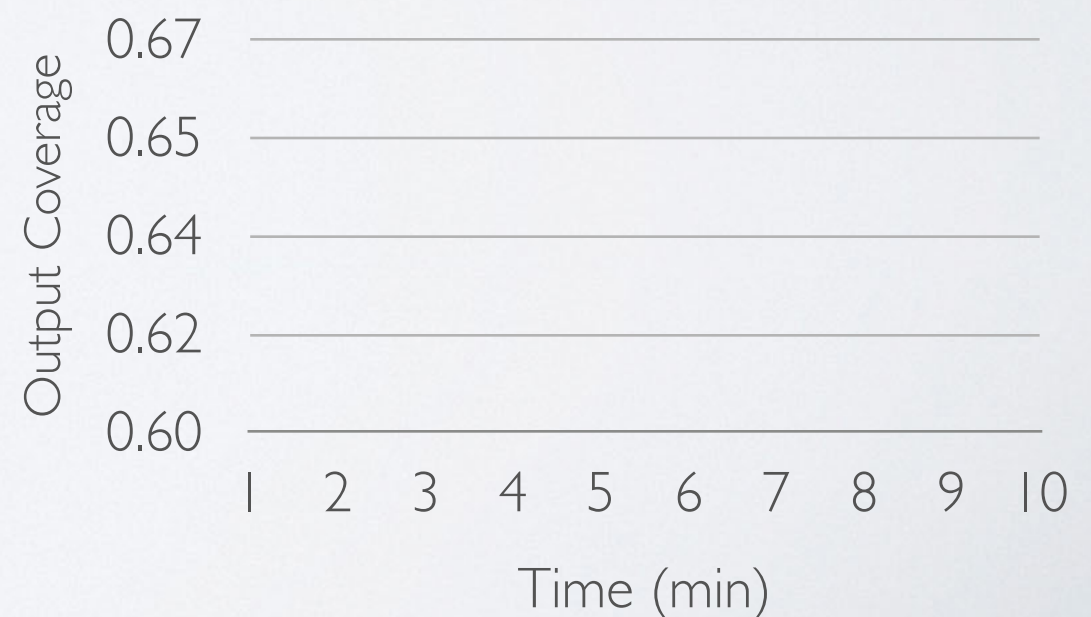
Method



Exception



Ouput

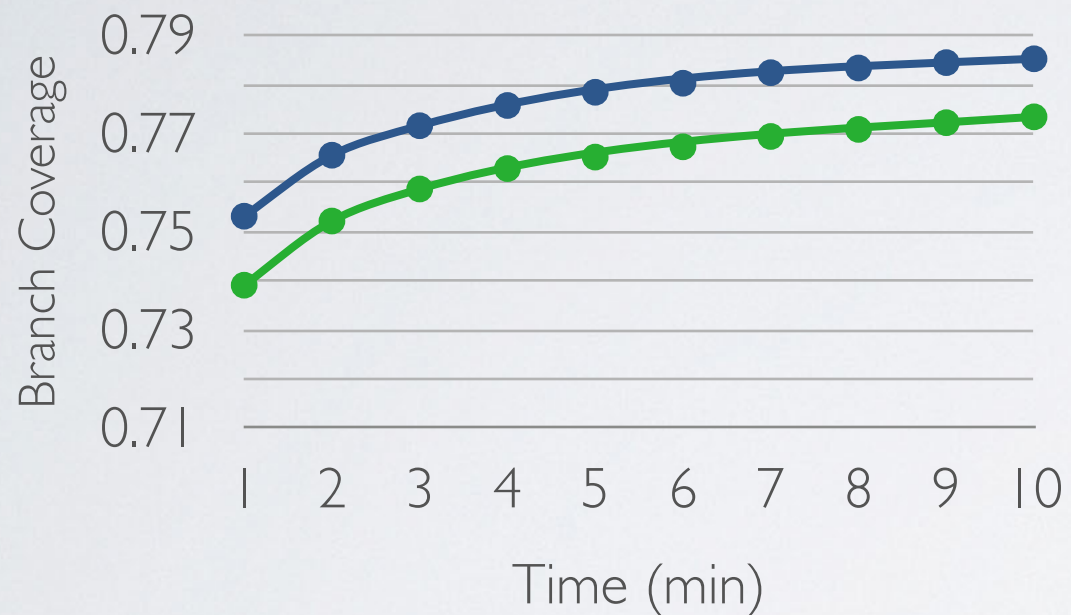


INCREASED SEARCH BUDGET

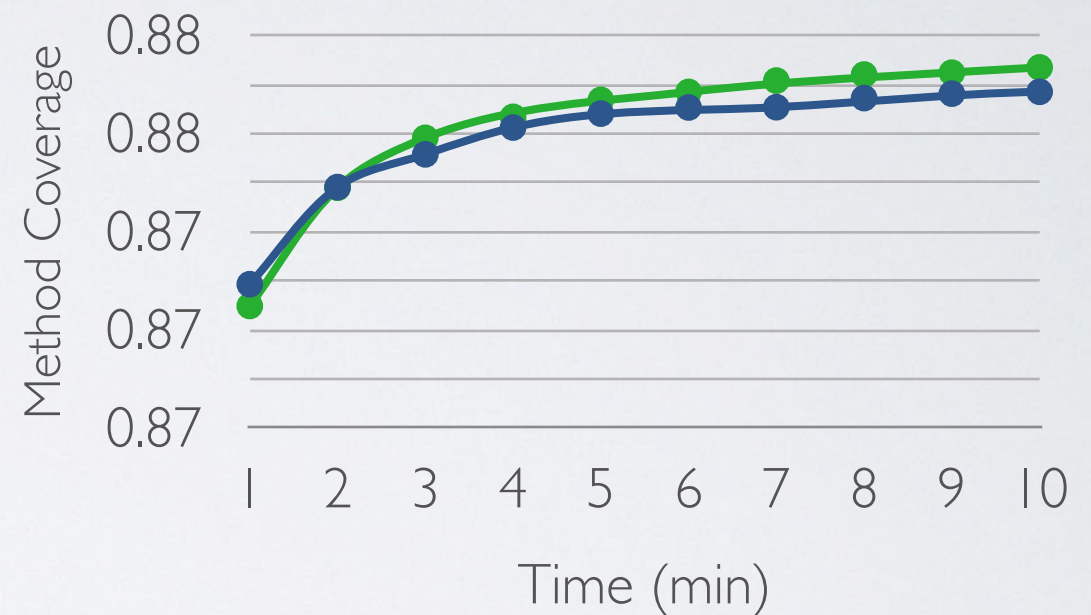
● Criterion

● All

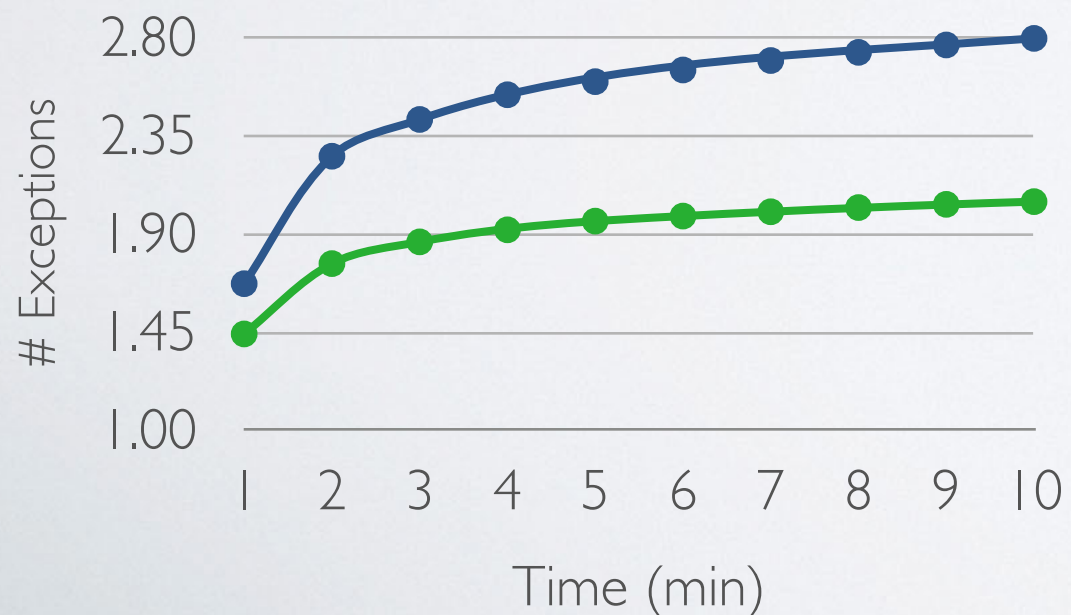
Branch



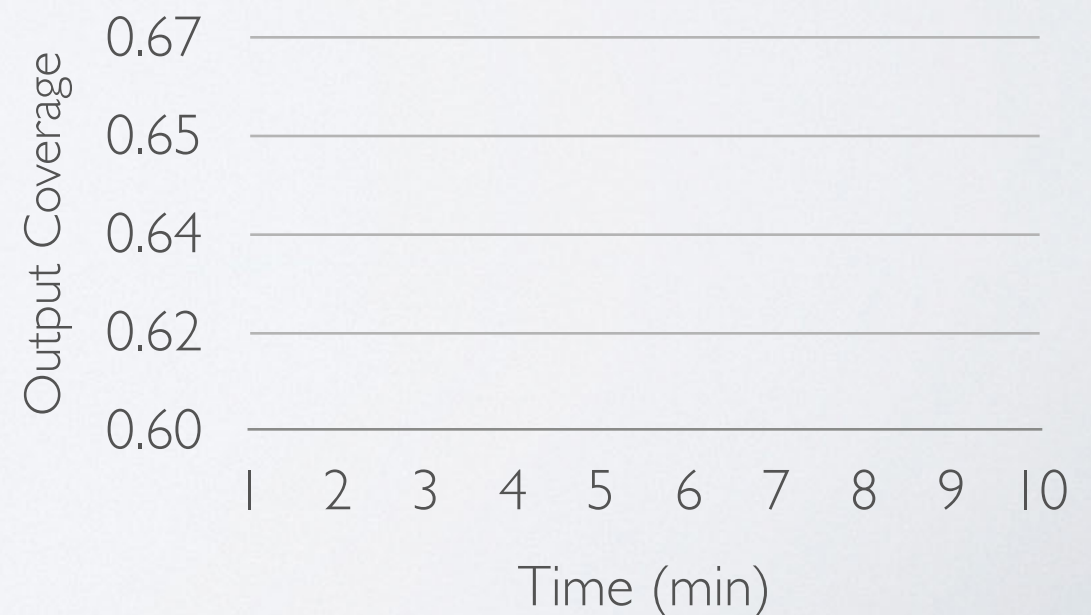
Method



Exception



Ouput

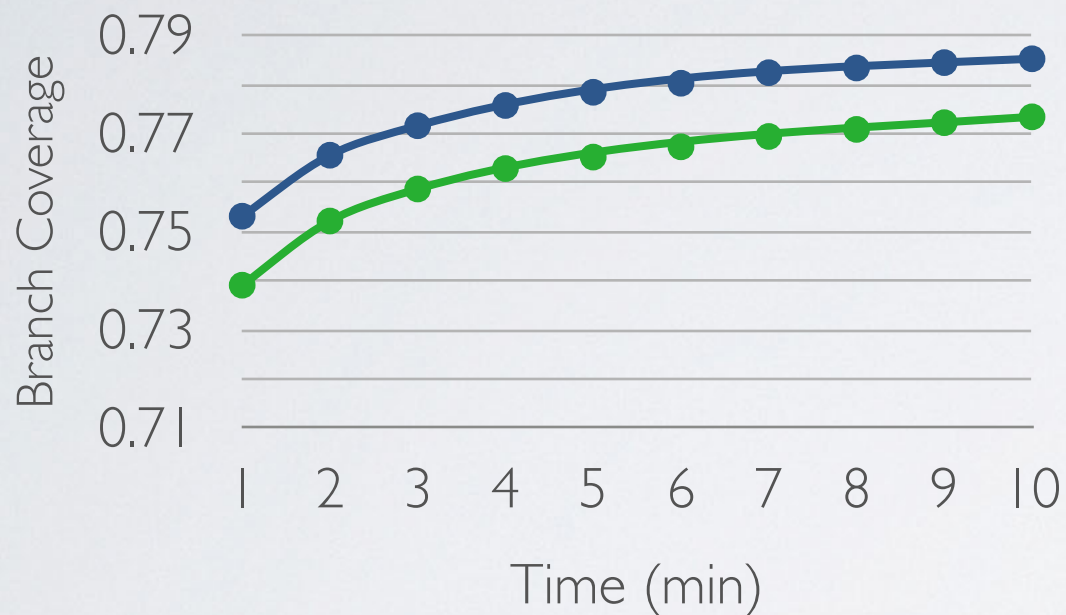


INCREASED SEARCH BUDGET

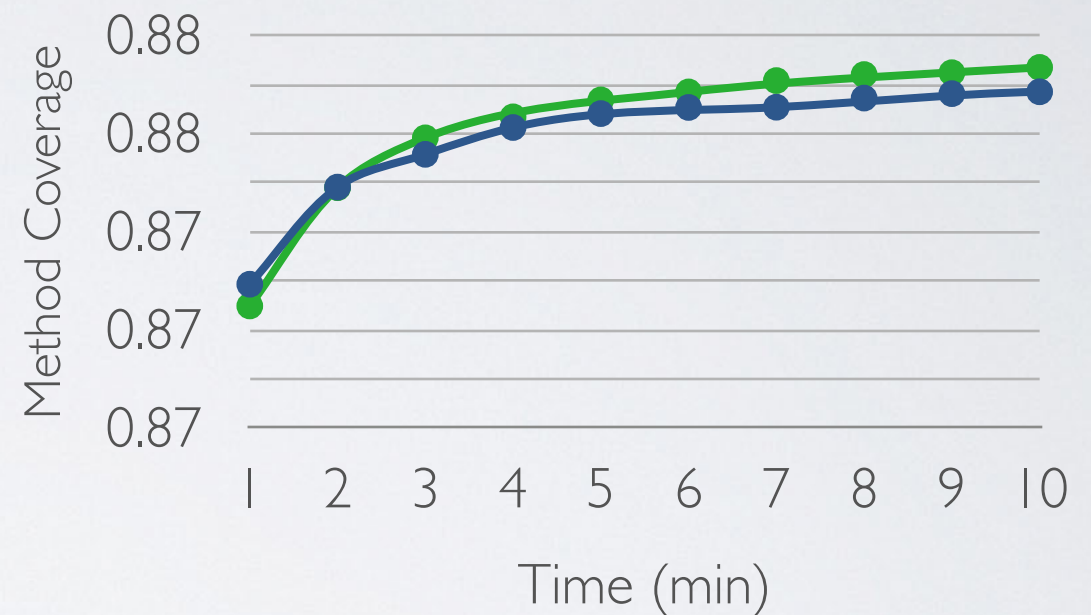
● Criterion

● All

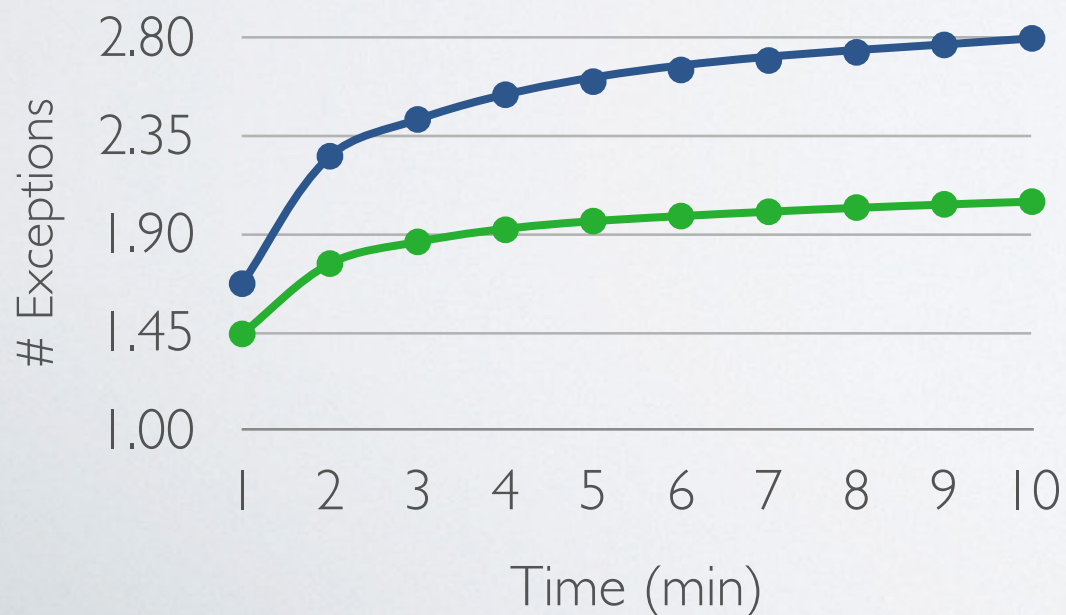
Branch



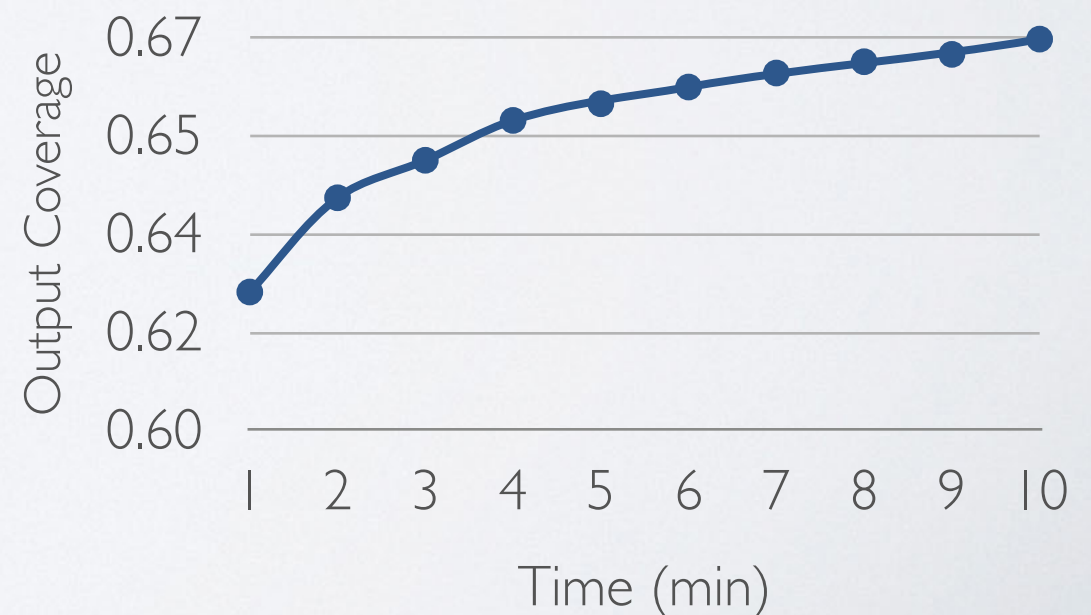
Method



Exception



Ouput

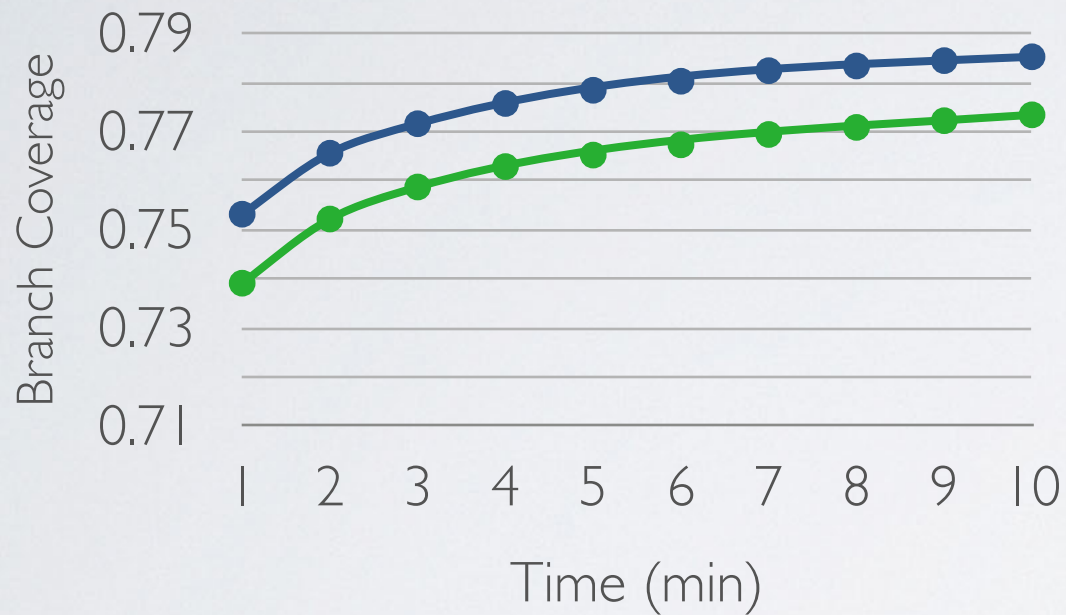


INCREASED SEARCH BUDGET

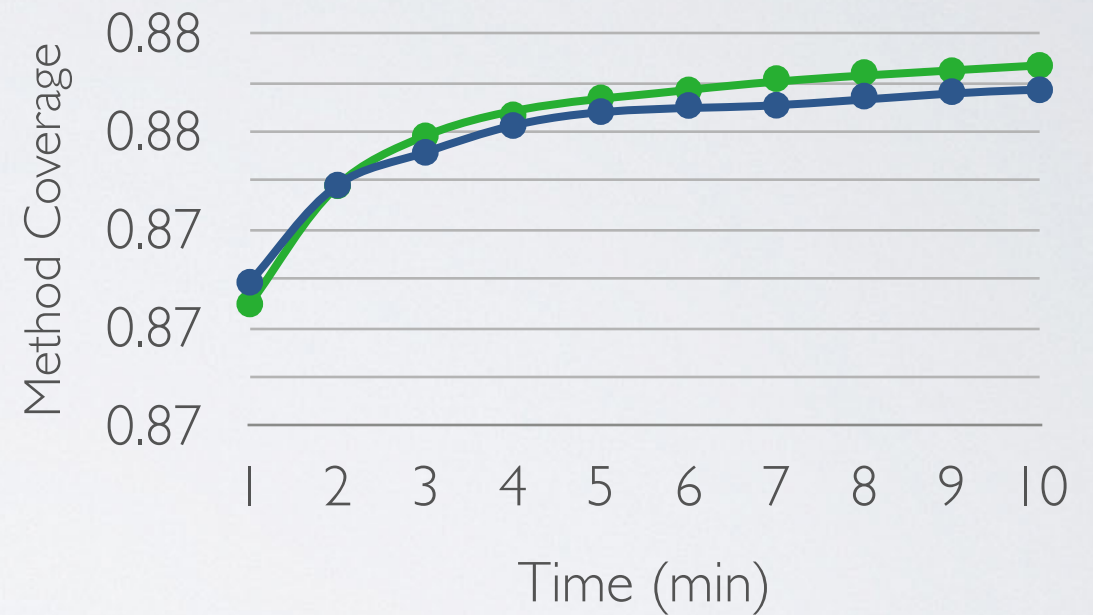
● Criterion

● All

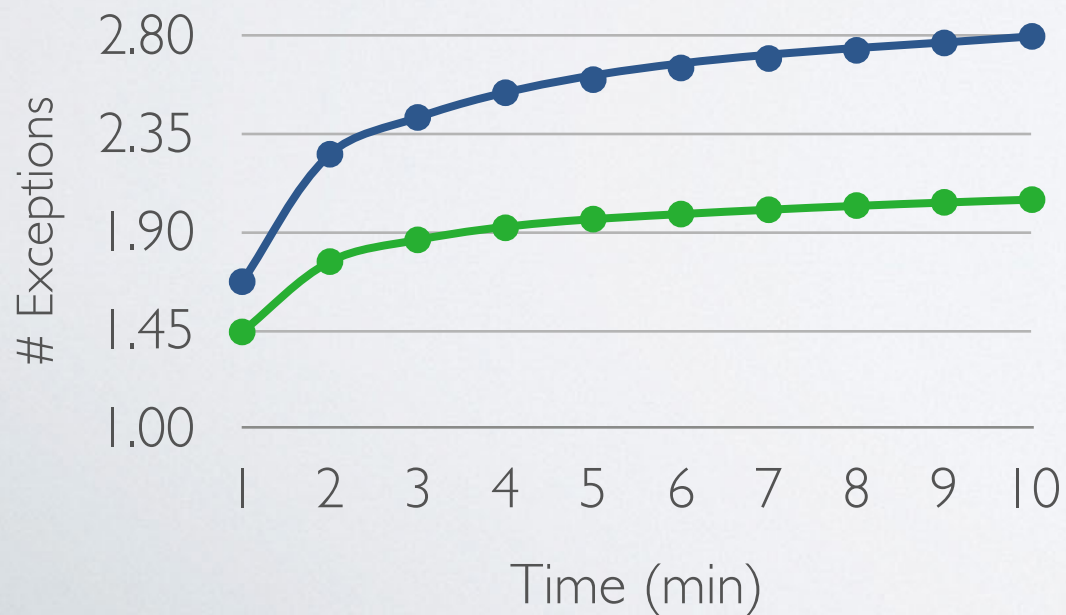
Branch



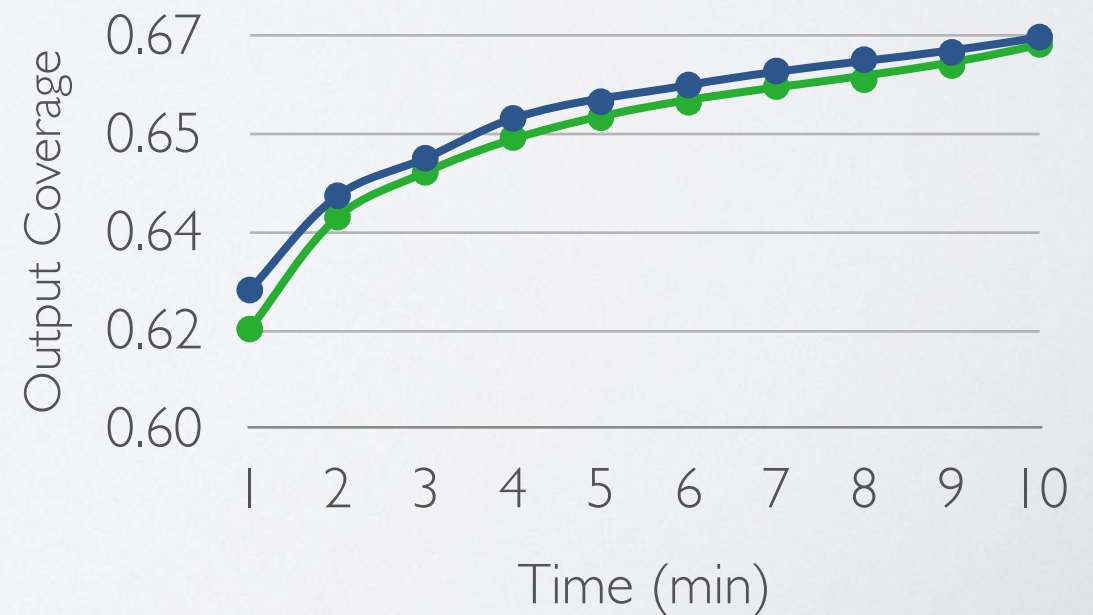
Method



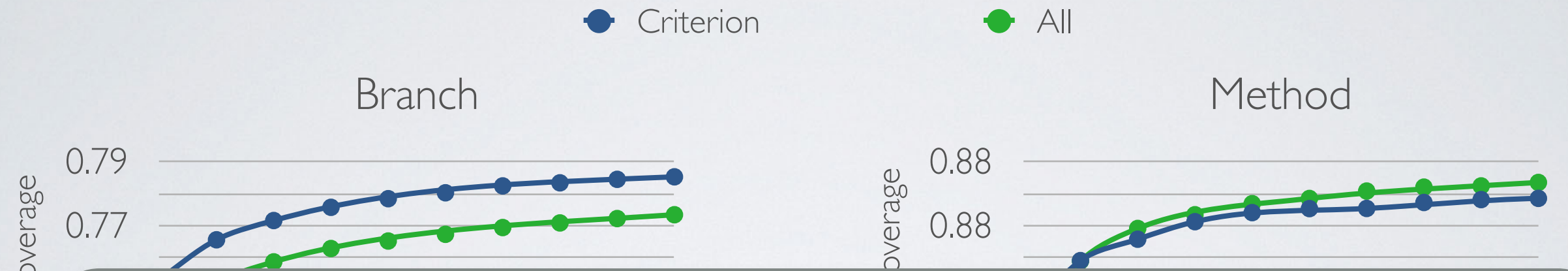
Exception



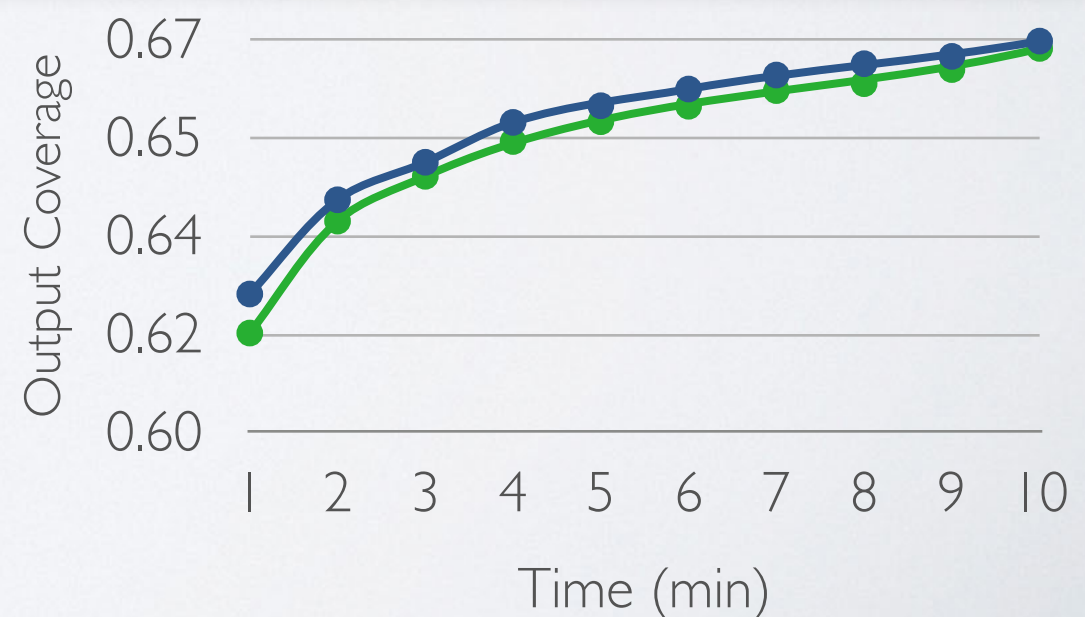
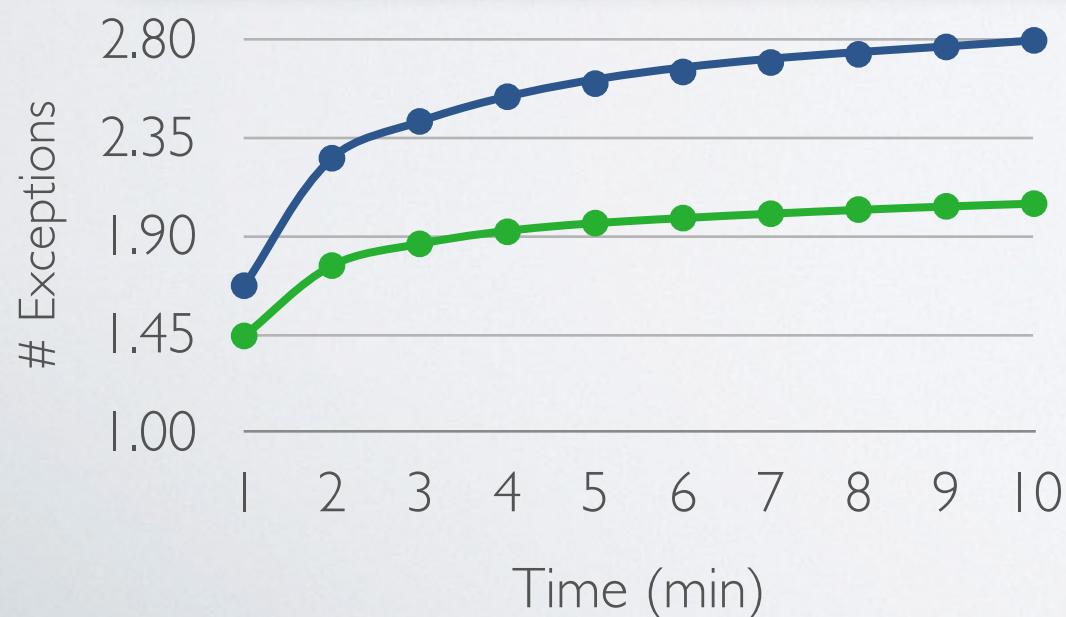
Ouput



INCREASED SEARCH BUDGET



RQ4. The influence of combining multiple criteria persists even with larger search budget.



DEMO

Eclipse plugin update site

<http://www.evosite.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