Comparing Non-adequate Test Suites using Coverage Criteria

Milos Gligoric[1], Alex Groce[2], Chaoqiang Zhang[2]
Rohan Sharma[1], Amin Alipour[2], Darko Marinov[1]

ISSTA 2013
Lugano, Switzerland
July 18, 2013

1 ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

2 Oregon State UNIVERSITY OSU

Motivation

- Publications are increasingly using coverage criteria to compare test suites and techniques

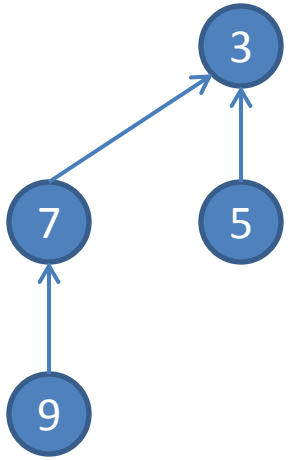
- What coverage criterion should researchers use to compare test suites?

Quiz 1

- Consider two test suites T_1 and T_2
  - T_1 50% statement coverage, 75% branch coverage
  - T_2 60% statement coverage, 55% branch coverage
- Which test suite is better?

Example: BinomialHeap

```
// public class BinomialHeap { ...
static class Node { int key; Node parent; }
Node nodes; int size;
void decreaseKey(int oldValue, int newValue) {
  Node tmp = nodes.findANodeWithKey(oldValue);
  if (tmp == null) return;
  tmp.key = newValue;
  Node tmpParent = tmp.parent;
  while ((tmpParent != null) && (tmp.key < tmpParent.key)) {
    int z = tmp.key;
    tmp.key = tmpParent.key;
    tmpParent.key = z;
    tmp = tmpParent;
    tmpParent = tmpParent.parent;
  }
}
```

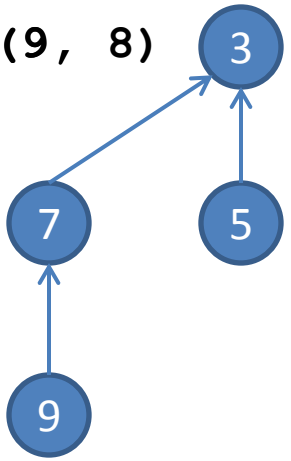


Acyclic Intra-Method Path (AIMP)

```
void decreaseKey(int oldValue, int newValue) {
  Coverage.beginMethod(0);
  Node tmp = nodes.findANodeWithKey(oldValue);
  if (tmp == null) { Coverage.cover(1); return;}
  Coverage.cover(2);
  tmp.key = newValue;
  Node tmpParent = tmp.parent;
  while ((tmpParent != null) && (tmp.key < tmpParent.key)) {
    Coverage.cover(3);
    int z = tmp.key;
    tmp.key = tmpParent.key;
    tmpParent.key = z;
    tmp = tmpParent;
    tmpParent = tmpParent.parent;
  } Coverage.cover(4);
}
```

decreaseKey(9, 8)

AIMP: 0, 2, 4



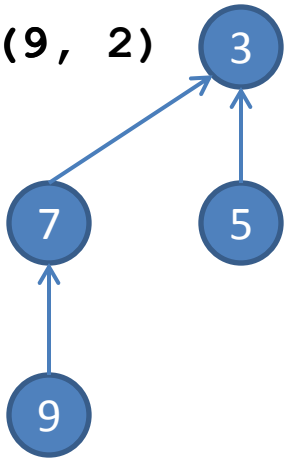
Acyclic Intra-Method Path (AIMP)

```
void decreaseKey(int oldValue, int newValue) {
  Coverage.beginMethod(0);
  Node tmp = nodes.findANodeWithKey(oldValue);
  if (tmp == null) { Coverage.cover(1); return;}
  Coverage.cover(2);
  tmp.key = newValue;
  Node tmpParent = tmp.parent;
  while ((tmpParent != null) && (tmp.key < tmpParent.key)) {
    Coverage.cover(3);
    int z = tmp.key;
    tmp.key = tmpParent.key;
    tmpParent.key = z;
    tmp = tmpParent;
    tmpParent = tmpParent.parent;
  } Coverage.cover(4);
}
```

decreaseKey(9, 2)

AIMP: 0, 2, 3

AIMP: 3, 4



Predicate-Complete Test Coverage (PCT)

```
void decreaseKey(int oldValue, int newValue) {
  Coverage.beginMethod(0);
  Node tmp = nodes.findANodeWithKey(oldValue);
  if (tmp == null) { Coverage.cover(1); return;}
  Coverage.cover(2);
  tmp.key = newValue;
  Node tmpParent = tmp.parent;
  while ((tmpParent != null) && (tmp.key < tmpParent.key)) {
    Coverage.cover(3);
    int z = tmp.key;
    tmp.key = tmpParent.key;
    tmpParent.key = z;
    tmp = tmpParent;
    tmpParent = tmpParent.parent;
  } Coverage.cover(4);
}
```

1. Extract predicates:

```
tmp == null
tmpParent != null
tmp.key < tmpParent.key
```

Predicate-Complete Test Coverage (PCT)

```
void decreaseKey(int oldValue, int newValue) {
  Coverage.beginMethod(0);
  Node tmp = nodes.findANodeWithKey(oldValue);
  if (tmp == null) { Coverage.cover(1, …); return;}
  Coverage.cover(2, …);
  tmp.key = newValue;
  Node tmpParent = tmp.parent;
  while ((tmpParent != null) && (tmp.key < tmpParent.key)) {
    Coverage.cover(3, p$49(tmp, tmpParent), …);
    int z = tmp.key;
    tmp.key = tmpParent.key;
    tmpParent.key = z;
    tmp = tmpParent;
    tmpParent = tmpParent.parent;
  } Coverage.cover(4, p$49(tmp, tmpParent));
}
```

2. Insert evaluation of predicates:

```
// tmp.key < tmpParent.key
boolean p$49(Node tmp , Node tmpParent) {
  if (tmpParent == null) return false;
  if (tmp == null) return false;
  return tmp.key < tmpParent.key;
}
```

Quiz 2

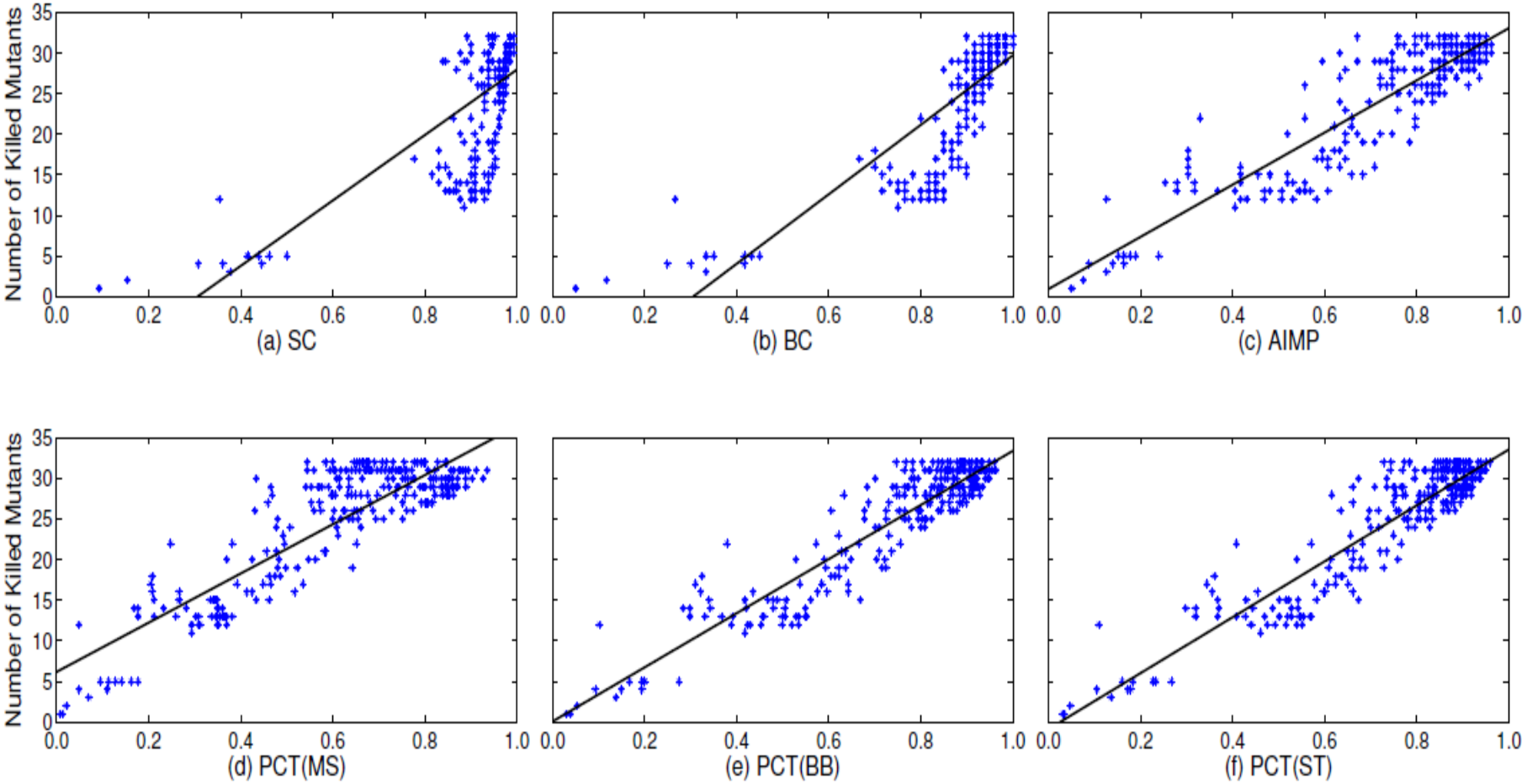
- Which coverage to use to compare test suites?
  - Statement (SC)
  - Branch (BC)
  - Intra-method path (AIMP)
  - Predicate-Complete Test Coverage (PCT)

Problem Discussion

- What does it mean that one coverage is good? Or that one coverage is better than another?
- We want coverage to be a predictor of finding bugs: test suites with higher coverage should find, on average, more (real) bugs
- Instead of prediction of real bugs, our experiments use prediction of mutation score

Towards the Answer

- Ability of coverage to predict mutation score



Statistical Evaluation

- Quantify the degree to which this holds:
  - If a suite A has higher coverage than a suite B, then the suite A has a higher mutation score
- Statistical tools
  - Kendall τ – measures how well coverage predicts the relative ordering of mutation score
  - R^2 - correlates coverage values with mutation score using a linear regression model

Steps

1. Select subjects
2. Obtain test pools for the subjects
3. Obtain mutants for the subjects
4. Create test suites from the test pools
5. Collect several metrics for the selected suites
6. Apply statistical tools to measure correlation

Step 1: Experimental Subjects

Subject	NBNC
AvlTree	344
BinomialHeap	264
BinTree	100
FibHeap	264
FibonacciHeap	397
HeapArray	98
IntAVLTreeMap	213
IntRedBlackTree	296
JFreeChart	72,490
JodaTime	27,472
LinkedList	245
NodeCachLList	234
SinglyLList	98
TreeMap	449
TreeSet	323

Subject	NBNC
Printtokens	479
Printtokens2	401
Replace	512
Schedule	292
Schedule2	297
SglibRbtree	476
Space	6,200
SQLite	81,934
Totinfo	340
Tcas	135
YAFFS2	11,760

Step 2: Tests

Subject	NBNC	tests
AvlTree	344	11,041
BinomialHeap	264	8,423
BinTree	100	13,825
FibHeap	264	12,842
FibonacciHeap	397	4,478
HeapArray	98	4,064
IntAVLTreeMap	213	17,072
IntRedBlackTree	296	20,419
JFreeChart	72,490	2,217
JodaTime	27,472	3,828
LinkedList	245	1,307
NodeCachLList	234	1,776
SinglyLList	98	1,762
TreeMap	449	14,076
TreeSet	323	17,400

Subject	NBNC	tests
Printtokens	479	4,130
Printtokens2	401	4,115
Replace	512	5,542
Schedule	292	2,650
Schedule2	297	2,710
SglibRbtree	476	5,000
Space	6,200	1,350
SQLite	81,934	117,240
Totinfo	340	917
Tcas	135	1,608
YAFFS2	11,760	5,000

- Automatically generated: Random, Shape Abstraction, Adaptation-based programming
- Manually written (bigger examples)

Step 3: Mutants

Subject	NBNC	tests	mutants
AvlTree	344	11,041	335
BinomialHeap	264	8,423	205
BinTree	100	13,825	55
FibHeap	264	12,842	186
FibonacciHeap	397	4,478	295
HeapArray	98	4,064	122
IntAVLTreeMap	213	17,072	199
IntRedBlackTree	296	20,419	279
JFreeChart	72,490	2,217	45,409
JodaTime	27,472	3,828	24,956
LinkedList	245	1,307	167
NodeCachLList	234	1,776	159
SinglyLList	98	1,762	57
TreeMap	449	14,076	463
TreeSet	323	17,400	360

Subject	NBNC	tests	mutants
Printtokens	479	4,130	536
Printtokens2	401	4,115	343
Replace	512	5,542	613
Schedule	292	2,650	140
Schedule2	297	2,710	300
SglibRbtree	476	5,000	443
Space	6,200	1,350	1,142
SQLite	81,934	117,240	52,367
Totinfo	340	917	511
Tcas	135	1,608	311
YAFFS2	11,760	5,000	10,674

- Javalanche used to mutate Java programs
- Proteum used to mutate C programs

Step 4: Creating Test Suites

- Coverage method
  - 300 test suites
  - Uniformly selecting values for PCT coverage
  - Then randomly choose tests to reach the coverage
- Size method (used in previous studies)
  - 100 random suites for each size between 1 and 50 (less varied coverage)

Step 5: Collecting Metrics

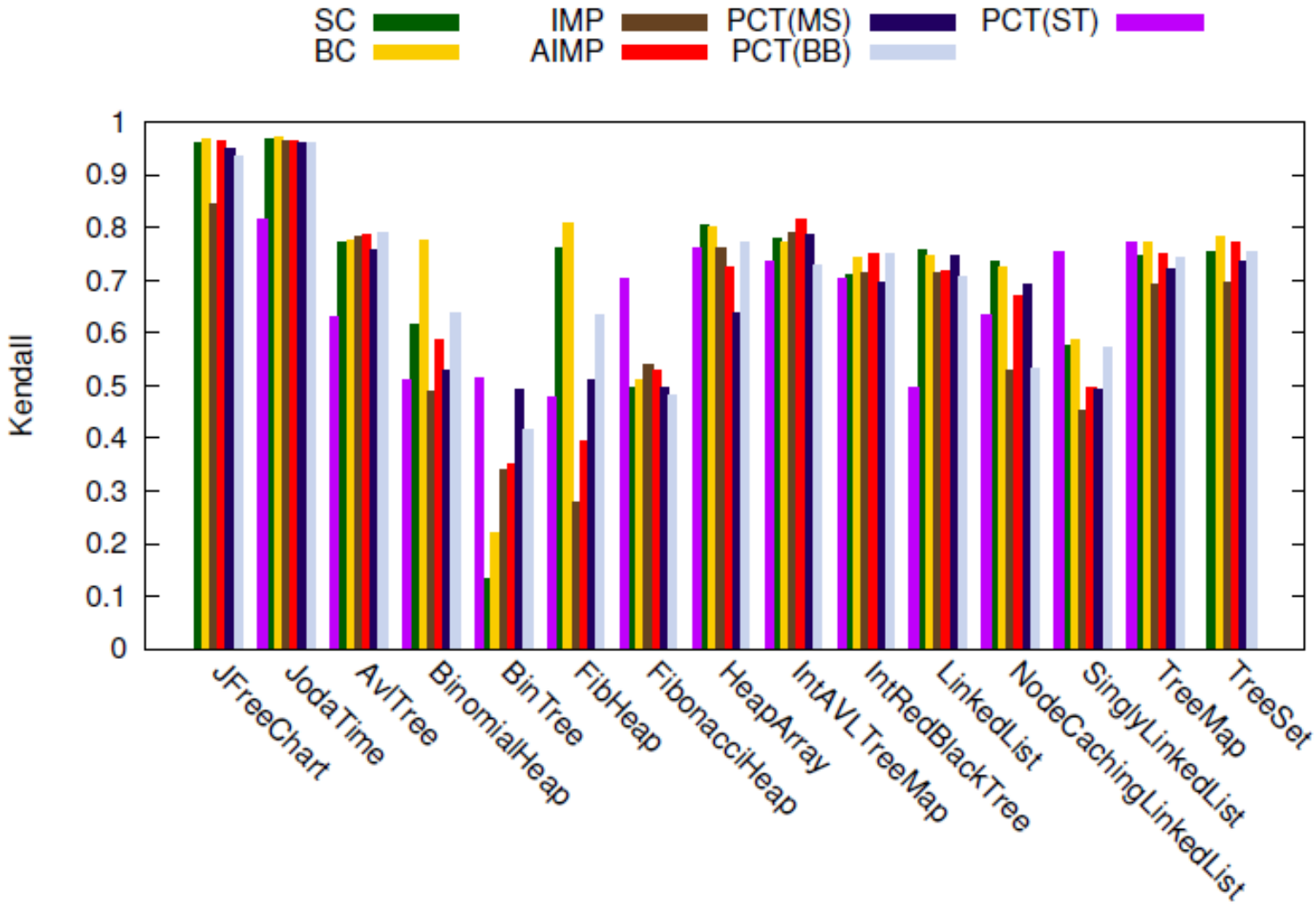
- Coverage criteria
  - SC, BC, AIMP, PCT (more in the paper)
- Mutation score
- Runtime overhead
- Example (for one of the subjects)

	SC	BC	AIMP	PCT	Mutation score	Overhead
Test suite 1	C_1				M_1	
Test suite 2	C_2				M_2	
…						
Test suite N	C_N				M_N	

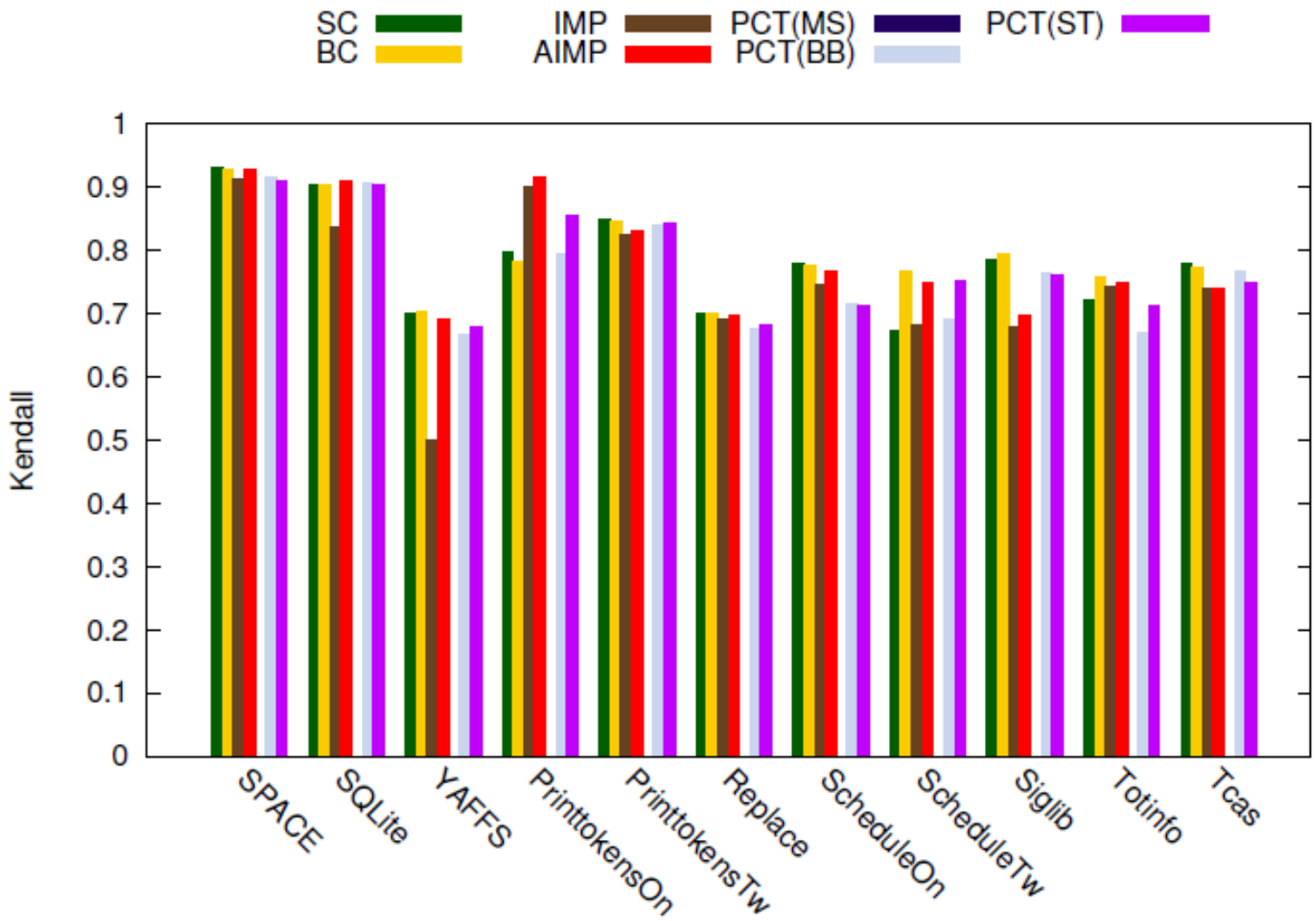
Step 6: Statistical analysis

- Kendall τ rank correlation coefficient
  - Consider two pairs (C_1, M_1) and (C_2, M_2)
  - Concordant if ordering of C_1 and C_2 matches M_1 and M_2, discordant otherwise
  - Ratio of the difference between the number of concordant and discordant pairs and the total number of pairs
- R^2 coefficient of determination
  - Linear regression model for each criterion
  - Given an indication of correlation
  - Intuitively, if one suite has X% higher coverage value than another suite, does it have a c∗X% higher mutation score?

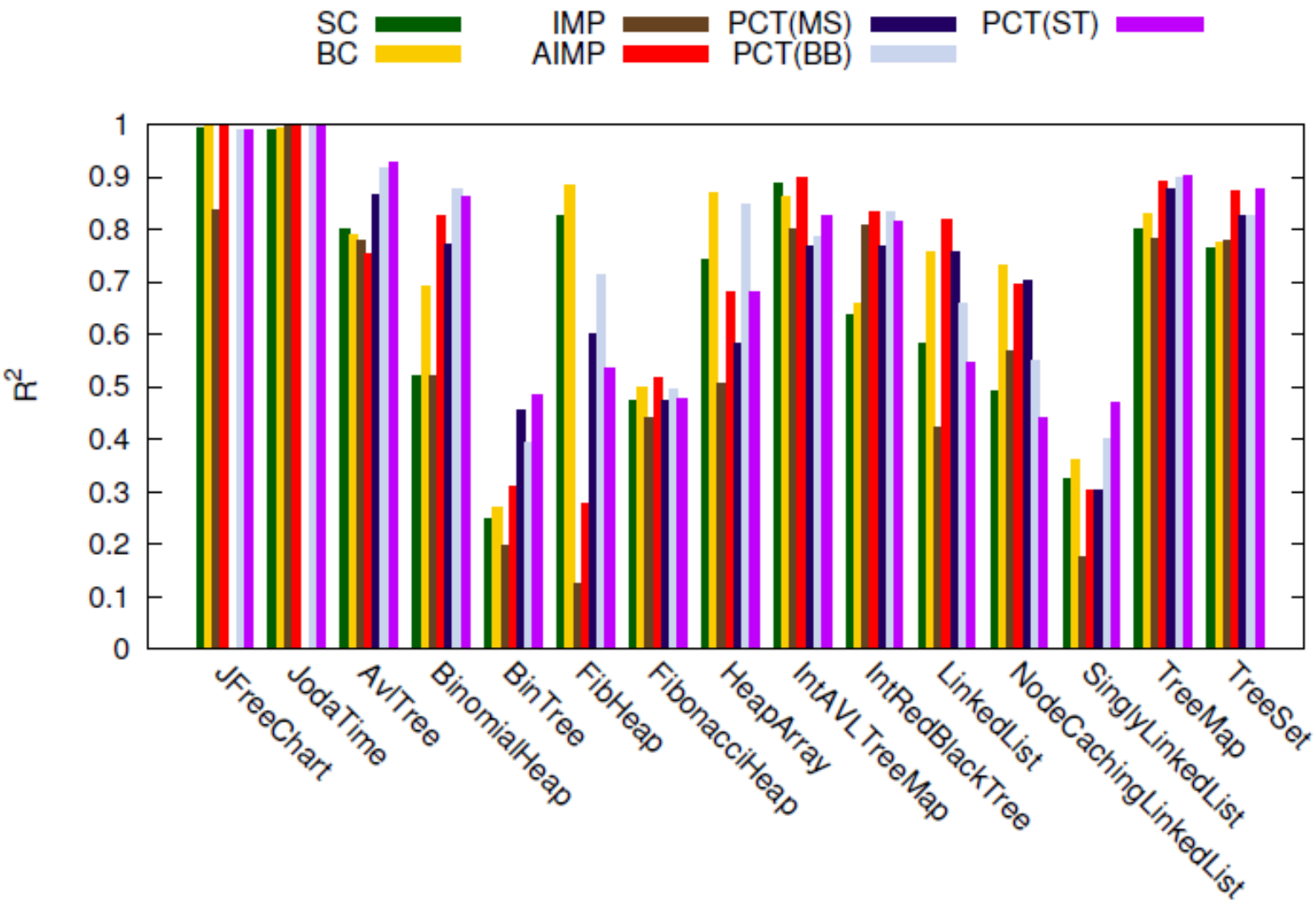
Results: Kendall τ for Java Subjects



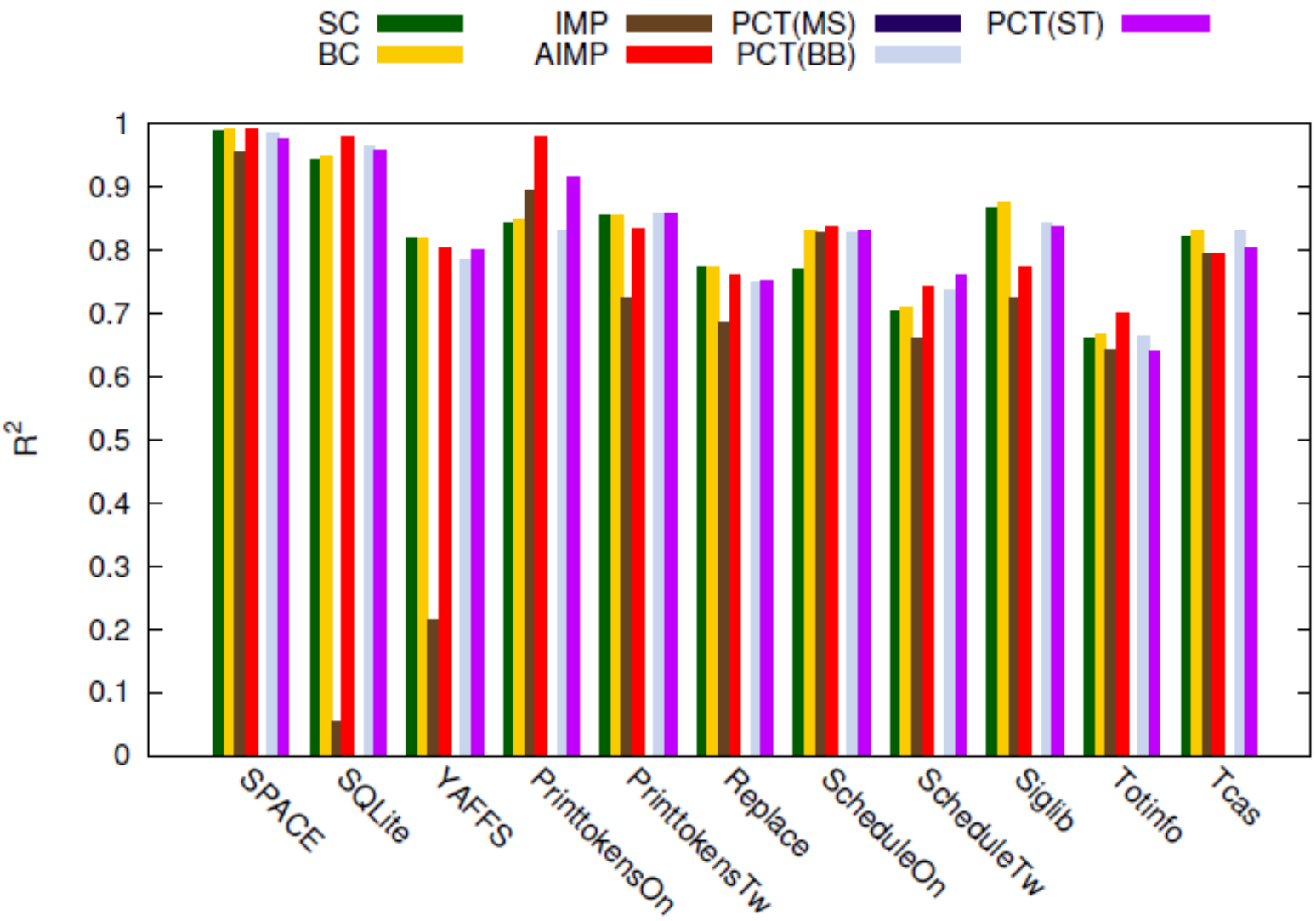
Kendall τ for C Subjects



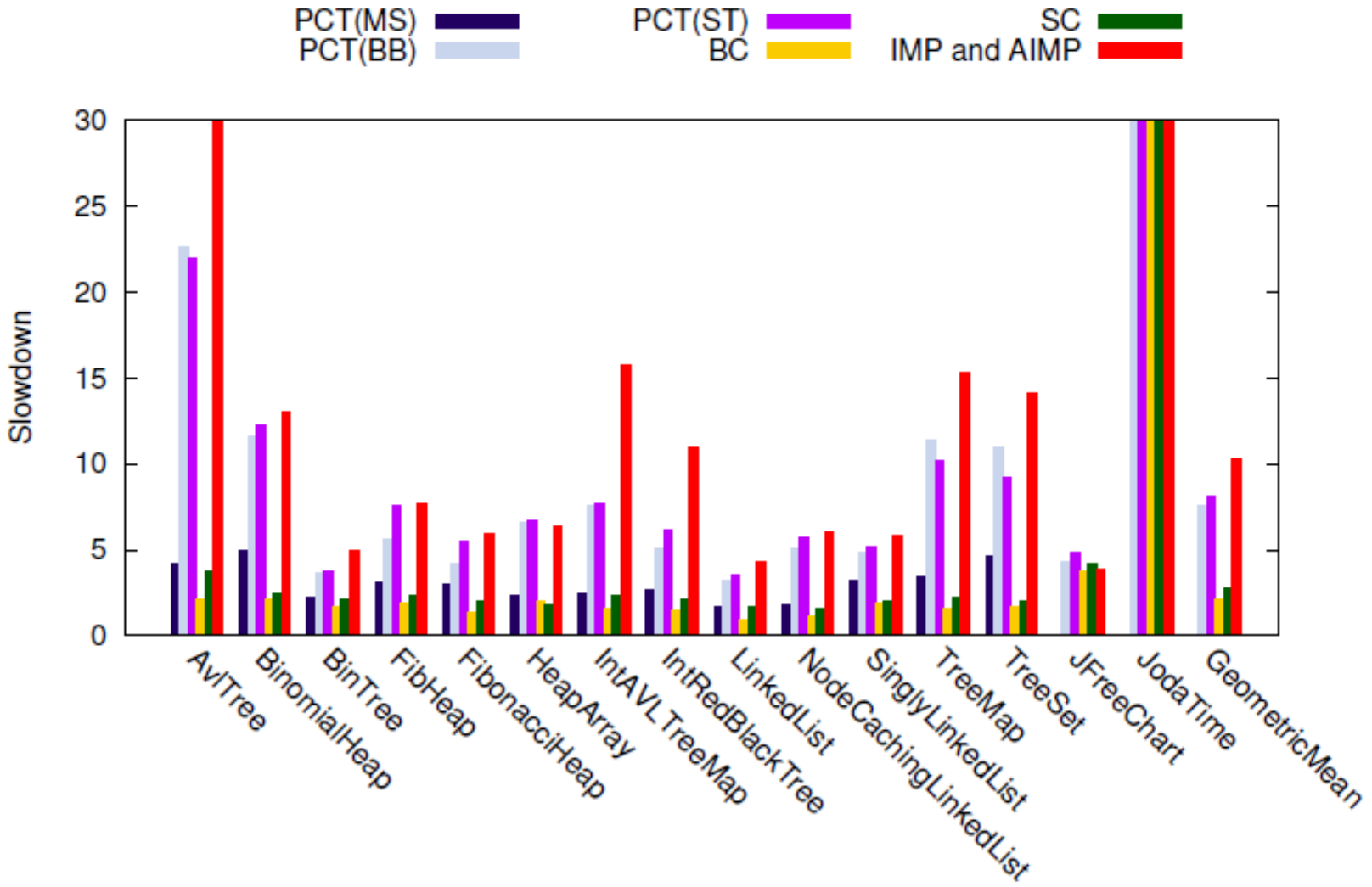
R^2 for Java Subjects



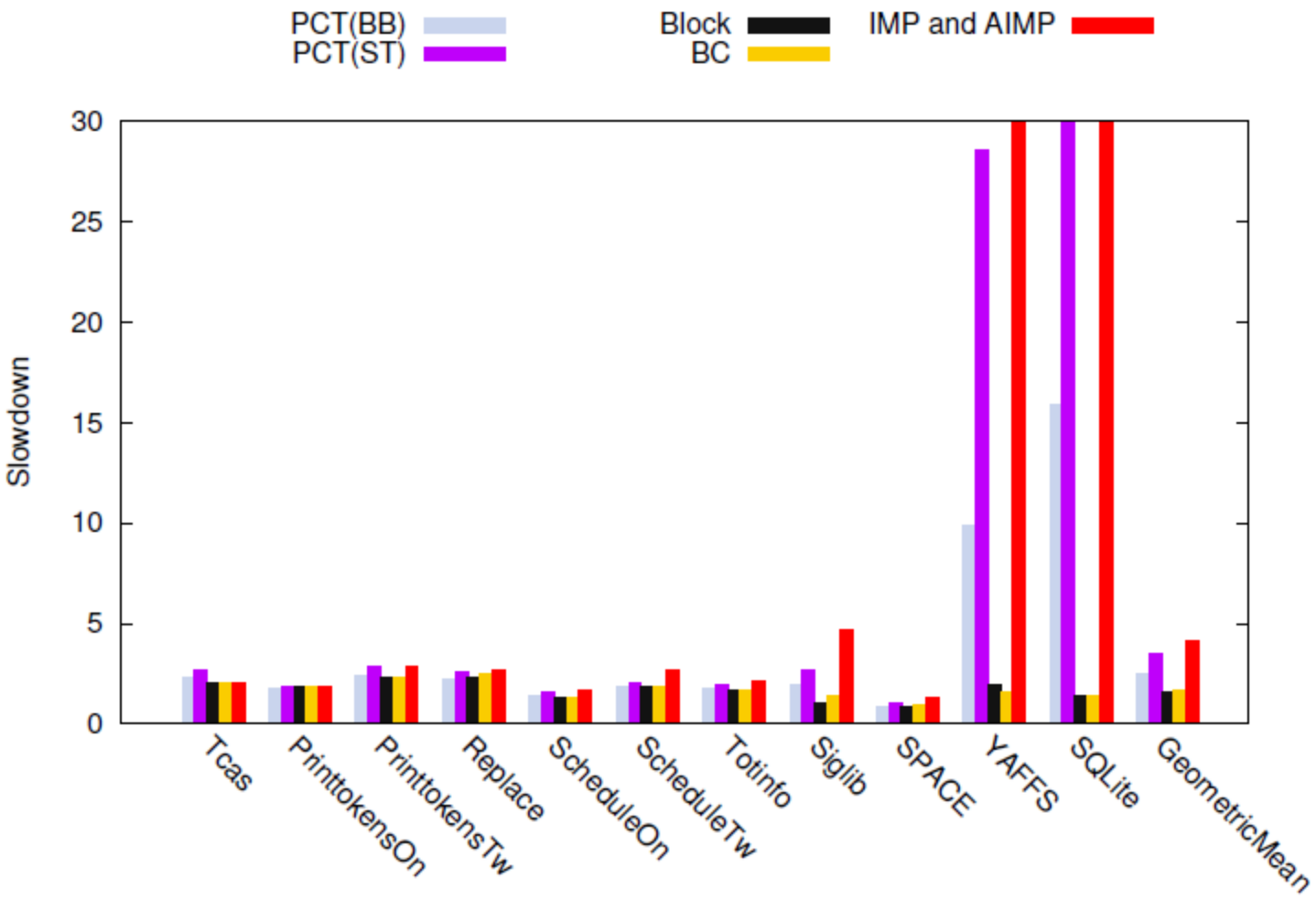
R^2 for C Subjects



Overhead for Java Subjects



Overhead for C Subjects



Conclusions

- Publications are increasingly using coverage criteria to compare test suites and techniques
- Our study compared coverages
- Take-away messages
  - Due to high effectiveness and low overhead, researchers should use branch coverage to compare suites whenever possible
  - Intra-procedural acyclic path coverage performed best of all non-branch coverage criteria

http://mir.cs.illinois.edu/coco/