Lightweight Fault Localization Using Multiple Coverage Types

Raul Santelices

James A. Jones* Yanbing Yu

Mary Jean Harrold

Work supported by NSF CCF-0429117, CCF-0541049, and CCF-0725202, and by TCS Ltd.

Presenter supported by SIGSOFT CAPS

* UCIrvine
University of California, Irvine

Aristotle Research Group

Home of the 1996 Olympic Village
Georgia Tech
College of Computing

Previous Research and Limitations

Techniques based on coverage

- Tarantula [Jones, Harrold et al. ICSE 2002, ASE 2005, ISSTA 2007, ICSE 2008]
- Nearest Neighbors [Renieris, Reiss ASE 2003]
- SBI [Liblit, Naik, Zheng, Aiken, Jordan PLDI 2005]
- SOBER [Liu, Yan, Fei, Han, Midkiff ESEC/FSE 2005]
- Other derivative work [Abreu, Zoeteweij, van Gemund TAIC-PART 2007] [Masri AUB-Tech.Rep. 2007]

Previous Research and Limitations

Techniques based on coverage

- Tarantula [Jones, Harrold et al. ICSE 2002, ASE 2005, ISSTA 2007, ICSE 2008]
- Nearest Neighbors [Renieris, Reiss ASE 2003]
- SBI [Liblit, Naik, Zheng, Aiken, Jordan PLDI 2005]
- S
- C TAIC-
  F

But…

- Operate on individual coverage types (e.g., statements, branches)
- No thorough comparison of the effectiveness of these coverage types

In This Work

- Method and study comparing coverage types
  - Statements
  - Branches
  - Data dependencies (du-pairs)
- Method and study combining coverage types
- Method and study reducing overhead
  - Overhead of statements and branches is 10–20%
  - Overhead of du-pairs is 60–120%, but du-pair coverage can be inferred (approximately) from branch coverage

Outline

- Coverage-based fault localization
- Comparing coverage types
- Combining coverage types for effectiveness
- Inferring du-pair coverage (approximately) to reduce overhead
- Conclusion and future work

Coverage-based Fault Localization

```
  mid() {
    int x,y,z,m;
read(“Enter 3 integers:’
m = z;
if (y<z)
   if (x<y)
       m = y;
   else if (x<z)
       m = y;
else
   if (x>y)
       m = z;
    else if (x>z)
       m = x;
print(“Middle number is
   }
```

Test suite

Runtime information

- entities executed (covered)
- passing/failing tests

Analysis

- computes *suspiciousness* of each entity

Intuition: entities executed mostly by failing test cases are more suspicious than entities executed mostly by passing tests

Coverage-based Fault Localization

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
`mid() {` `int x,y,z,m;`	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•
`2:m = z;`	•	•	•	•	•	•	•	•	•	•
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•
`4:   if (x<y)`	•	•			•	•		•		•
`5:      m = y;`		•								
`6:   else if (x<z)`	•				•	•		•		•
`7:      m = y;`	•				•			•		•
`8:else`			•	•			•		•	
`9:   if (x>y)`			•	•			•		•	
`10:      m = z;`			•				•		•	
`11:   else if (x>z)`				•						
`12:      m = x;`										
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F

Technique: Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8 2,1,3	t9 5,4,2	t10 5,2,6	suspiciousness
`int x,y,z,m;`											
`1:read(“Enter 3 integers:”,x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50
`2:m = z;`	•	•	•	•	•	•	•	•		•	
`3:if (y<z)`	•	•	•	•	•	•			•	•	
`4:   if (x<y)`	•									•	
`5:      m = y;`											
`6:   else if (x<z)`	•									•	
`7:      m = y;`	•									•	
`8:else`											
`9:   if (x>y)`			•	•			•		•		
`10:      m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:      m = x;`											
`13:print(“Middle number is:”, m);`	•	•	•	•	•	•	•	•	•	•	
`}`											
Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

$$susp(1) = \frac{\frac{4}{4}}{\frac{6}{6} + \frac{4}{4}} = 0.50$$

Technique: Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	suspiciousness
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6	
`int x,y,z,m;`											
`1:read("Enter 3 integers:",x,y,z);`	•									•	0.50
`2:m = z;`	•									•	
`3:if (y<z)`	•									•	
`4:   if (x<y)`	•									•	
`5:      m = y;`											
`6:   else if (x<z)`	•				•	•				•	
`7:      m = y;`	•				•			•		•	0.60
`8:else`			•	•			•		•		
`9:   if (x>y)`			•	•			•		•		
`10:      m = z;`			•				•		•		
`11:   else if (x>z)`				•							
`12:      m = x;`											
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	
`}`											
Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

$$susp(7) = \frac{\frac{2}{4}}{\frac{2}{6} + \frac{2}{4}} = 0.60$$

Technique: Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

int x,y,z,m;	t1 3,3,5	t2 1,2,3	t3 3,2,2	t4 5,5,5	t5 1,1,4	t6 5,3,4	t7 3,2,1	t8 2,1,3	t9 5,4,2	t10 5,2,6	suspiciousness
1:read(“Enter 3 integers:”,x,y,z);	•	•	•	•	•	•	•	•	•	•	0.50
2:m = z;	•	•	•	•	•	•	•	•	•	•	0.50
3:if (y<z)	•	•	•	•	•	•	•	•	•	•	0.50
4: if (x<y)	•	•			•	•		•		•	0.43
5: m = y;		•									0.00
6: else if (x<z)	•				•	•		•		•	0.50
7: m = y;	•				•			•		•	0.60
8:else			•	•			•		•		0.60
9: if (x>y)			•	•			•		•		0.60
10: m = z;			•				•		•		0.75
11: else if (x>z)				•							0.00
12: m = x;											0.00
13:print(“Middle number is:”, m);	•	•	•	•	•	•	•	•	•	•	0.50
} Pass/fail Status	P	P	P	P	P	P	F	F	F	F	

Technique: Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	suspiciousness	rank
`int x,y,z,m;`	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50	9
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	0.50	9
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	0.50	9
`4:    if (x<y)`	•	•			•	•		•		•	0.43	10
`5:        m = y;`		•									0.00	13
`6:    else if (x<z)`	•				•	•		•		•	0.50	9
`7:        m = y;`	•				•			•		•	0.60	4
`8:else`			•	•			•		•		0.60	4
`9:    if (x>y)`			•	•			•		•		0.60	4
`10:       m = z;`			•				•		•		0.75	1
`11:    else if (x>z)`				•							0.00	13
`12:       m = x;`											0.00	13
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	0.50	9
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F		

Technique: Tarantula

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	suspiciousness	rank
`int x,y,z,m;`	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	2,1,3	5,4,2	5,2,6		
`1:read("Enter 3 integers:",x,y,z);`	•	•	•	•	•	•	•	•	•	•	0.50	9
`2:m = z;`	•	•	•	•	•	•	•	•	•	•	0.50	9
`3:if (y<z)`	•	•	•	•	•	•	•	•	•	•	0.50	9
`4:   if (x<y)`	•	•			•	•		•		•	0.43	10
`5:       m = y;`		•										13
`6:   else if (x<z)`	•											9
`7:       m = y;`	•											4
`8:else`												4
`9:   if (x>y)`			•	•			•		•		0.60	4
`10:       m = z; //bug;correct m=y`			•				•		•		0.75	1
`11:   else if (x>z)`				•							0.00	13
`12:       m = x;`											0.00	13
`13:print("Middle number is:", m);`	•	•	•	•	•	•	•	•	•	•	0.50	9
`}` Pass/fail Status	P	P	P	P	P	P	F	F	F	F		

cost metric

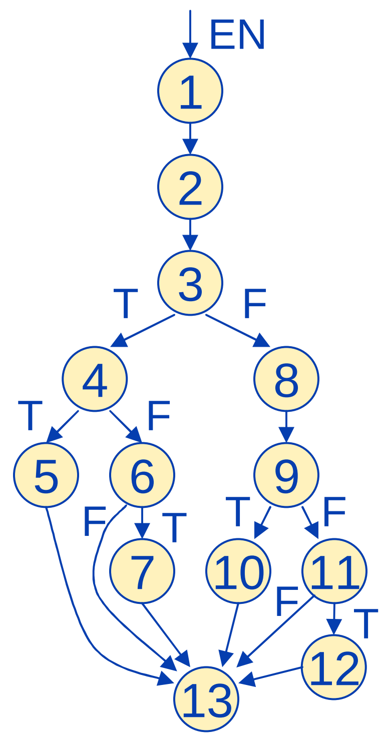
1/13 statements (7.7%)

Comparing Multiple Coverage Types

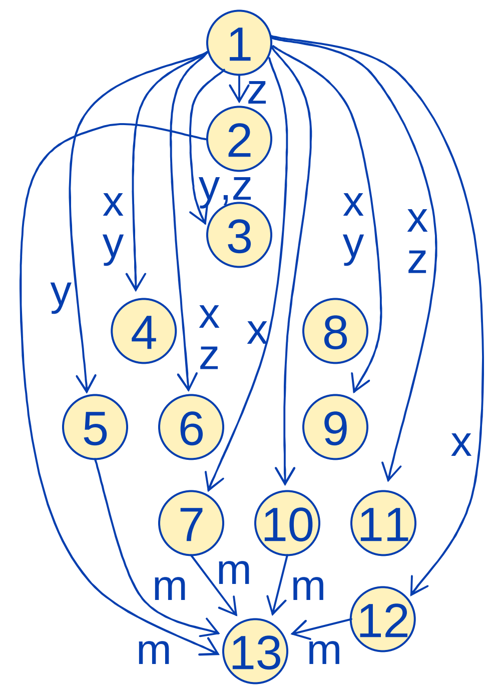
Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Branches



DU-pairs



Execution

statement coverage

Tarantula

statement scores, ranking

Execution

branch coverage

Tarantula

branch scores, ranking

Execution

du-pair coverage

Tarantula

du-pair scores, ranking

Comparing Multiple Coverage Types

Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Execution

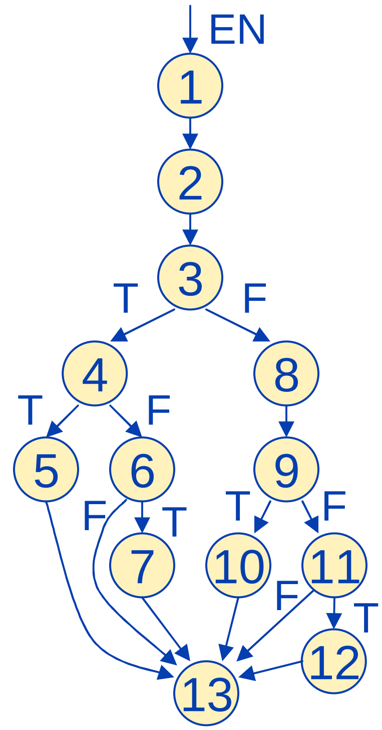
statement coverage

Tarantula

statement scores, ranking

?

Branches



Execution

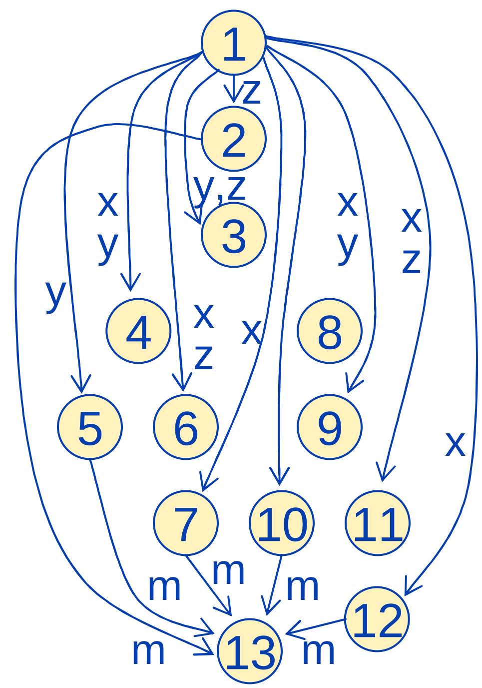
branch coverage

Tarantula

branch scores, ranking

?

DU-pairs



Execution

du-pair coverage

Tarantula

du-pair scores, ranking

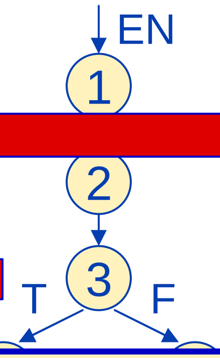
Comparing Multiple Coverage Types

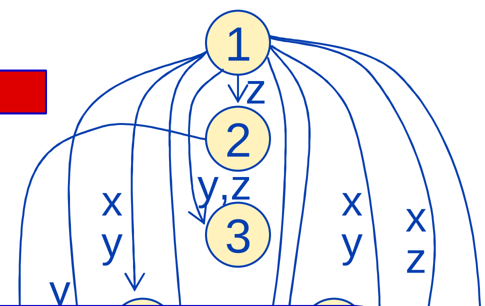
Statements | Branches | DU-pairs

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.
8.  e
9.
10.
11.
12.
13. pr
```

map

map





Solution

Statement ranking for each type!

Execution

statement coverage

Tarantula

statement scores, ranking

?

Execution

branch coverage

Tarantula

branch scores, ranking

?

Execution

du-pair coverage

Tarantula

du-pair scores, ranking

Branch, Du-pair Mappings

1. Gather branch (du-pair) coverage
2. Compute suspiciousness for each branch (du-pair)
3. Map branches (du-pairs) to statements
4. Transfer suspiciousness scores of branches (du-pairs) to corresponding statements
5. Rank statements using the transferred suspiciousness scores

Result: statement rankings based on branches (du-pairs)

Experiment 1

Goal: compare individual coverage types

Setup

- Coverage tool: DUA-Forensics [Santelices, Harrold ASE 2007]
  - Target language: Java bytecode
  - Monitors statements, branches, and du-pairs
- Fault-localization tool: Tarantula with Mapper
- Measure of cost: % of statements visited to reach first faulty statement

Experiment 1: Subjects

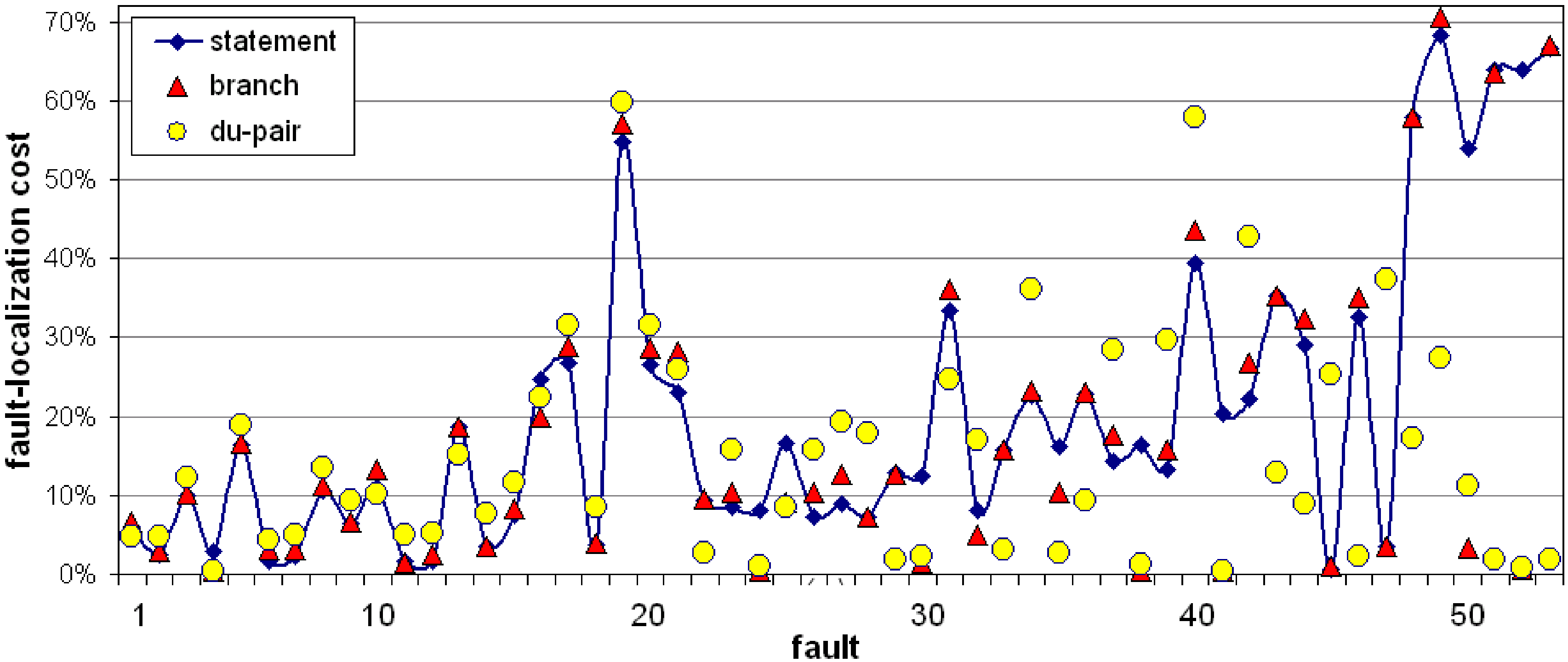
- Siemens suite translated to Java
- NanoXML
- XML-Security
- JABA
- Total: 14 subjects

subject	LOC	tests	faults
Tcas	131	1608	10
Tot_info	283	1052	10
Schedule1	290	2650	9
Schedule2	317	2710	7
Print_tokens1	478	4130	5
Print_tokens2	410	4115	10
NanoXML v1	3497	214	7
NanoXML v2	4009	214	7
NanoXML v3	4608	216	8
NanoXML v5	4782	216	7
XML-security v1	21613	92	7
XML-security v2	22318	94	7
XML-security v3	19895	84	2
JABA	37966	677	11

TOTAL FAULTS	**107**

Experiment 1: Results

Faults with most different costs (>1% between best and worst coverage type)

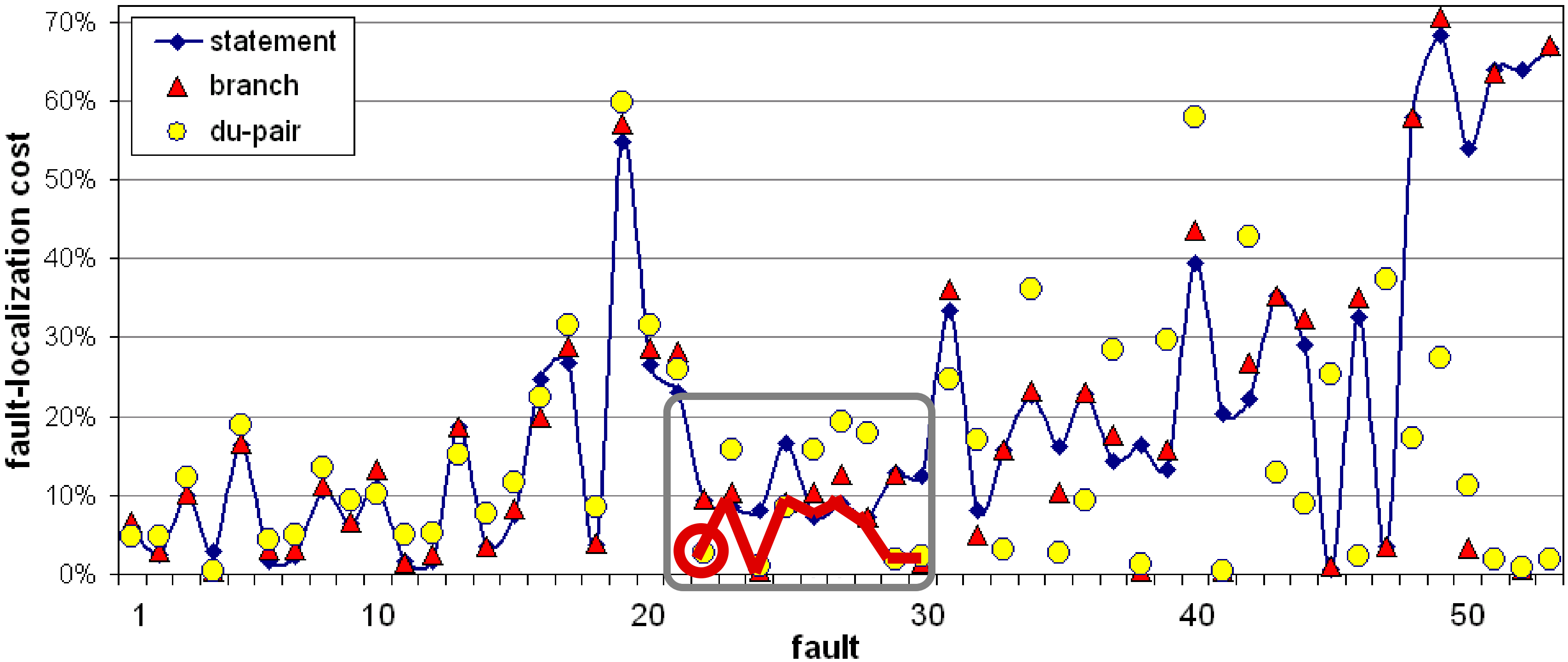


For all 107 faults:

measure	statement	branch	du-pair
average cost	11.49%	10.24%	9.02%
standard dev.	*16.25%*	*15.40%*	*12.04%*

Experiment 1: Results

Faults with most different costs (>1% between best and worst coverage type)

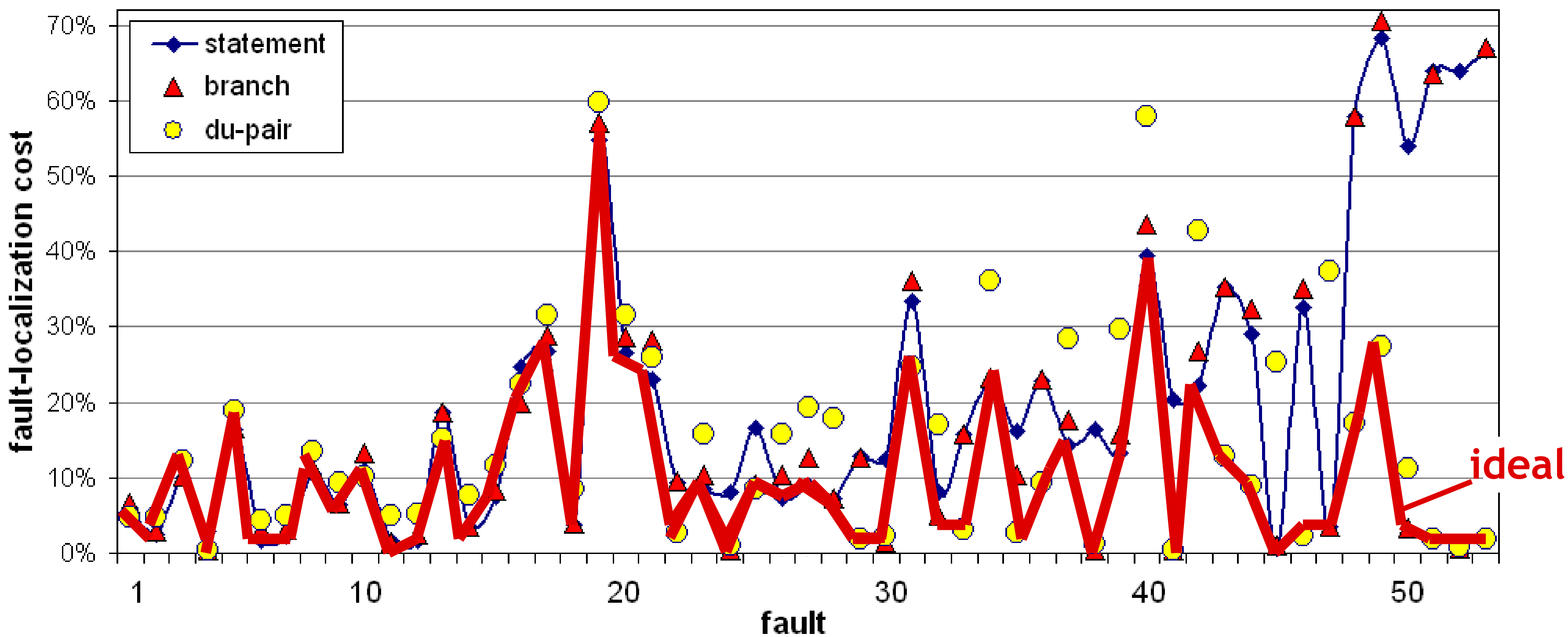


For all 107 faults:

measure	statement	branch	du-pair
average cost	**11.49%**	**10.24%**	**9.02%**
standard dev.	*16.25%*	*15.40%*	*12.04%*

Experiment 1: Results

Faults with most different costs (>1% between best and worst coverage type)



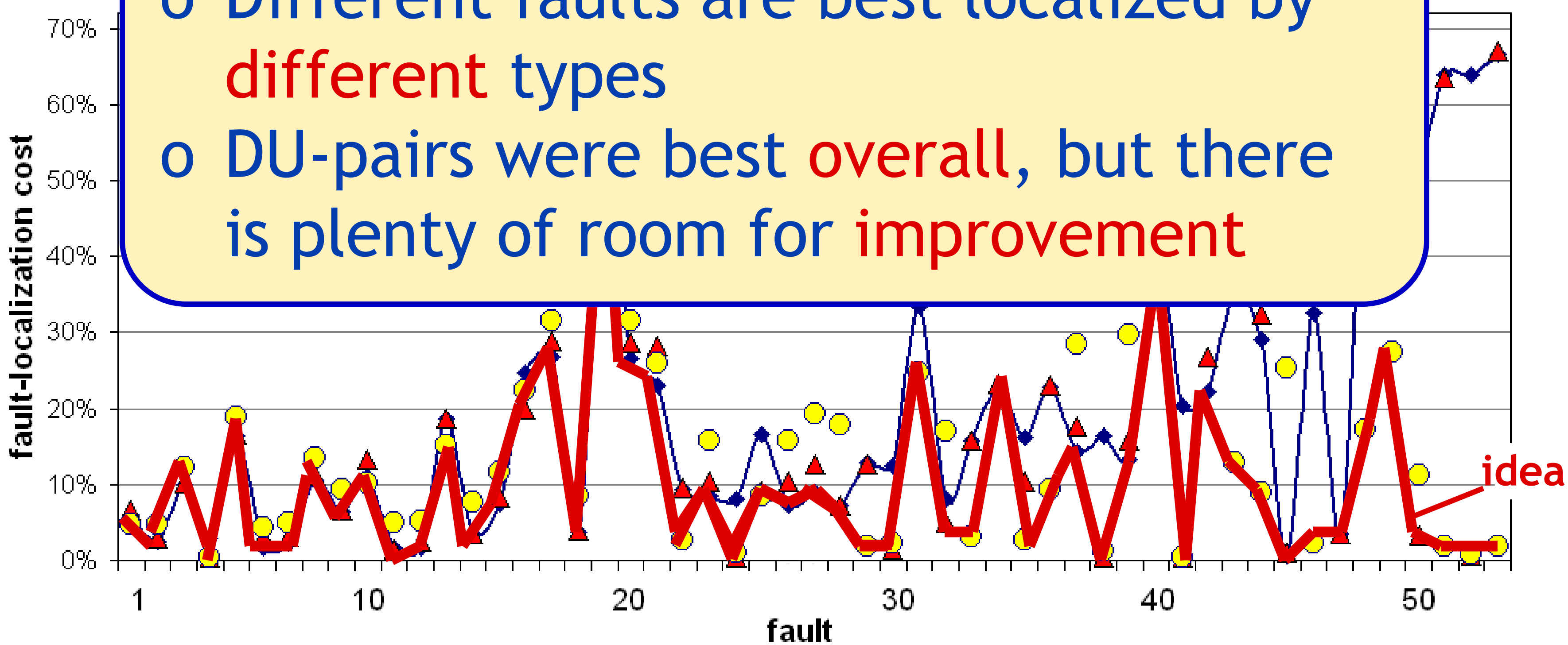
For all
107 faults:

measure	statement	branch	du-pair	ideal
average cost	11.49%	10.24%	9.02%	6.35%
standard dev.	*16.25%*	*15.40%*	*12.04%*	*9.27%*

Experiment 1: Results

Faults ... ype)

- Different faults are best localized by different types
- DU-pairs were best overall, but there is plenty of room for improvement



For all 107 faults:

measure	statement	branch	du-pair	ideal
average cost	11.49%	10.24%	9.02%	6.35%
standard dev.	*16.25%*	*15.40%*	*12.04%*	*9.27%*

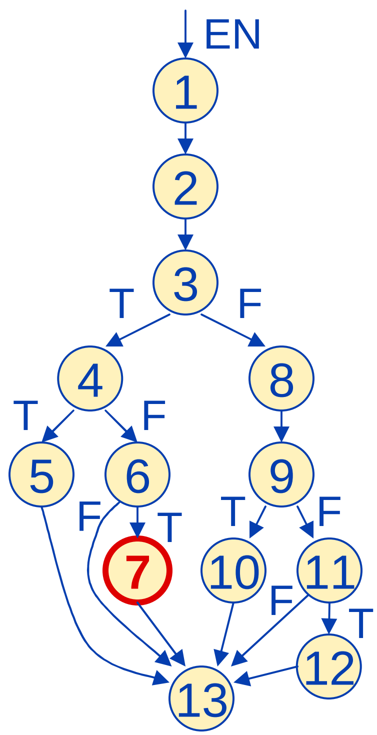
Combining Coverage Types

Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y; // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

fault

Branches



DU-pairs

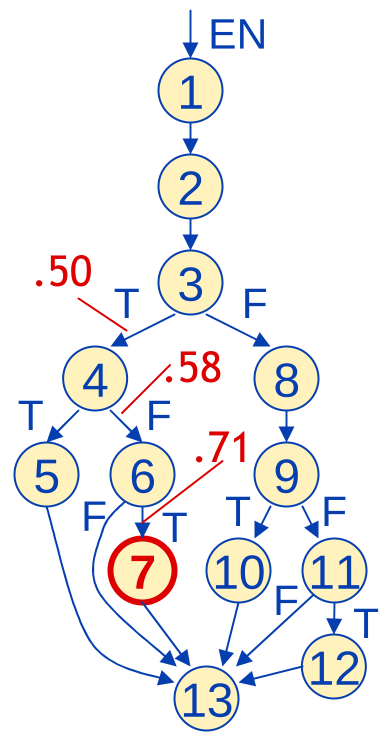
Combining Coverage Types

Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y; // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

fault

Branches



DU-pairs

stmt.	score	rank
6	.71	2
7	.71	2

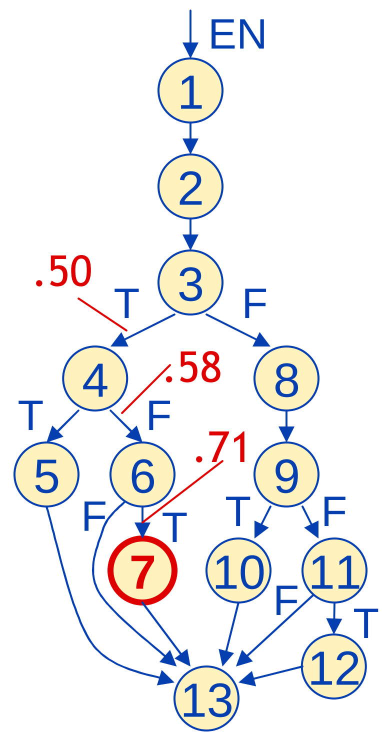
Combining Coverage Types

Statements

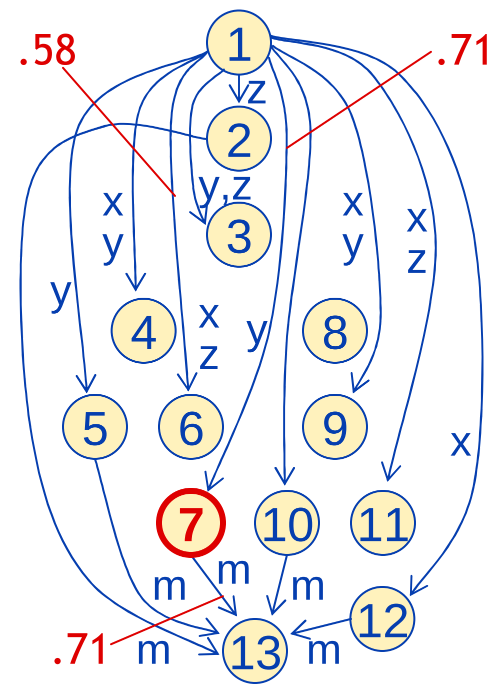
```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y; // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

fault

Branches



DU-pairs



stmt.	score	rank
6	.71	2
7	.71	2

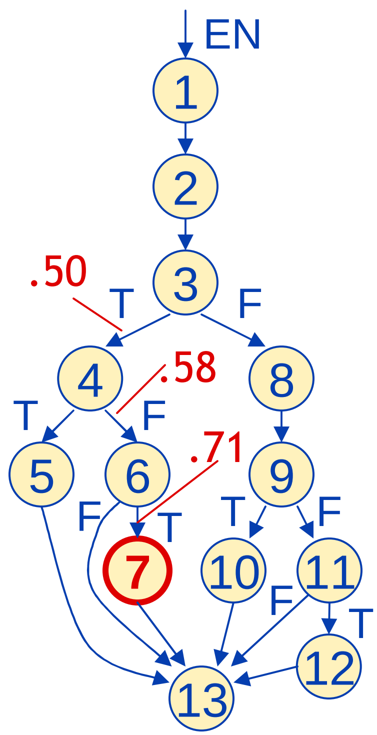
Combining Coverage Types

Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y; // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

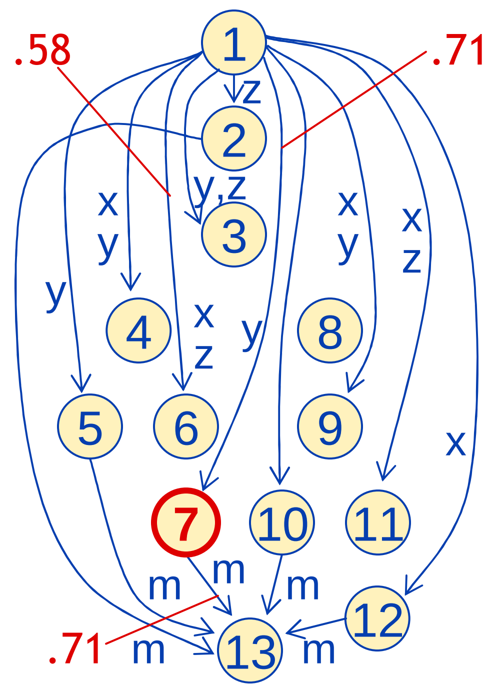
fault

Branches



stmt.	score	rank
6	.71	2
7	.71	2

DU-pairs



stmt.	score	rank
1	.71	3
7	.71	3
13	.71	3

Combining Coverage Types

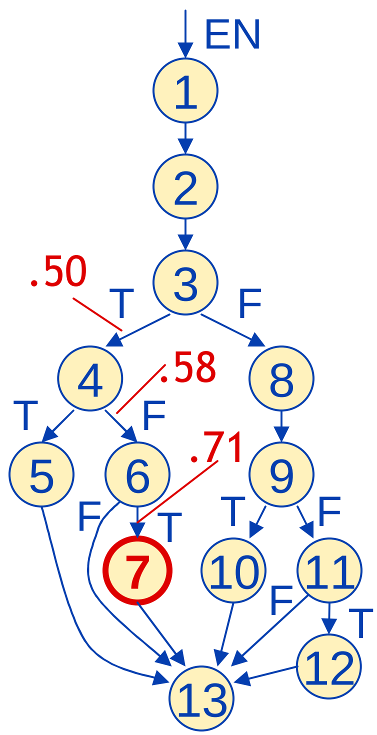
Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y;  // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

fault

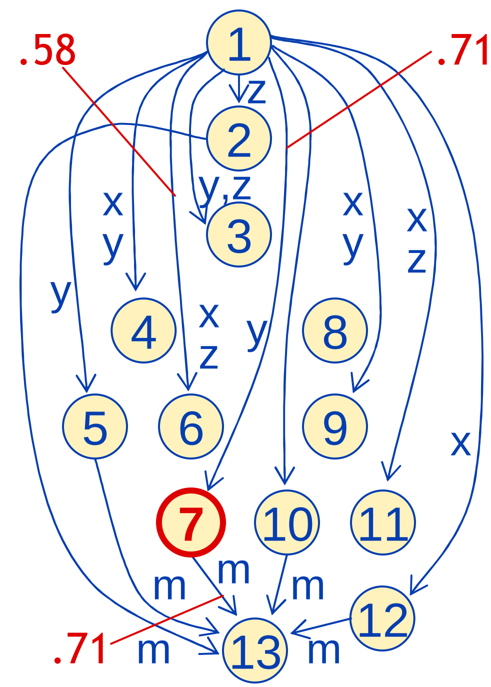
stmt.	average score	rank
1	(.50+.71)/2	
6		
7		
13		

Branches



stmt.	score	rank
6	.71	2
7	.71	2

DU-pairs



stmt.	score	rank
1	.71	3
7	.71	3
13	.71	3

Combining Coverage Types

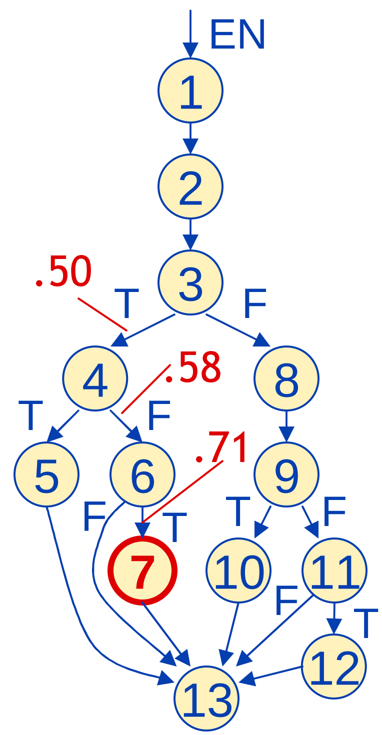
Statements

```
mid(): int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = y;  // m = x
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

fault

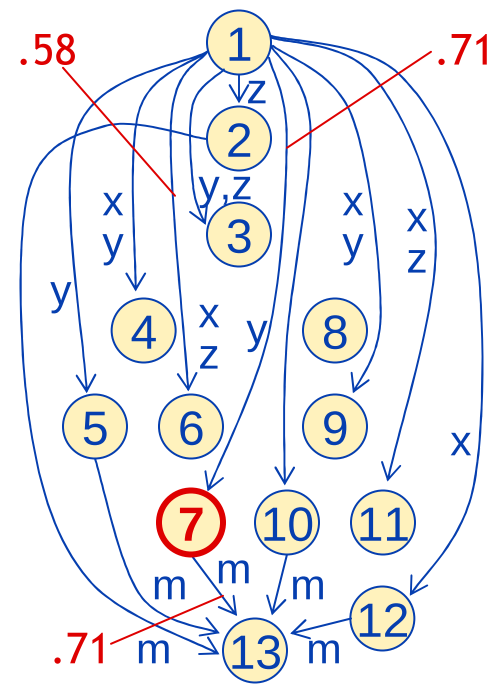
stmt.	average score	rank
1	.61	4
6	.65	2
7	.71	1
13	.56	3

Branches



stmt.	score	rank
6	.71	2
7	.71	2

DU-pairs



stmt.	score	rank
1	.71	3
7	.71	3
13	.71	3

Experiment 2

Goal: compare combinations with individual types

Setup

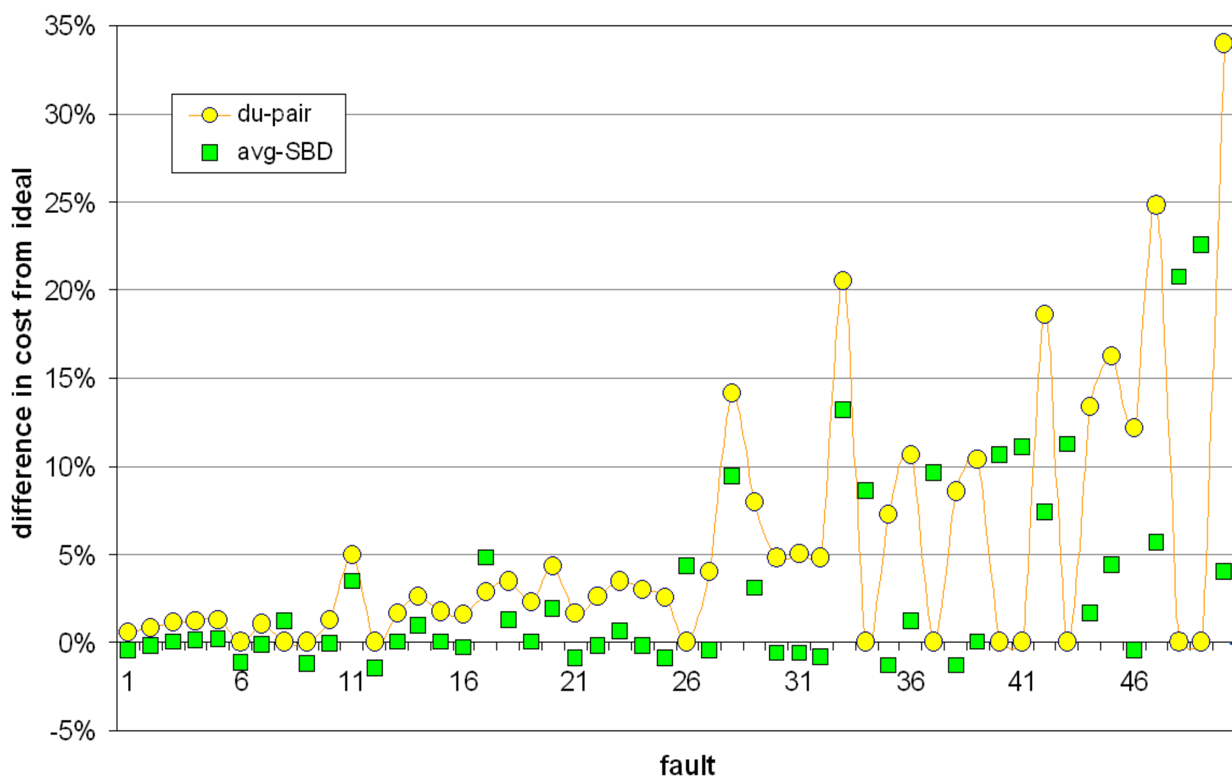
- Three combinations of scores for statements
  - ***avg-BD*(*s*)** = the average score of branches and du-pairs only, associated with *s*
  - ***avg-SBD*(*s*)** = the average score of all statements, branches, and du-pairs associated with statement *s*
  - ***max-SBD*(*s*)** = the maximum score of all statements, branches, and du-pairs associated with statement *s*
- Same as Experiment 1: 14 subjects, 107 faults
  - We measure the difference in cost with the ideal individual type

Experiment 2: Results

Difference in cost w.r.t. the ideal individual type

best

measure	stmt	branch	du-pair	avg-BD	avg-SBD	max-SBD
average diff	**5.14%**	**3.89%**	**2.68%**	**1.80%**	**1.48%**	**2.77%**
standard dev.	*13.28%*	*11.06%*	*5.50%*	*4.66%*	*4.03%*	*5.03%*



Faults with most different costs (>1% between *du-pair* and *avg-SBD*)

avg-SBD is the best combination and better than individual coverage types

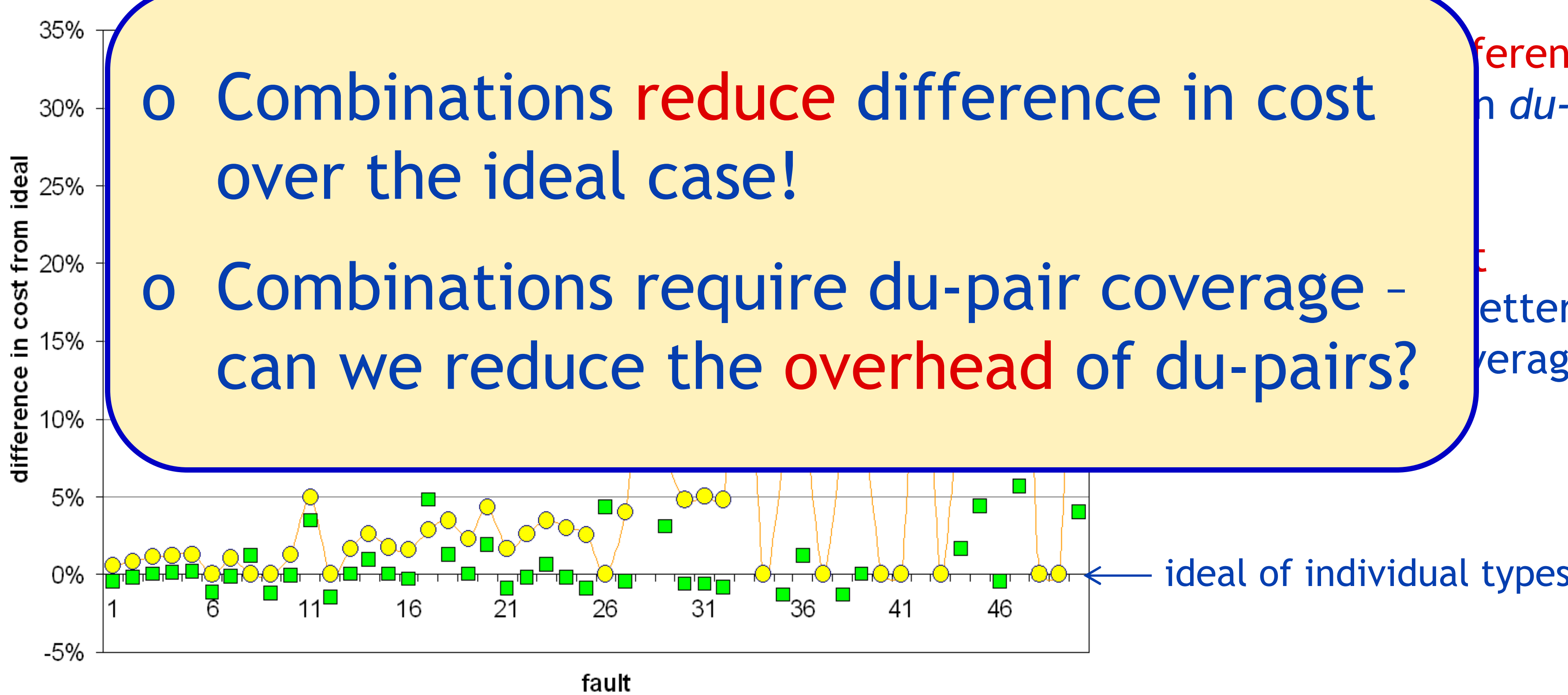
ideal of individual types

Experiment 2: Results

Difference in cost w.r.t. the ideal individual type

measure	stmt	branch	du-pair	avg-BD	avg-SBD	max-SBD
average diff	**5.14%**	**3.89%**	**2.68%**	**1.80%**	**1.48%**	**2.77%**
standard dev.	*13.28%*	*11.06%*	*5.50%*	*4.66%*	*4.03%*	*5.03%*

best



- Combinations reduce difference in cost over the ideal case!
- Combinations require du-pair coverage – can we reduce the overhead of du-pairs?

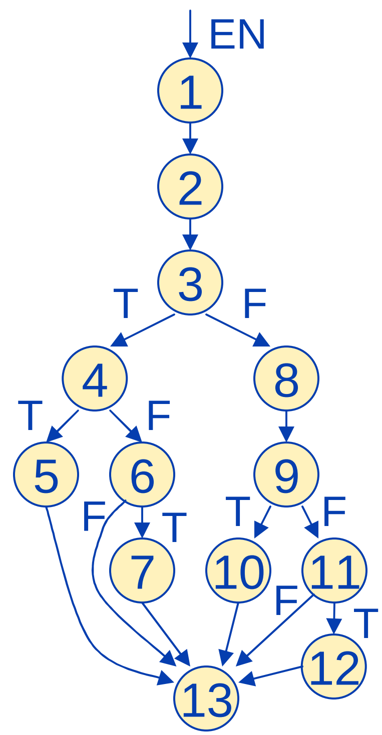
Monitoring Overhead

Statements

```
mid()
    int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

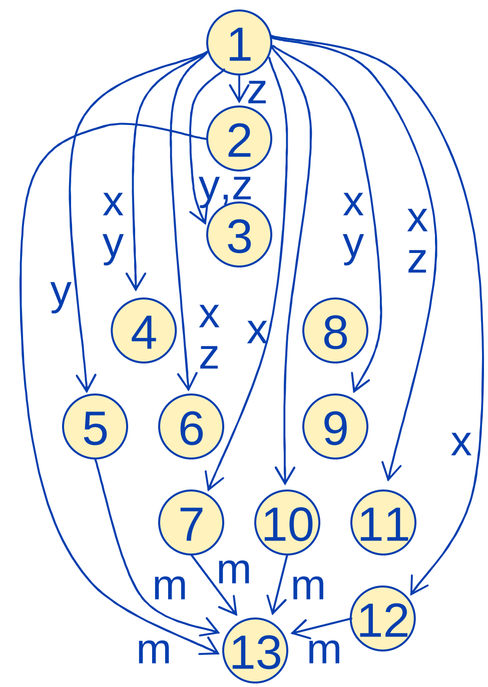
10-20% overhead

Branches



10-20% overhead

DU-pairs



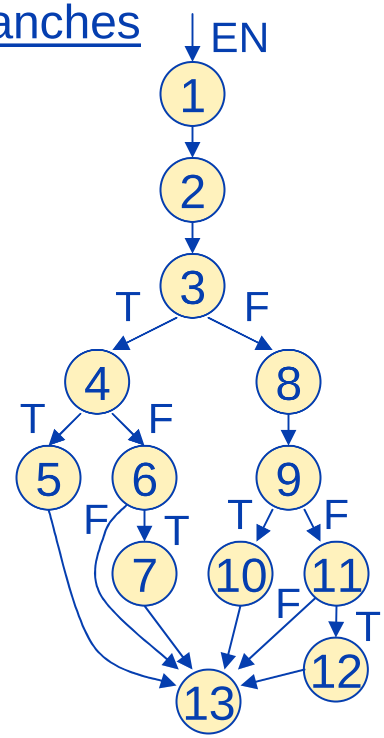
60-120% overhead

Inferring DU-pair Coverage

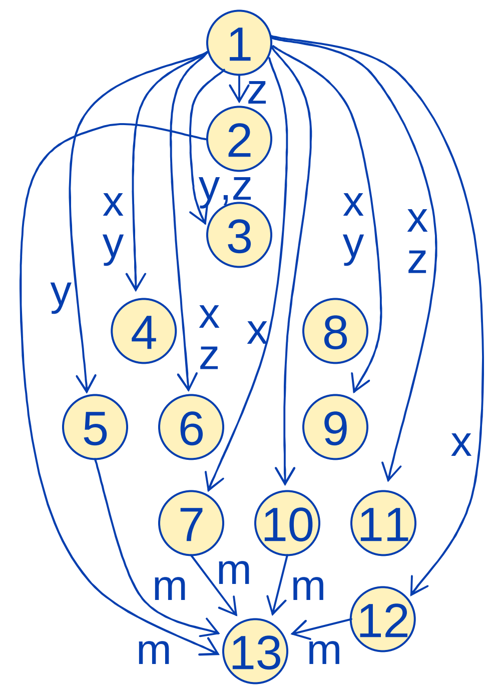
Statements

```
mid()
    int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Branches



DU-pairs



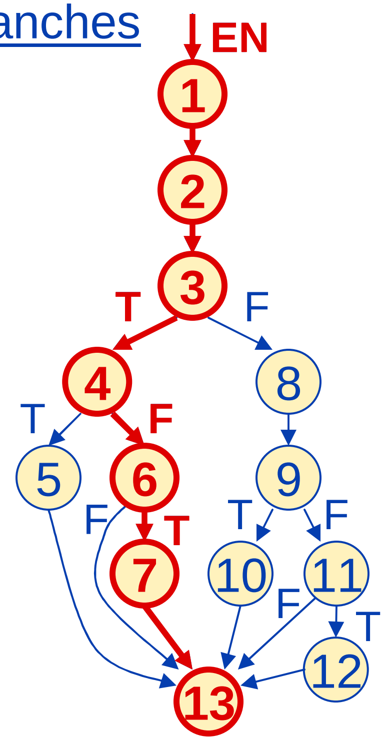
Santelices & Harrold
ASE 2007

Inferring DU-pair Coverage

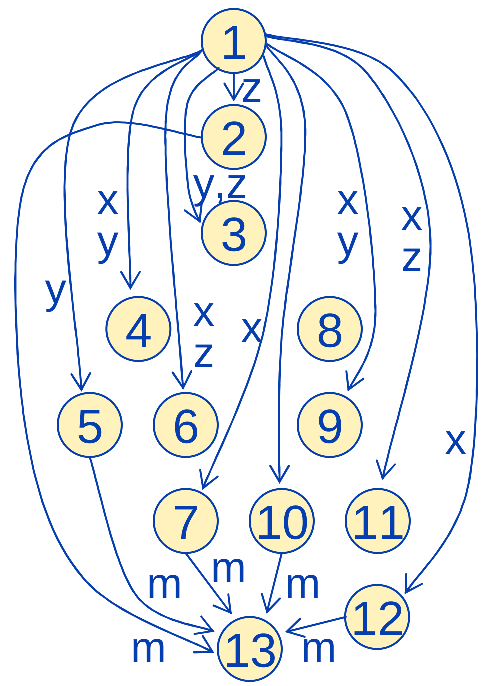
Statements

```
mid()
    int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Branches



DU-pairs



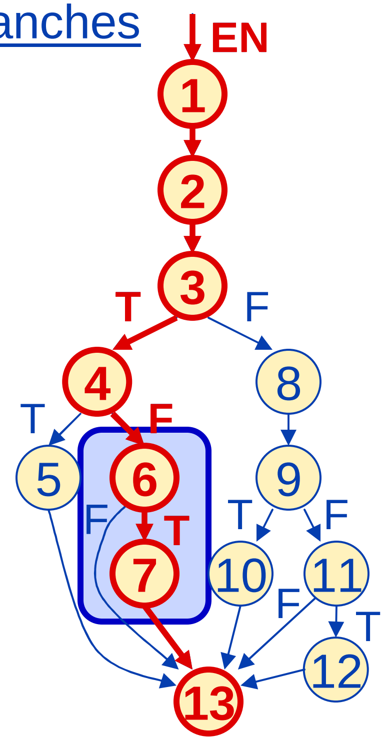
Santelices & Harrold
ASE 2007

Inferring DU-pair Coverage

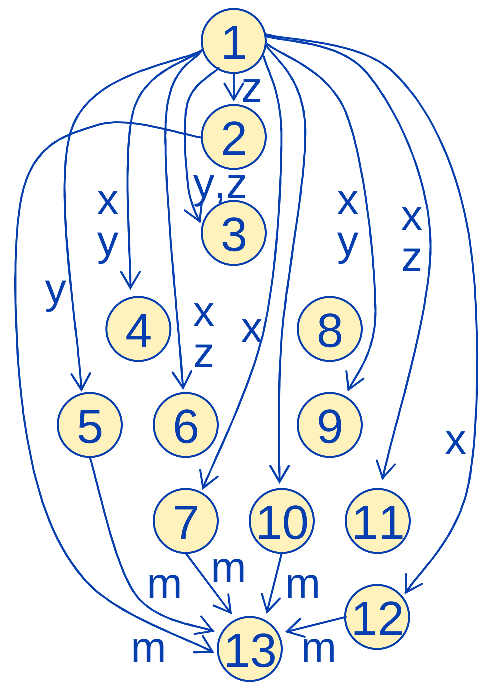
Statements

```
mid()
    int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Branches



DU-pairs



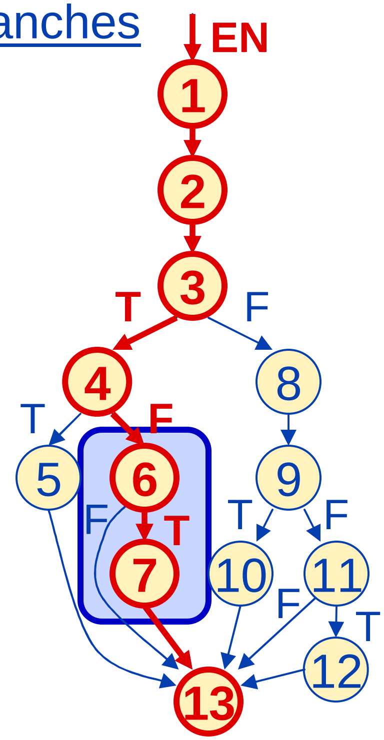
Santelices & Harrold
ASE 2007

Inferring DU-pair Coverage

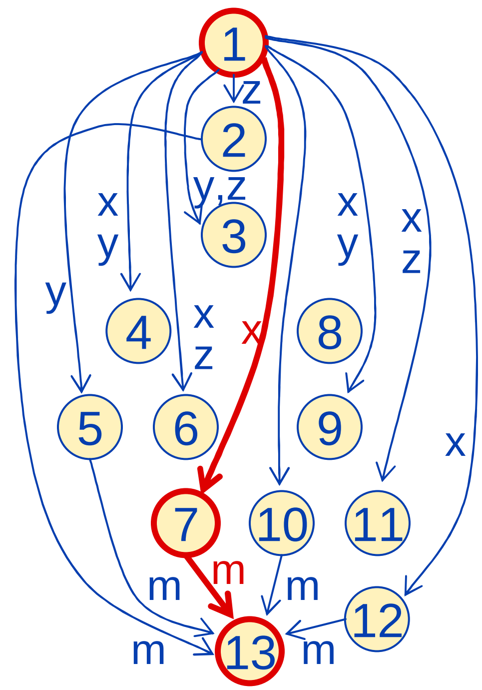
Statements

```
mid()
    int x,y,z,m;
1.  read(x,y,z);
2.  m = z;
3.  if (y<z)
4.    if (x<y)
5.      m = y;
6.    else if (x<z)
7.      m = x;
8.  else
9.    if (x>y)
10.     m = y;
11.   else if (x>z)
12.     m = x;
13. print(m);
```

Branches



DU-pairs



inference

$$6T \Rightarrow 1 \xrightarrow{x} 7,\ 7 \xrightarrow{m} 13$$

(Santelices & Harrold ASE 2007)

Experiment 3

Goal: comparison using inferred du-pair coverage

Setup

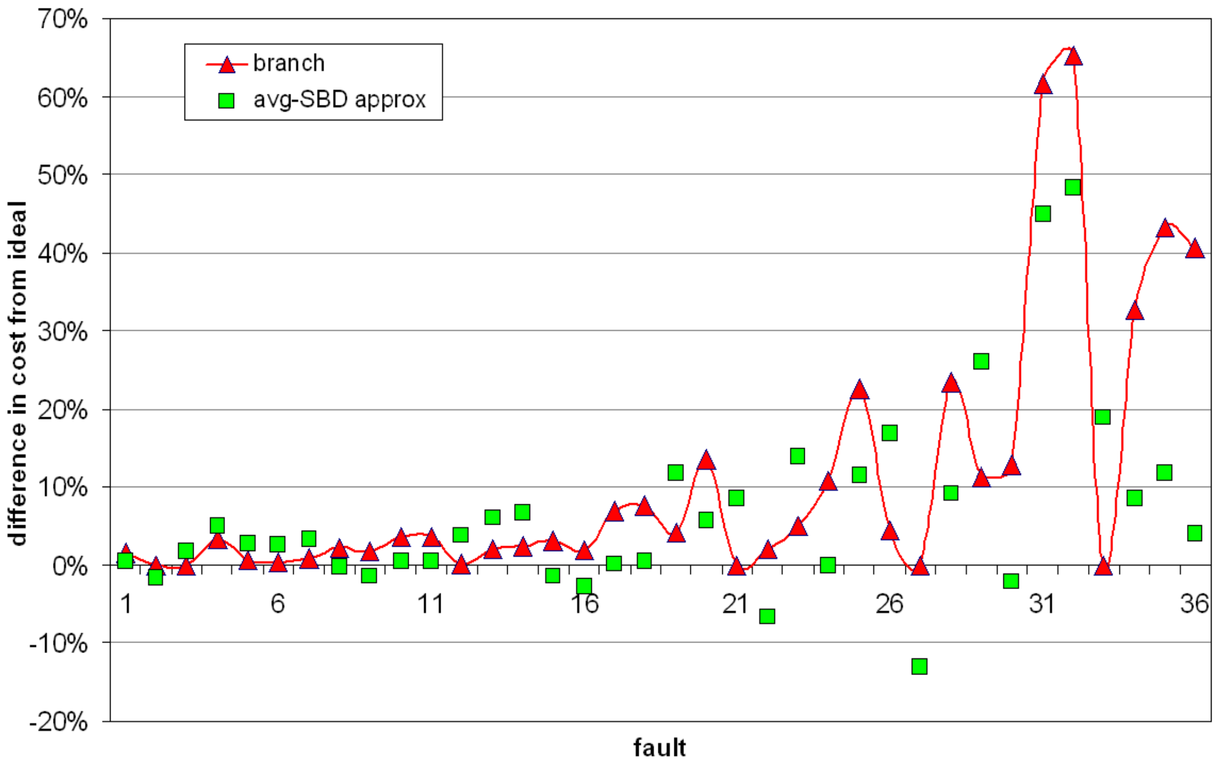
- Replace du-pair coverage with inferred coverage
  - ***du-pair-approx*(*s*)**
  - ***avg-BD-approx*(*s*)**
  - ***avg-SBD-approx*(*s*)**
  - ***max-SBD-approx*(*s*)**
- Same as Experiments 1 and 2: 14 subjects, 107 faults
  - We measure the difference in cost with the ideal individual type

Note: we only monitor for branch coverage now!

Experiment 3: Results

Difference in cost w.r.t. the ideal individual type

measure	stmt	branch	du-pair approx	avg-BD approx	best avg-SBD approx	max-SBD approx
average diff	**5.14%**	**3.89%**	**4.26%**	**2.64%**	**2.44%**	**3.71%**
standard dev.	*13.28%*	*11.06%*	*10.01%*	*8.02%*	*7.73%*	*7.70%*



Faults with most different costs (>1% between *du-pair* and *avg-SBD*)

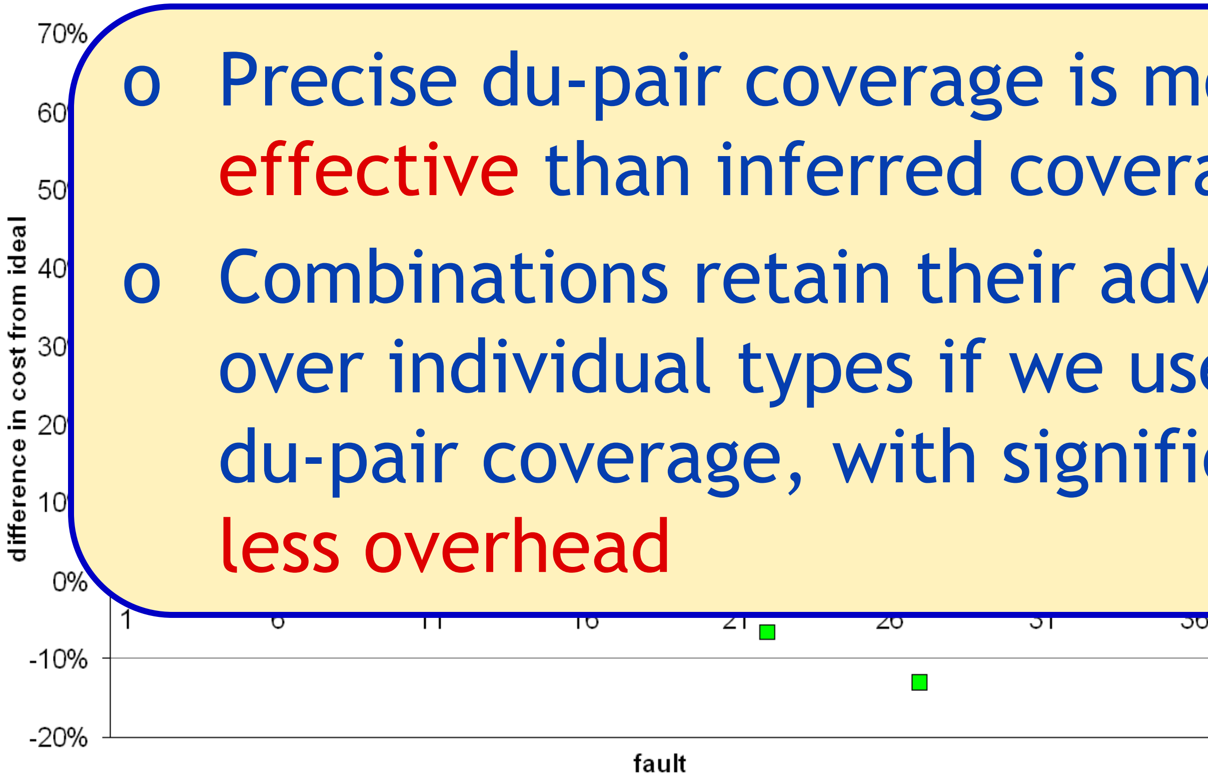
avg-SBD-approx is the best combination for inferred du-pair coverage

ideal of individual types

Experiment 3: Results

Difference in cost w.r.t. the ideal individual type

measure	stmt	branch	du-pair approx	avg-BD approx	best avg-SBD approx	max-SBD approx
average diff	**5.14%**	**3.89%**	**4.26%**	**2.64%**	**2.44%**	**3.71%**
standard dev.	*13.28%*	*11.06%*	*10.01%*	*8.02%*	*7.73%*	*7.70%*



- Precise du-pair coverage is more effective than inferred coverage
- Combinations retain their advantage over individual types if we use inferred du-pair coverage, with significantly less overhead

different
een *du-*
D)

is the
n for
coverage

dual types

Summary

Experiment 1

Different faults are best found using different coverage types

Summary

Experiment 1

Different faults are best found using different coverage types

But, there is no way to know beforehand which type to choose

Summary

Experiment 1

Different faults are best found using different coverage types

Experiment 2

Combining coverage types gives more effective fault localization than any individual type

Summary

Experiment 1

Different faults are best found using different coverage types

Experiment 2

Combining coverage types gives more effective fault localization than any individual type

Instrumentation for du-pairs incurs a relatively high runtime overhead

Summary

Experiment 1

Different faults are best found using different coverage types

Experiment 2

Combining coverage types gives more effective fault localization than any individual type

Experiment 3

Performing du-pair inferencing with only cheap branch instrumentation significantly reduces run-time overhead while retaining better combination effectiveness over individual types

Future Work

- Compare and combine additional lightweight coverage types (e.g., methods, acyclic paths)
- Experiment with additional heavyweight coverage (e.g., slice fragments) inferred from lighter-weight coverage (e.g., branches, paths)
- Perform more experiments on more subjects and faults, especially different types of faults

Conclusion

- First thorough comparison of effectiveness of different coverage types on fault localization
- First technique that combines coverage types to improve fault-localization effectiveness
- Application of du-pair inference that produces effective results with low overhead