FLOW-INSENSITIVE POINTER ANALYSIS

Susan Horwitz
Marc Shapiro

University of Wisconsin

```
x = 0;
*p = 1;
if  ( x )  S1;
else  S2;
```

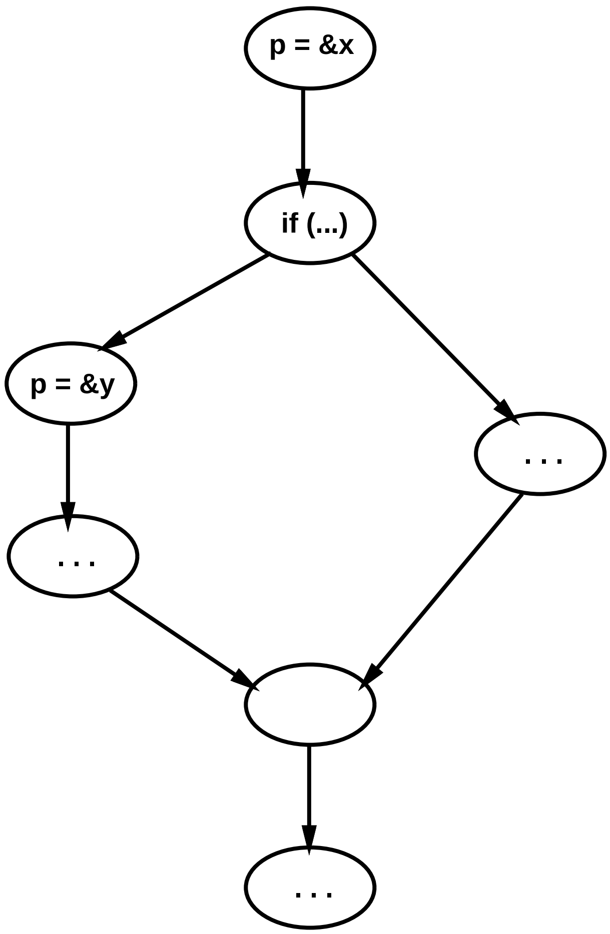
p does NOT point to x

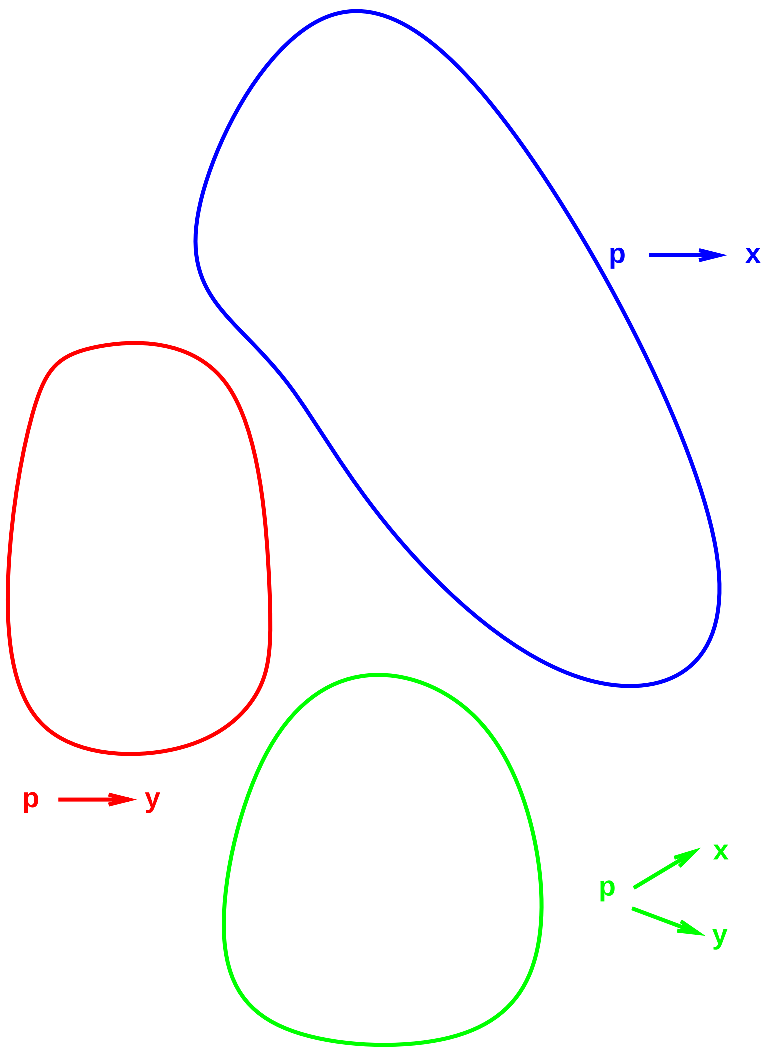
```
x = 0;

*p = 1;

S2;
```

p DOES
point to x

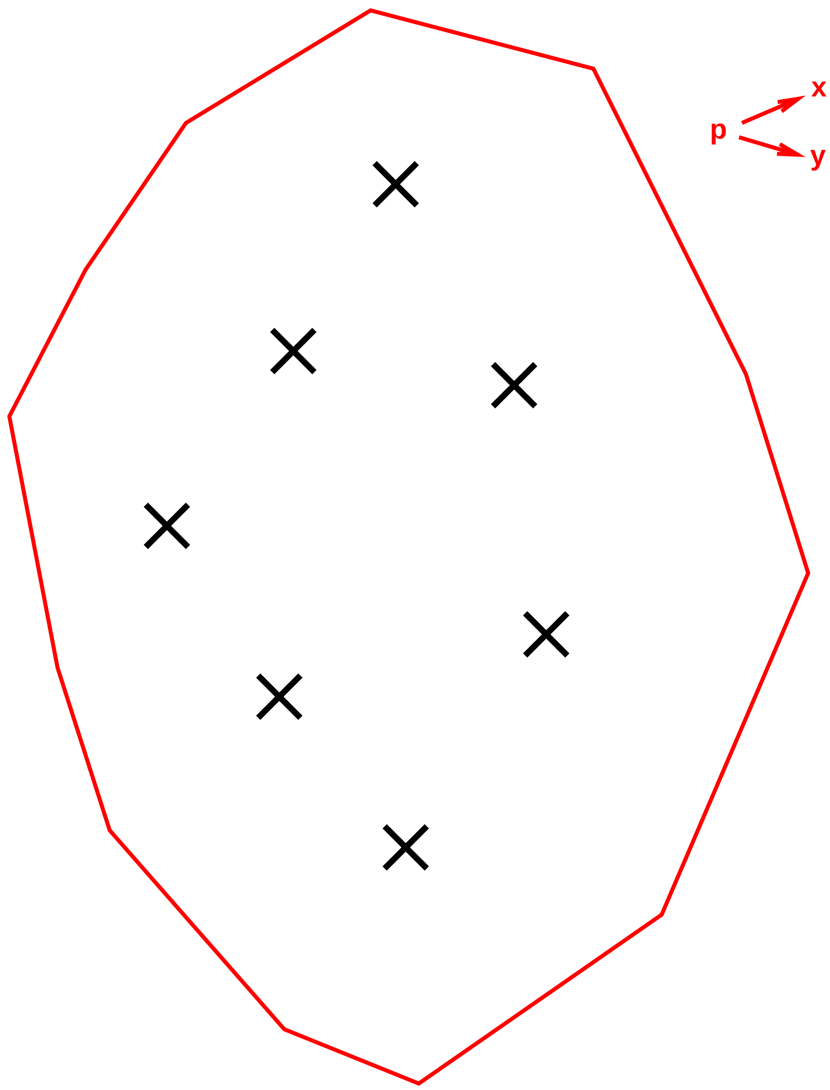
```
*p = 1;

S1;
```

FLOW−SENSITIVE vs FLOW−INSENSITIVE





Flow Sensitive



Flow Insensitive

RESULTS

- **NEW** POINTER-ANALYSIS ALGORITHMS

  + Flow Insensitive

  + Interprocedural

  + Stack-Based Storage

- EXPERIMENTAL RESULTS

  + Times and Precisions

  + Transitive Effects

PREVIOUS ALGORITHMS

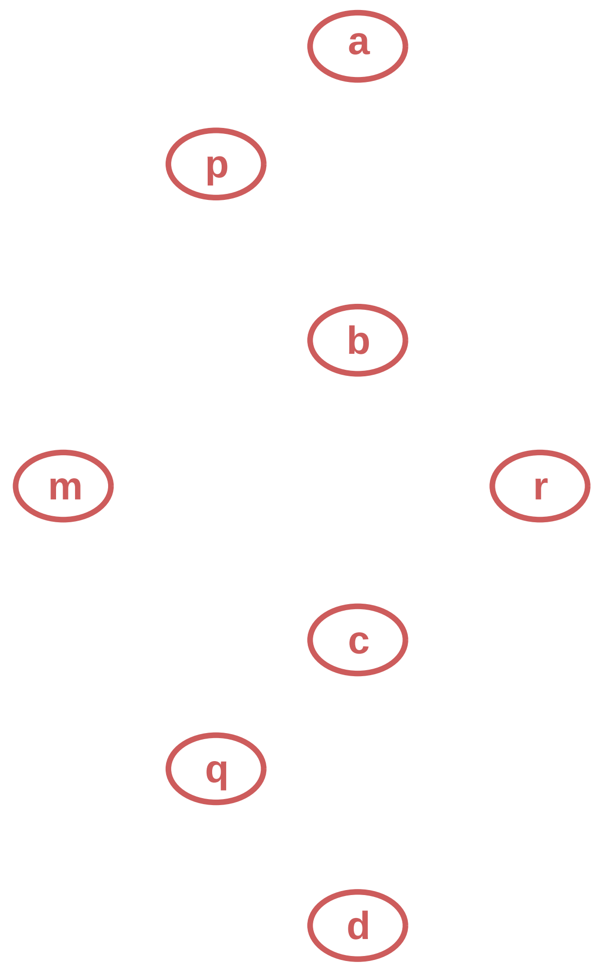
ANDERSEN (1994): $O(n^3)$

STEENSGAARD (1996): $O(n\ \mathtt{a}(n))$

- (almost) full C language
- structs, arrays, and heap storage are "collapsed"
- library functions are inlined

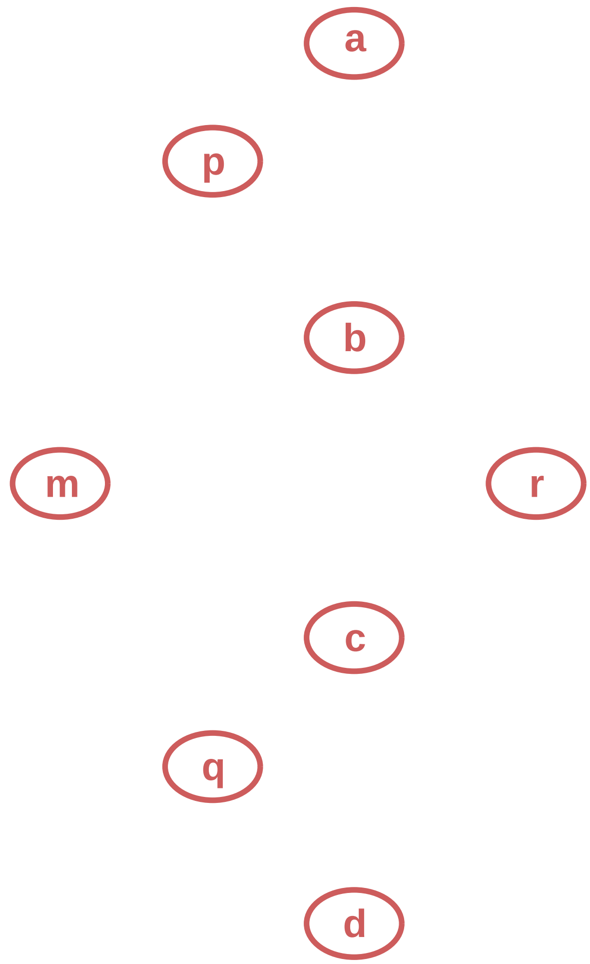
ANDERSEN

```
p = &a;
p = &b;
q = &c;
q = &d;
m = &p;
m = &q;
r = *m;
```



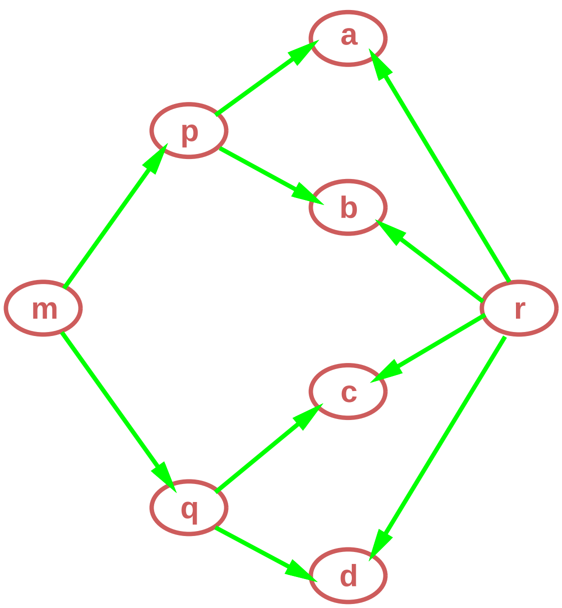
STEENSGAARD

```
p = &a;
p = &b;
q = &c;
q = &d;
m = &p;
m = &q;
r = *m;
```

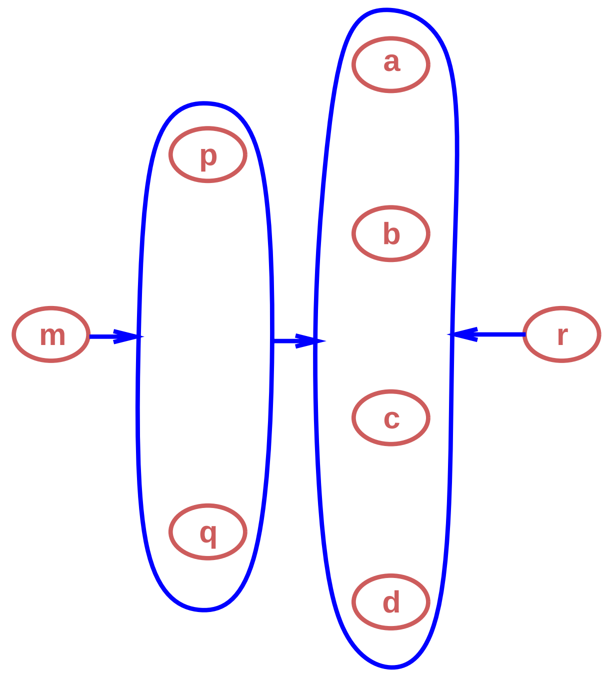


STORAGE SHAPE GRAPHS

ANDERSEN



STEENSGAARD



POINTS−TO SETS

ANDERSEN

STEENSGAARD

m

p

q

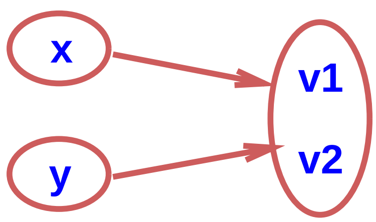
r

STEENSGAARD LOSS OF PRECISION

x points to both v1 and v2

- All points-to sets that contain `v1` also contain `v2`.

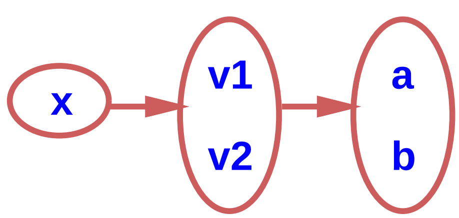
```
x = &v1;
x = &v2;
y = &v1;
```



But y does NOT point to v2!

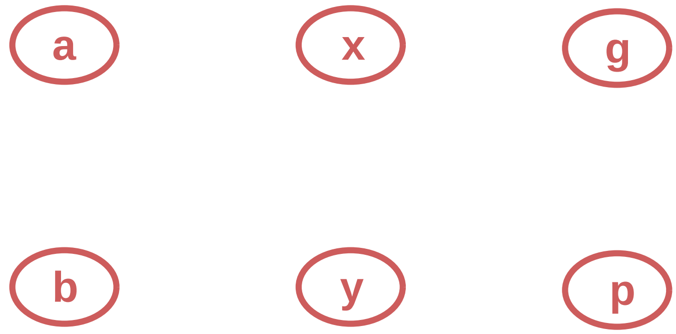
- Everything in `v1`'s points-to set is also in `v2`'s points-to set (and vice-versa).

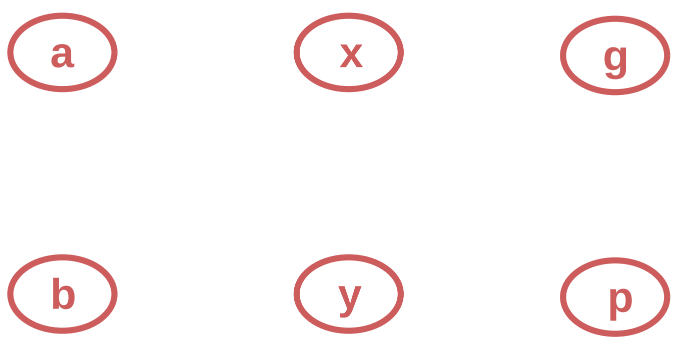
```
x = &v1;
x = &v2;
v1 = &a;
v2 = &b;
```



But v2 does NOT point to a!

```
a = &x;
b = &y;
*a = &g;
a = &y;
y = &p;
```





ANDERSEN vs STEENSGAARD IN PRACTICE

- 61 C PROGRAMS

  300 - 24,300 lines

  (preprocessed code)

- Steensgaard less precise

  average: 4X

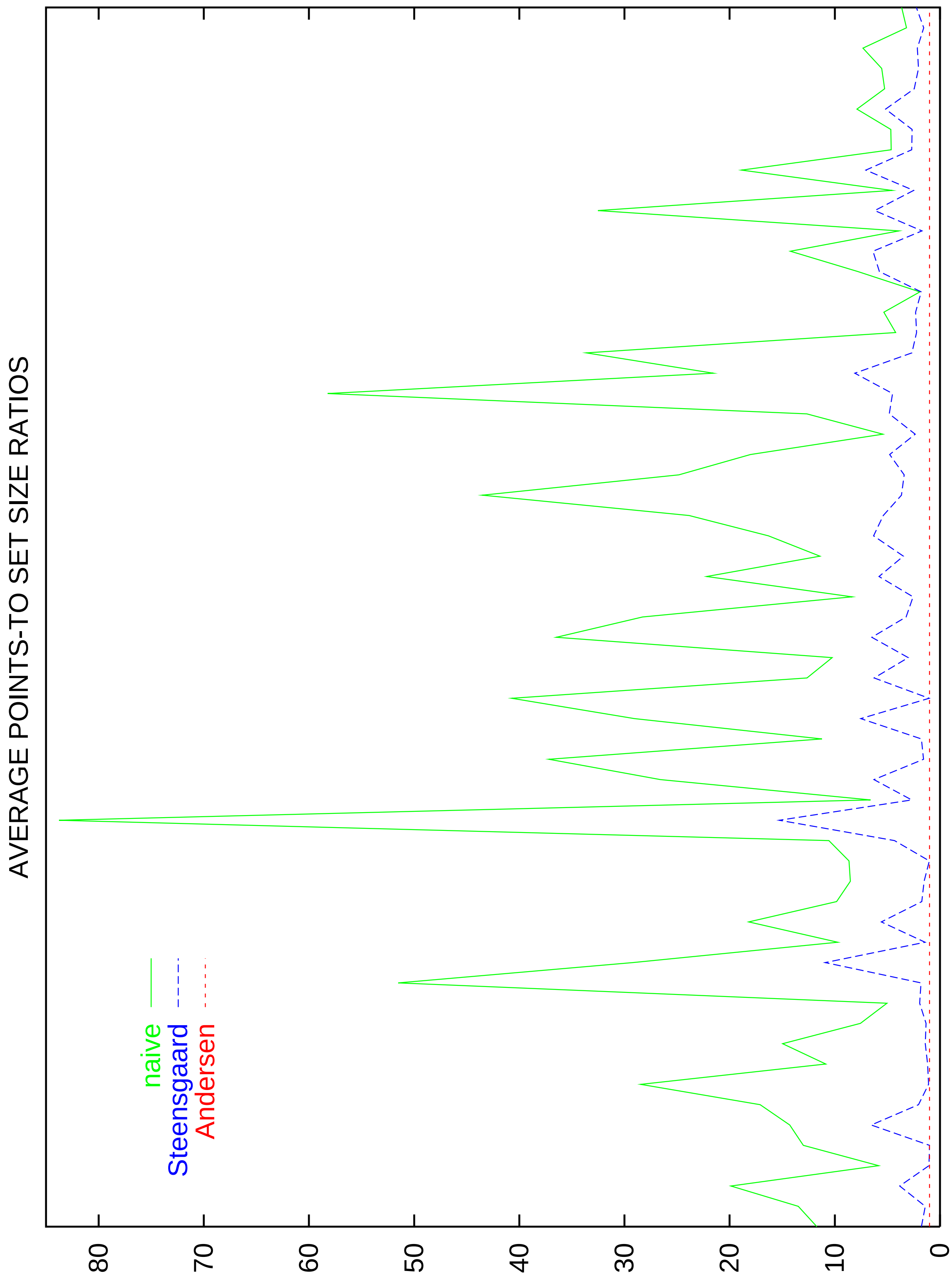
  worst: 15X

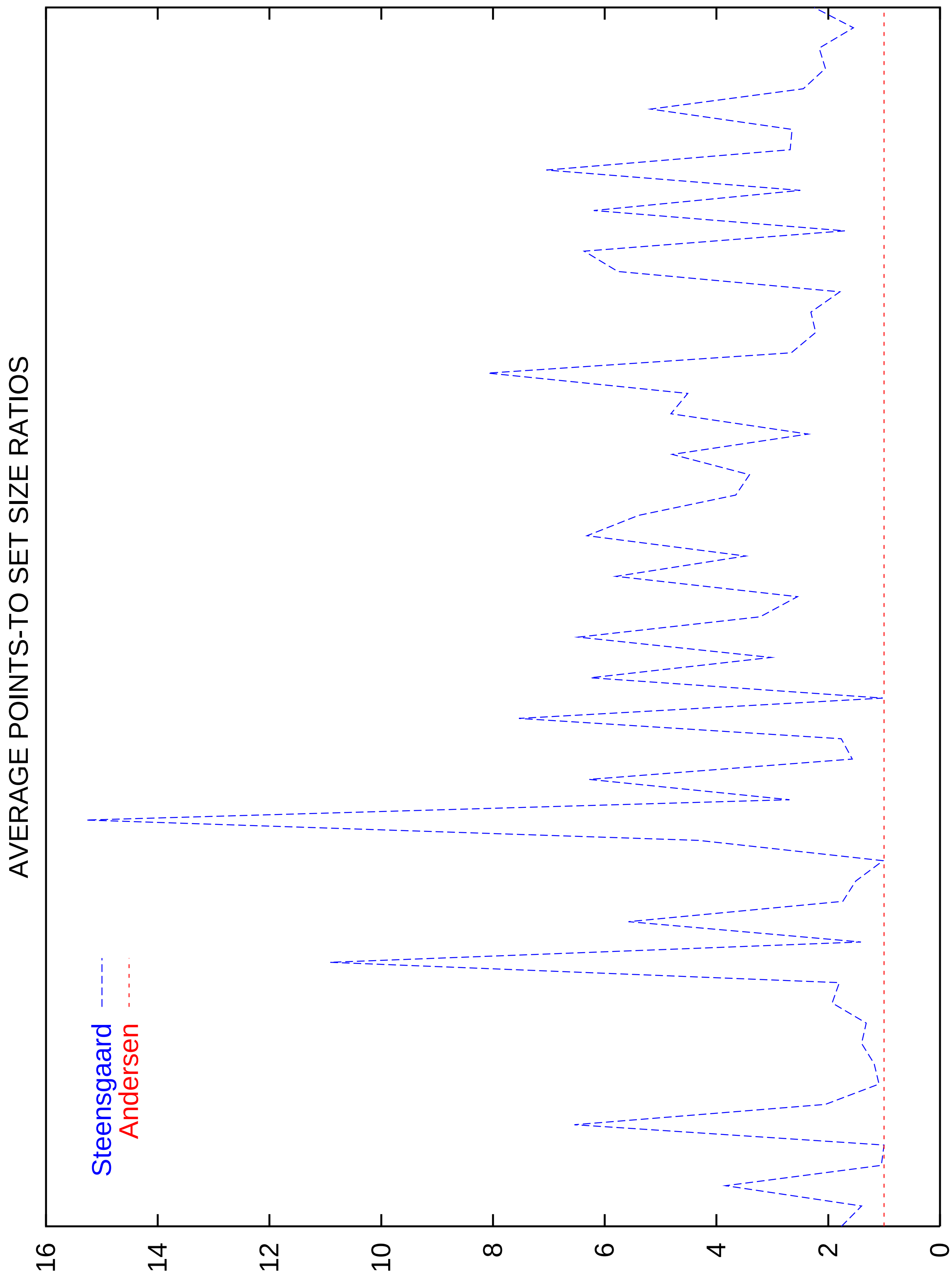
- Andersen slower

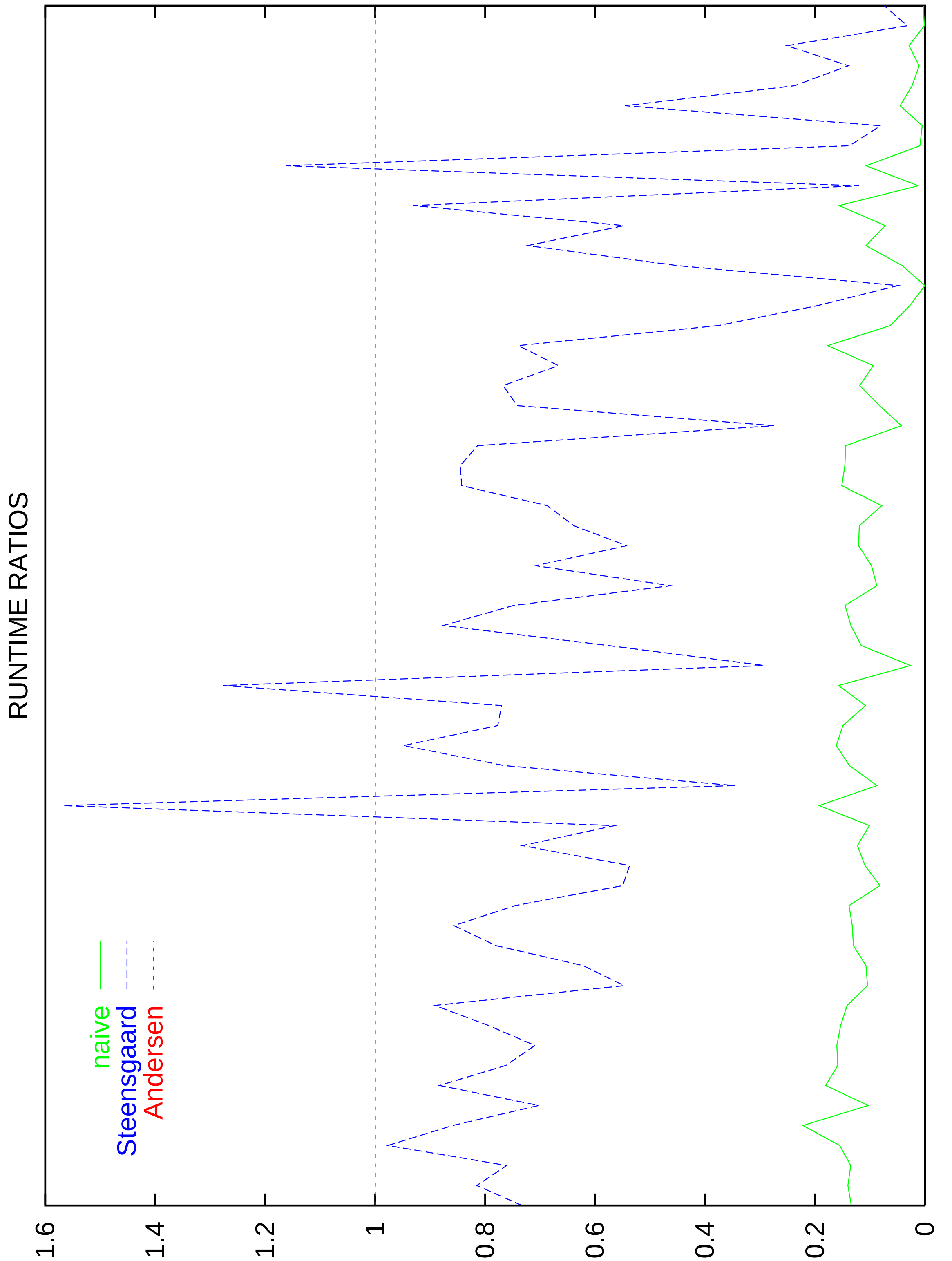
  average: 1.5X

  worst: 31X

- Both much better than naive


AVERAGE POINTS-TO SET SIZE RATIOS
naive
Steensgaard
Andersen
0
10
20
30
40
50
60
70
80


AVERAGE POINTS-TO SET SIZE RATIOS
Steensgaard
Andersen
16
14
12
10
8
6
4
2
0


RUNTIME RATIOS
naive
Steensgaard
Andersen
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0

BOTTOM LINE

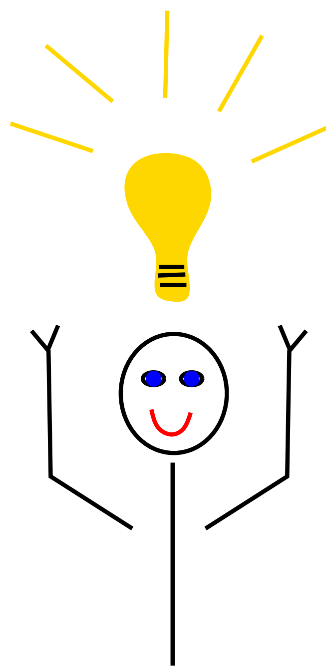
- USE ANDERSEN FOR SMALL PROGRAMS

- NEED ****NEW**** ALGORITHM FOR LARGE PROGRAMS

NEW ALGORITHM

ANDERSEN: max out-degree n precise but slow

STEENSGAARD: max out-degree 1 fast but imprecise



VARY THE MAX OUT−DEGREE !!

NEW ALG #1: MULTIPLE CATEGORIES

- Divide variables into k categories
- `x` points to both `v1` and `v2`:
  join ONLY if in same category
- k = 1 Steensgaard
  k = n Andersen
- Worst-case time: $O(k^2 n)$

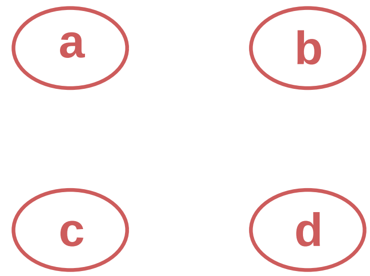
a = &b

a = &c

a = &d

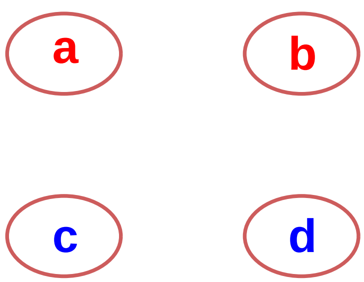
c = &d

1 CATEGORY (STEENSGAARD)



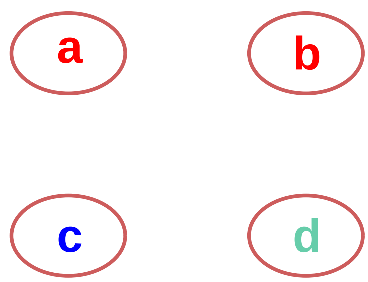
2 CATEGORIES

{ a, b }   { c, d }

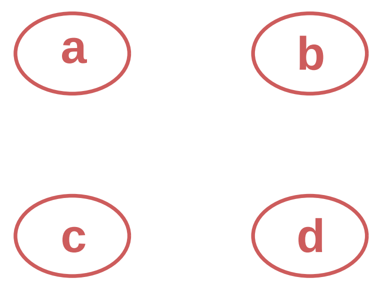


3 CATEGORIES

{ a, b }   { c }   { d }



4 CATEGORIES (ANDERSEN)



```
a = &b;
a = &c;
a = &d;
c = &d;
```

POINTS−TO SETS

	1 CAT (STEENS)	2 CAT	3 CAT	AND
a	{ b, c, d }	{ b, c, d }	{ b, c, d }	{ b, c, d }
b	{ b, c, d }	{ }	{ }	{ }
c	{ b, c, d }	{ c, d }	{ d }	{ d }
d	{ b, c, d }	{ c, d }	{ }	{ }

ANOTHER IDEA

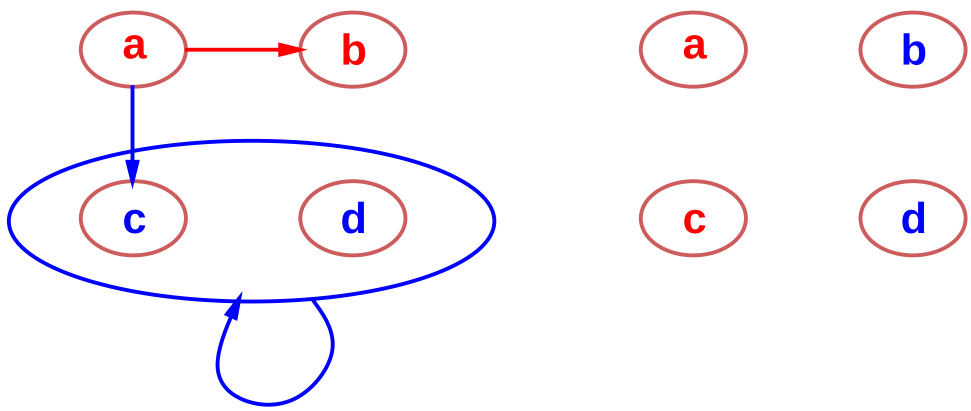
a = &b
a = &c
a = &d
c = &d

2 CATEGORIES #1

{ a, b } **{ c, d }**

2 CATEGORIES #2

{ a, c } **{ b, d }**



POINTS−TO SETS

	2 CAT #1	2 CAT #2	#1 ∩ #2	AND
a	{ b, c, d }	{ b, c, d }		{ b, c, d }
b	{ }	{ }		
c	{ c, d }	{ b, d }		{ d }
d	{ c, d }	{ }		

NEW ALG #2: MULTIPLE CATEGORIES, MULTIPLE RUNS

- Run algorithm #1 N times
- Use different categories each time
- Intersect points-to sets
- Time (k categories):

$$N * (\text{Alg \#1 time}) = O(N\,k^2\,n)$$

QUESTIONS

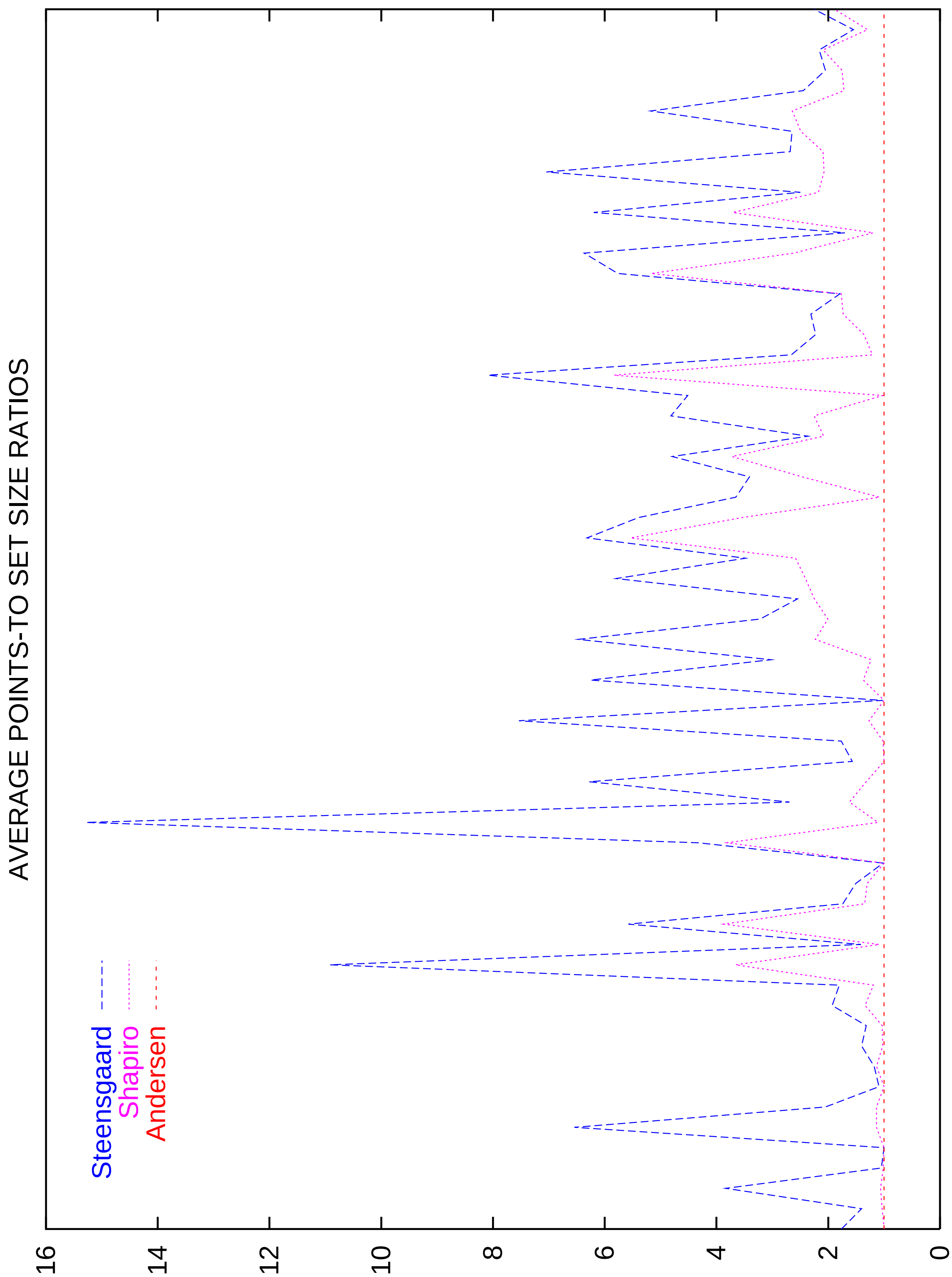
- How to choose categories?

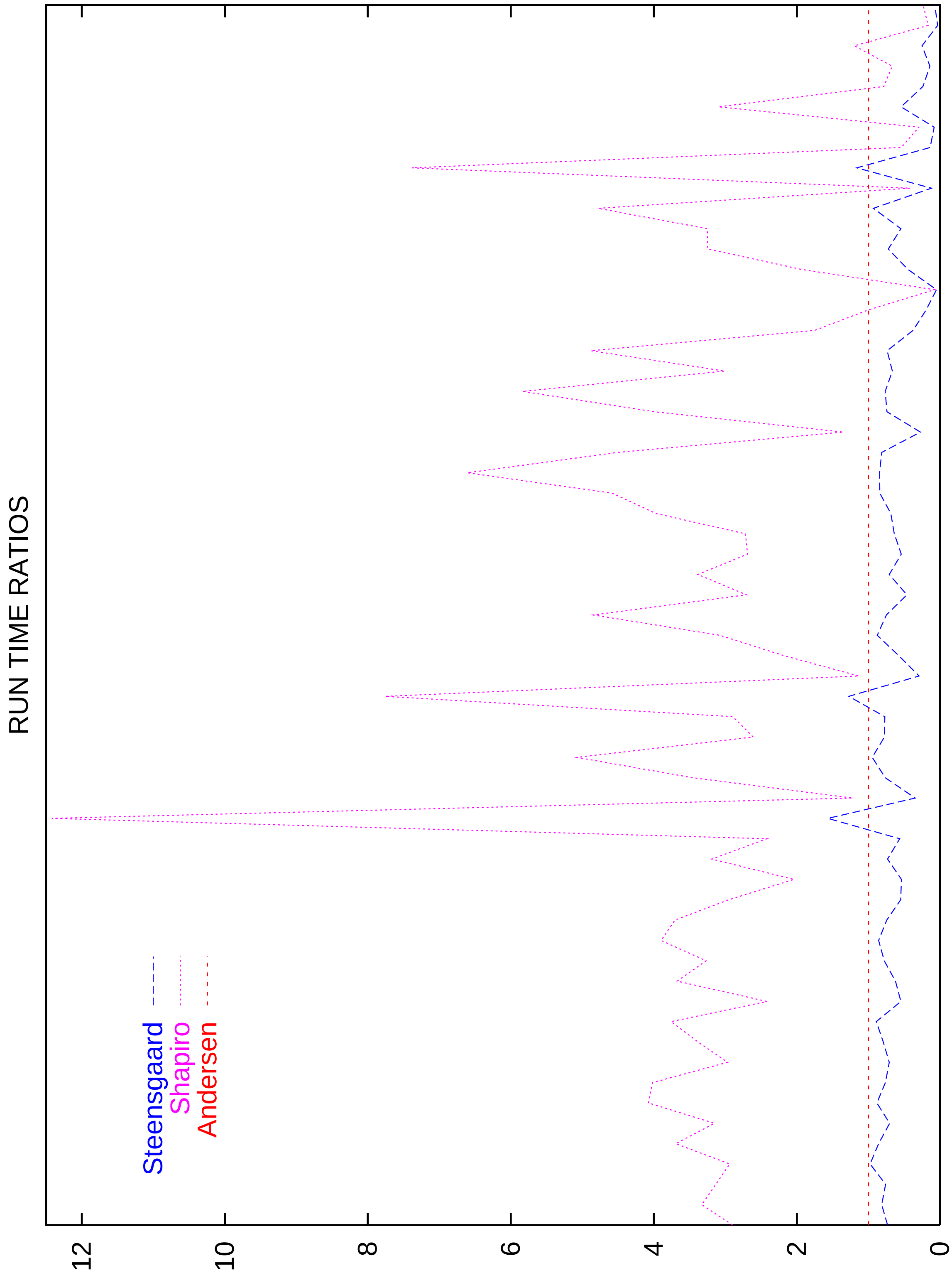
- How to choose number of runs?

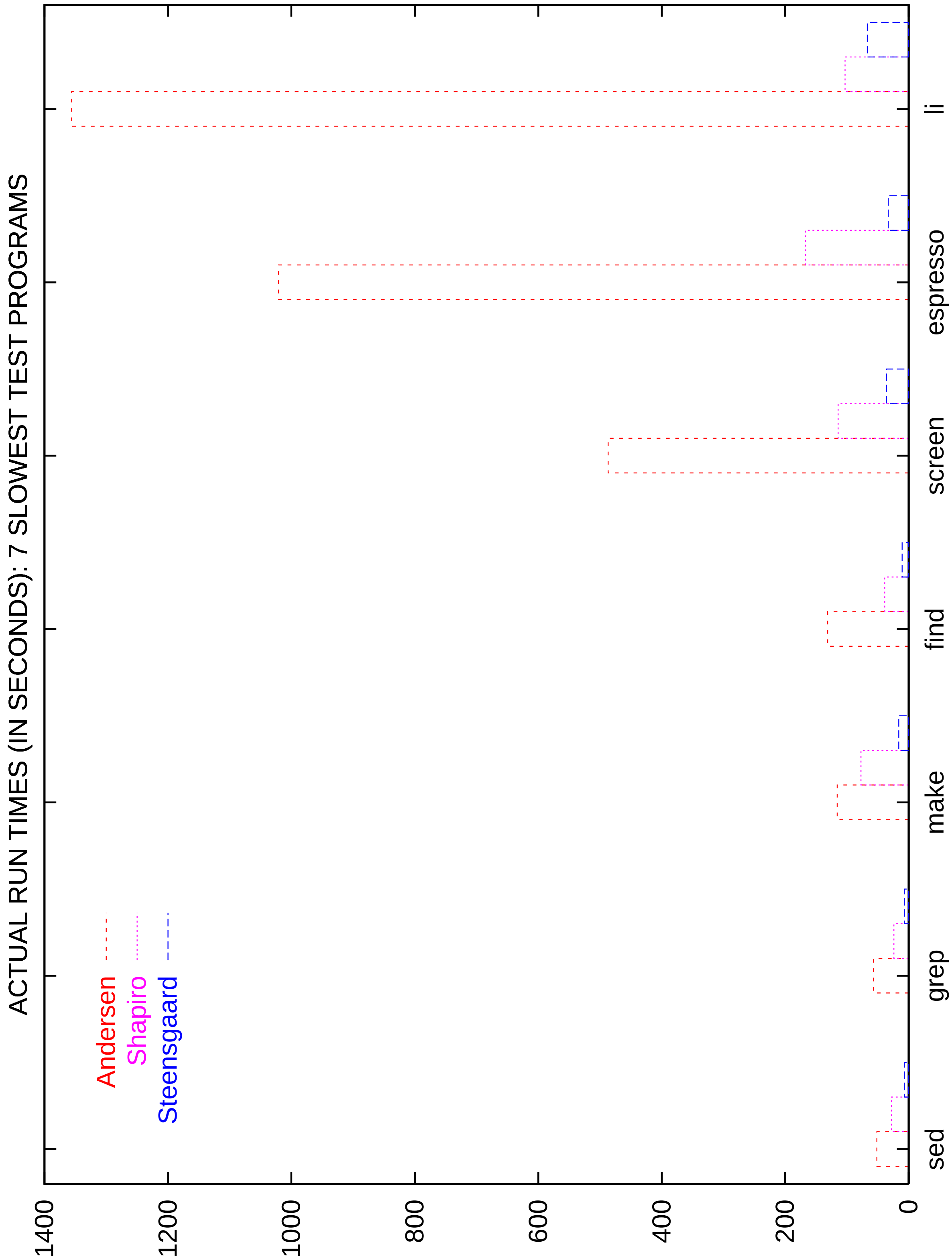
GOAL: “SEPARATE” ALL VARIABLE PAIRS

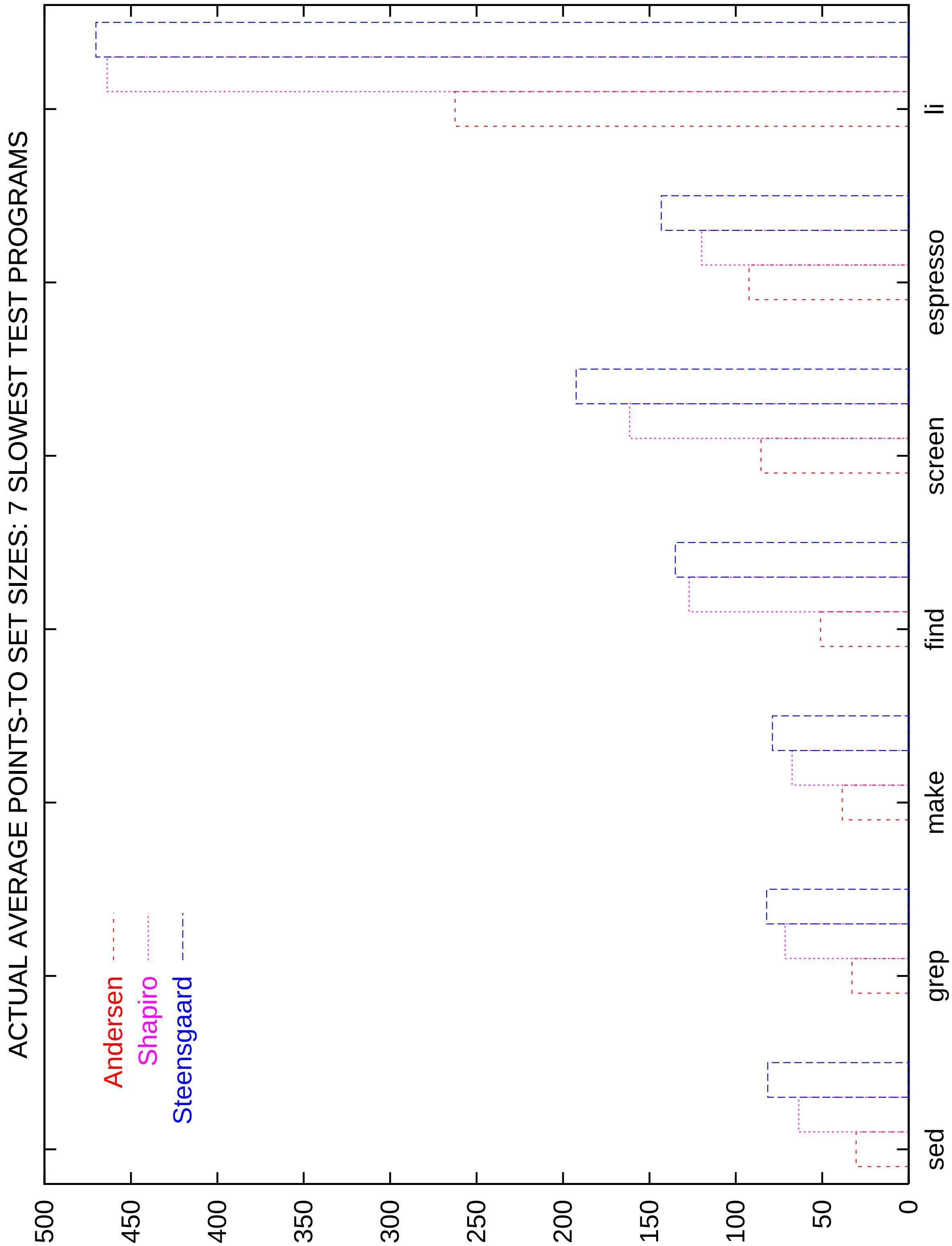
- Choose *number* of categories, k
- Write ID numbers for variables in base k
- On run r, use r^{th} digit as category
- Number of runs = $\log_k n$
- Time: $O(\log_k n \; k^2 \; n)$

`a`	0	0
`b`	0	1
`c`	1	0
`d`	1	1


AVERAGE POINTS-TO SET SIZE RATIOS
Steensgaard
Shapiro
Andersen
0
2
4
6
8
10
12
14
16


RUN TIME RATIOS
Steensgaard
Shapiro
Andersen
0
2
4
6
8
10
12


ACTUAL RUN TIMES (IN SECONDS): 7 SLOWEST TEST PROGRAMS
Andersen
Shapiro
Steensgaard
1400
1200
1000
800
600
400
200
0
sed
grep
make
find
screen
espresso
li


ACTUAL AVERAGE POINTS-TO SET SIZES: 7 SLOWEST TEST PROGRAMS
Andersen
Shapiro
Steensgaard
sed
grep
make
find
screen
espresso
li
0
50
100
150
200
250
300
350
400
450
500

SUMMARY

- ANDERSEN: O(n^3)

  precise but sometimes too slow

- STEENSGAARD: O(n)

  fast but sometimes too imprecise

- SHAPIRO: O($n \log n$)

  not quite linear

  intermediate precision

  can be improved?

NEW QUESTIONS:

- Does more precise pointer analysis lead to more precise *subsequent* analyses?
- What are the time trade-offs?

ANSWERS:

- Yes.
- Better points-to info faster subsequent analysis.

TRANSITIVE EFFECTS EXPERIMENTS

Step 1: Use 4 pointer analyses to identify use/def sets in CFG.

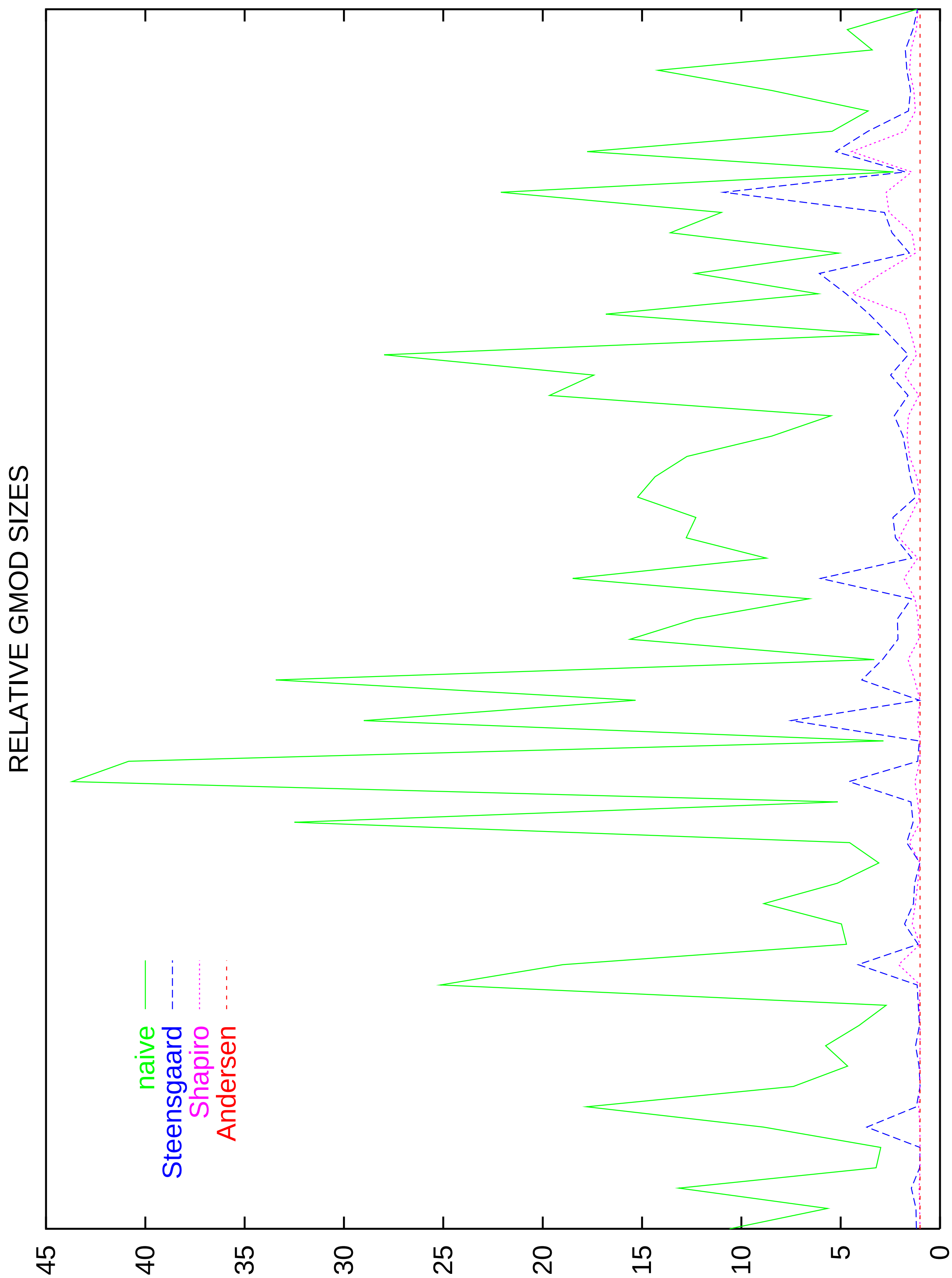
Step 2: Do dataflow analysis using each CFG.

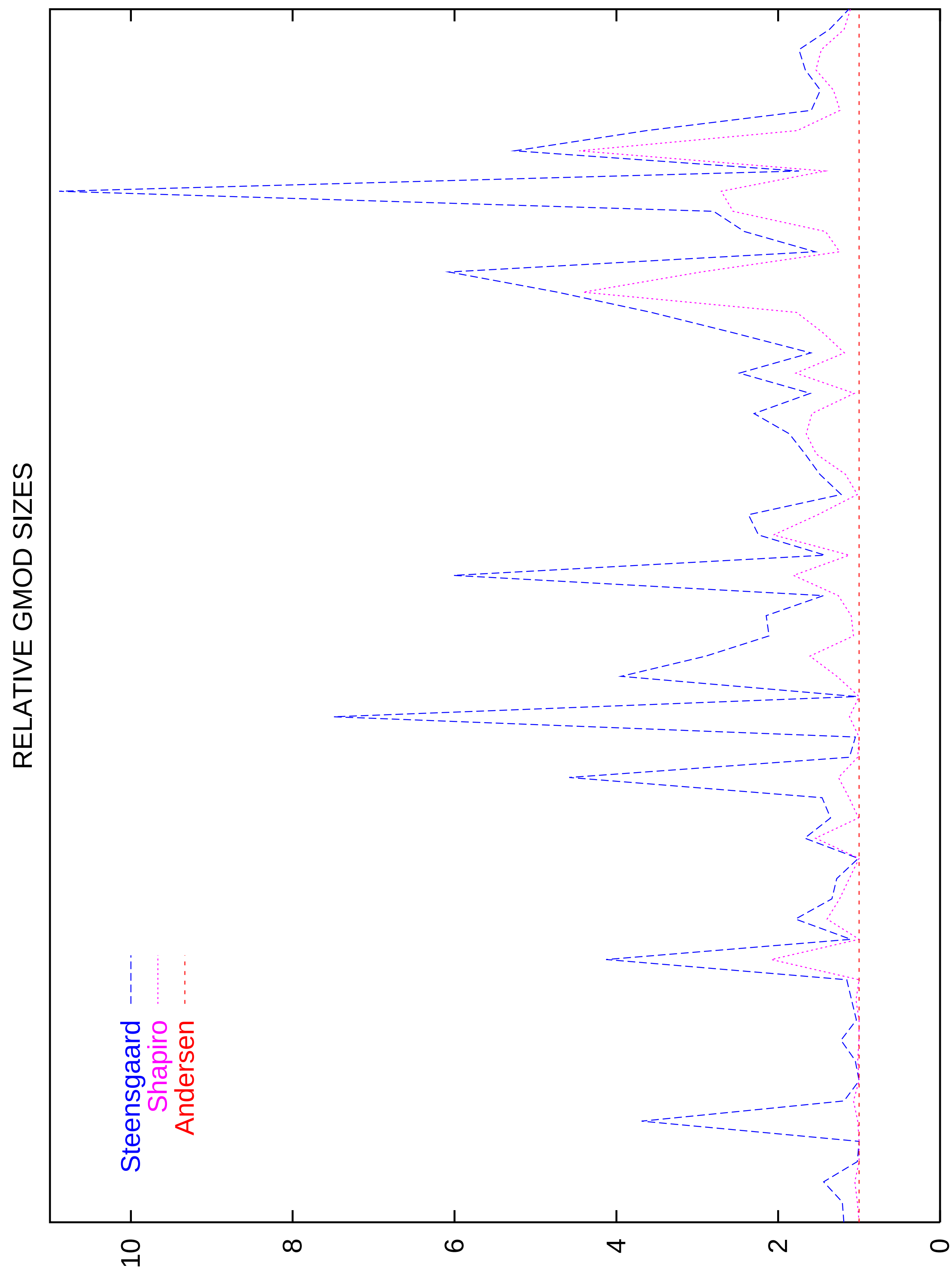
Step 3: Measure sizes of dataflow results.

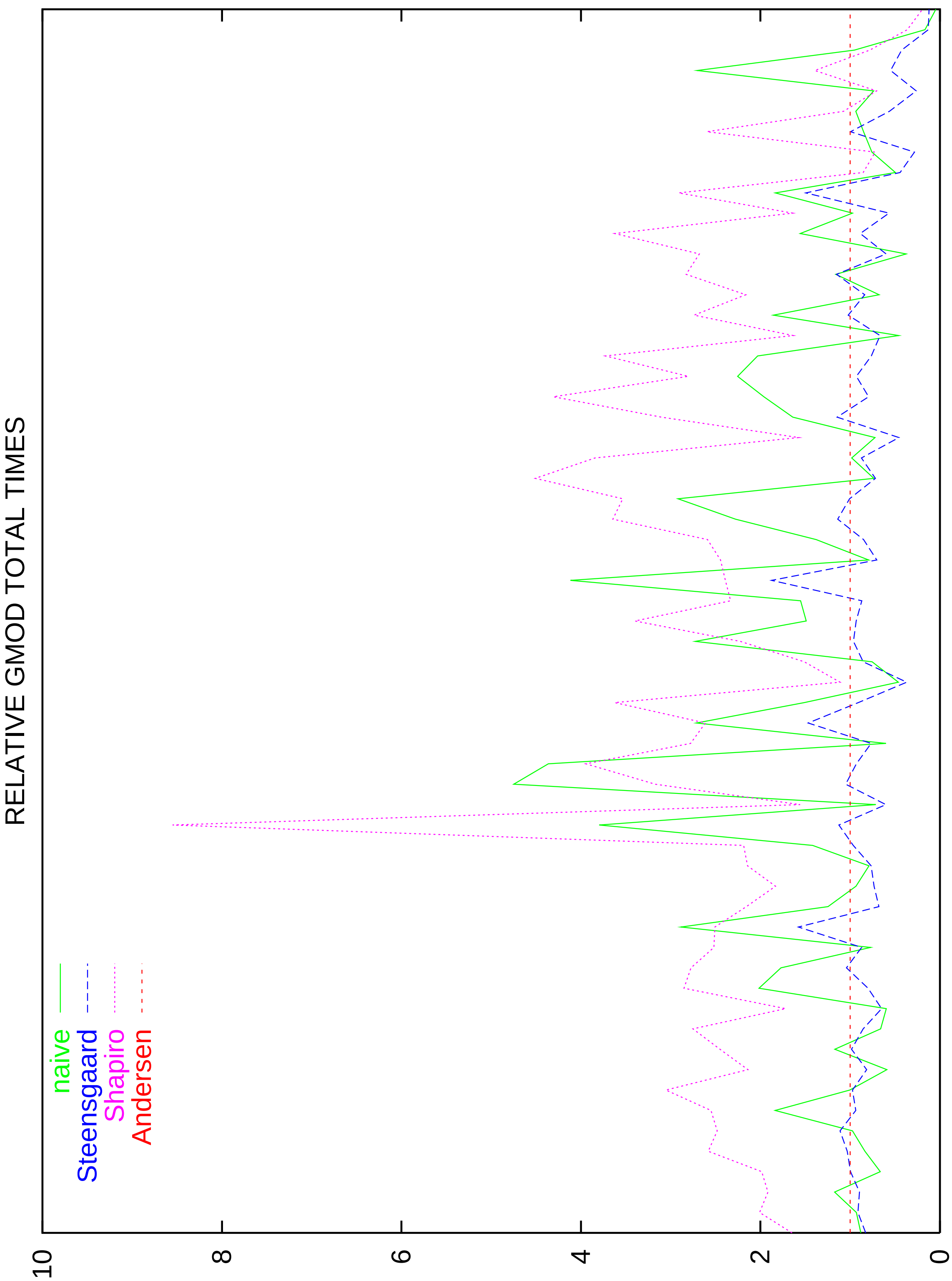
Step 4: Measure times for steps 1 and 2.

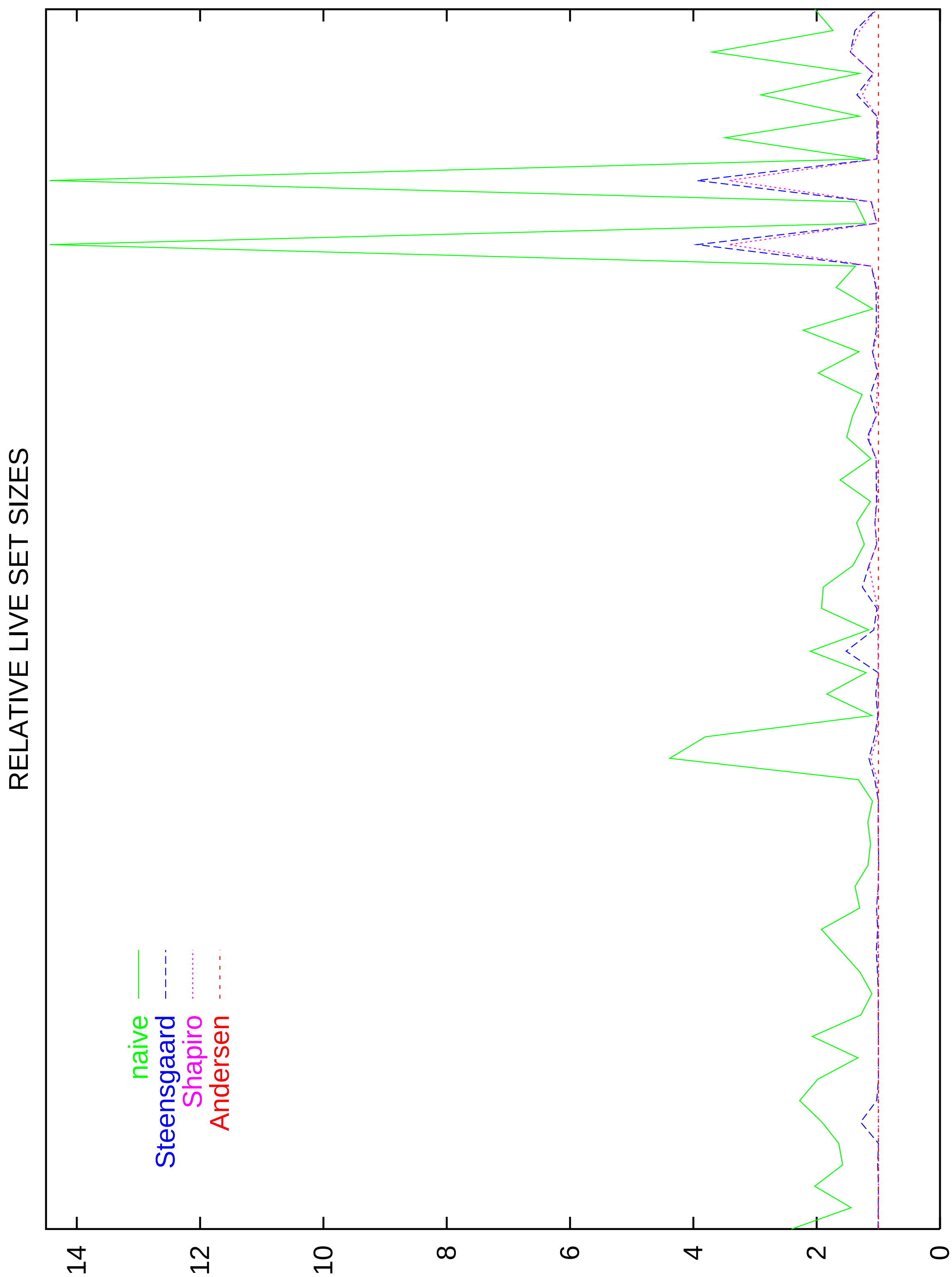
TRANSITIVE EFFECTS: DATAFLOW PROBLEMS

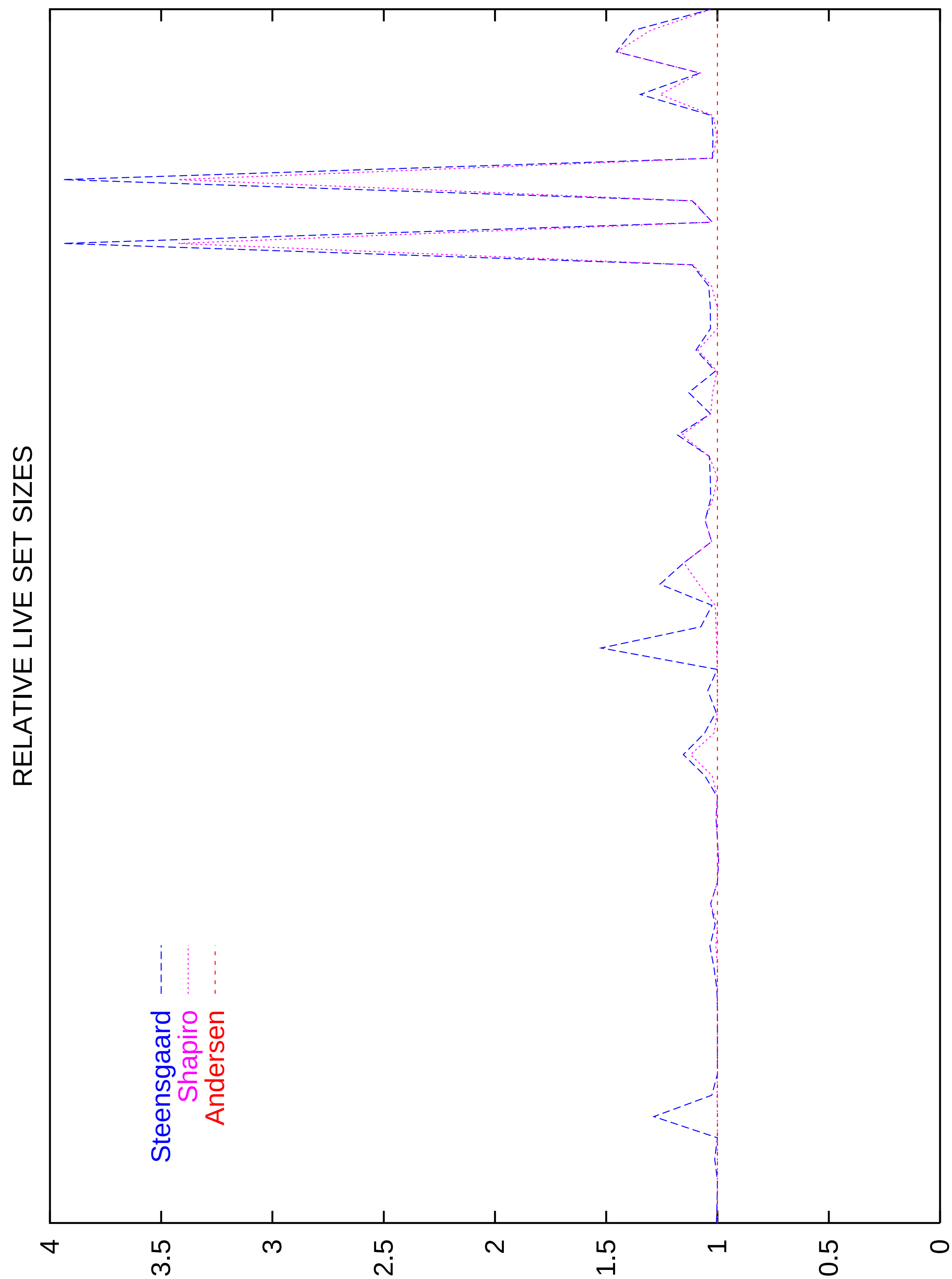
- GMOD
- Live Variables
- Truly Live Variables

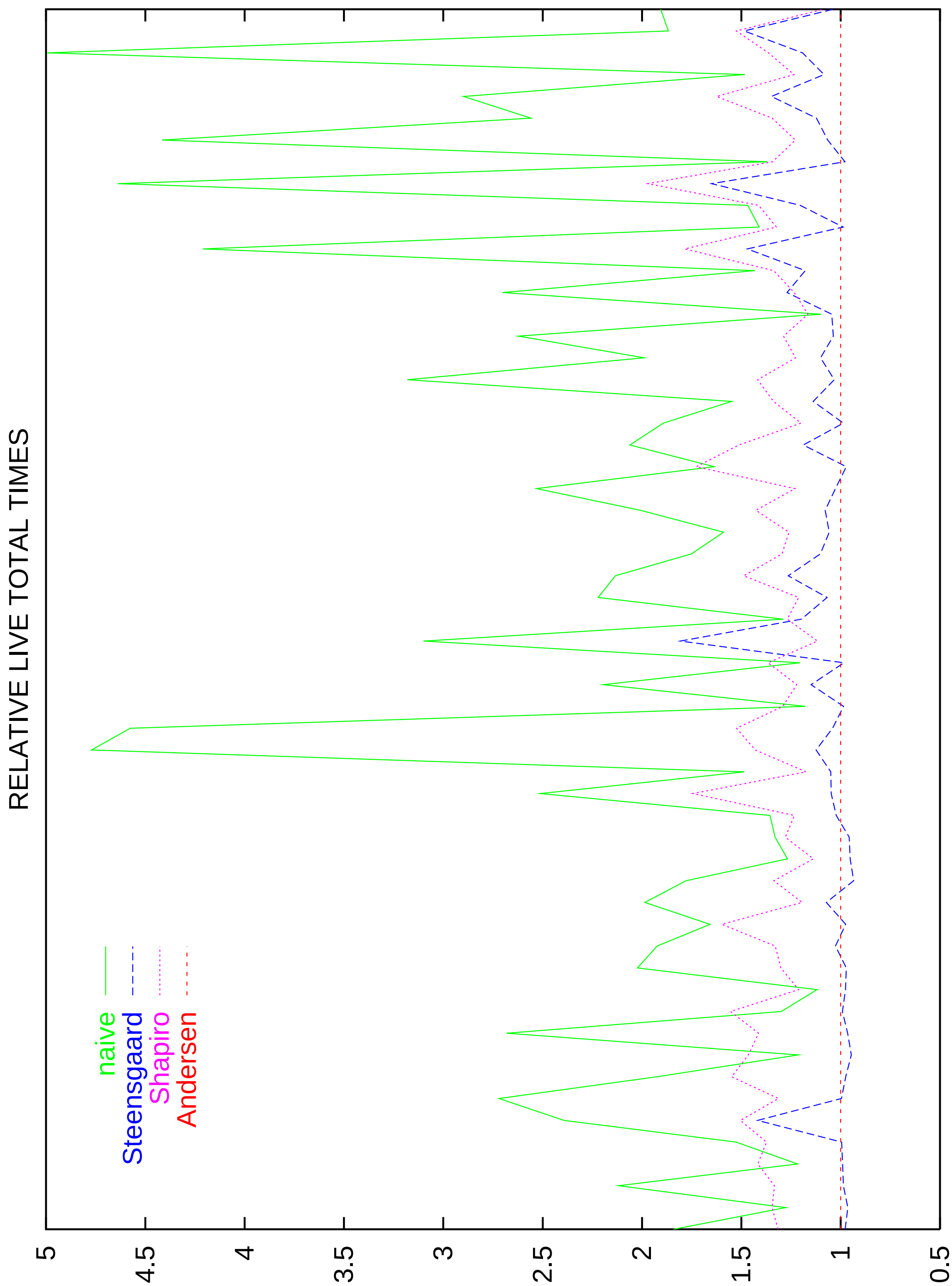

RELATIVE GMOD SIZES
naive
Steensgaard
Shapiro
Andersen
45
40
35
30
25
20
15
10
5
0

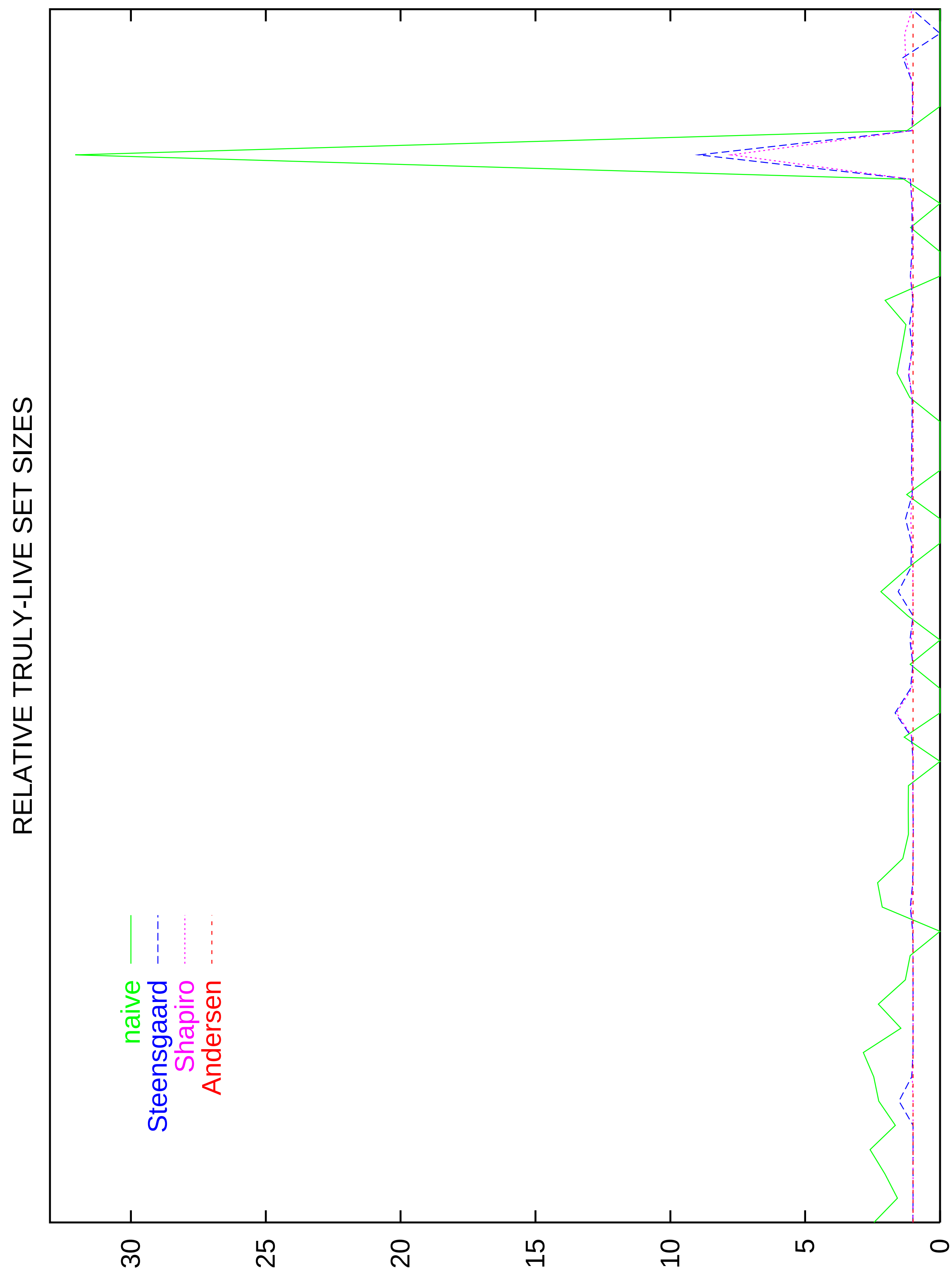

RELATIVE GMOD SIZES
Steensgaard
Shapiro
Andersen
0
2
4
6
8
10


RELATIVE GMOD TOTAL TIMES
naive
Steensgaard
Shapiro
Andersen
10
8
6
4
2
0

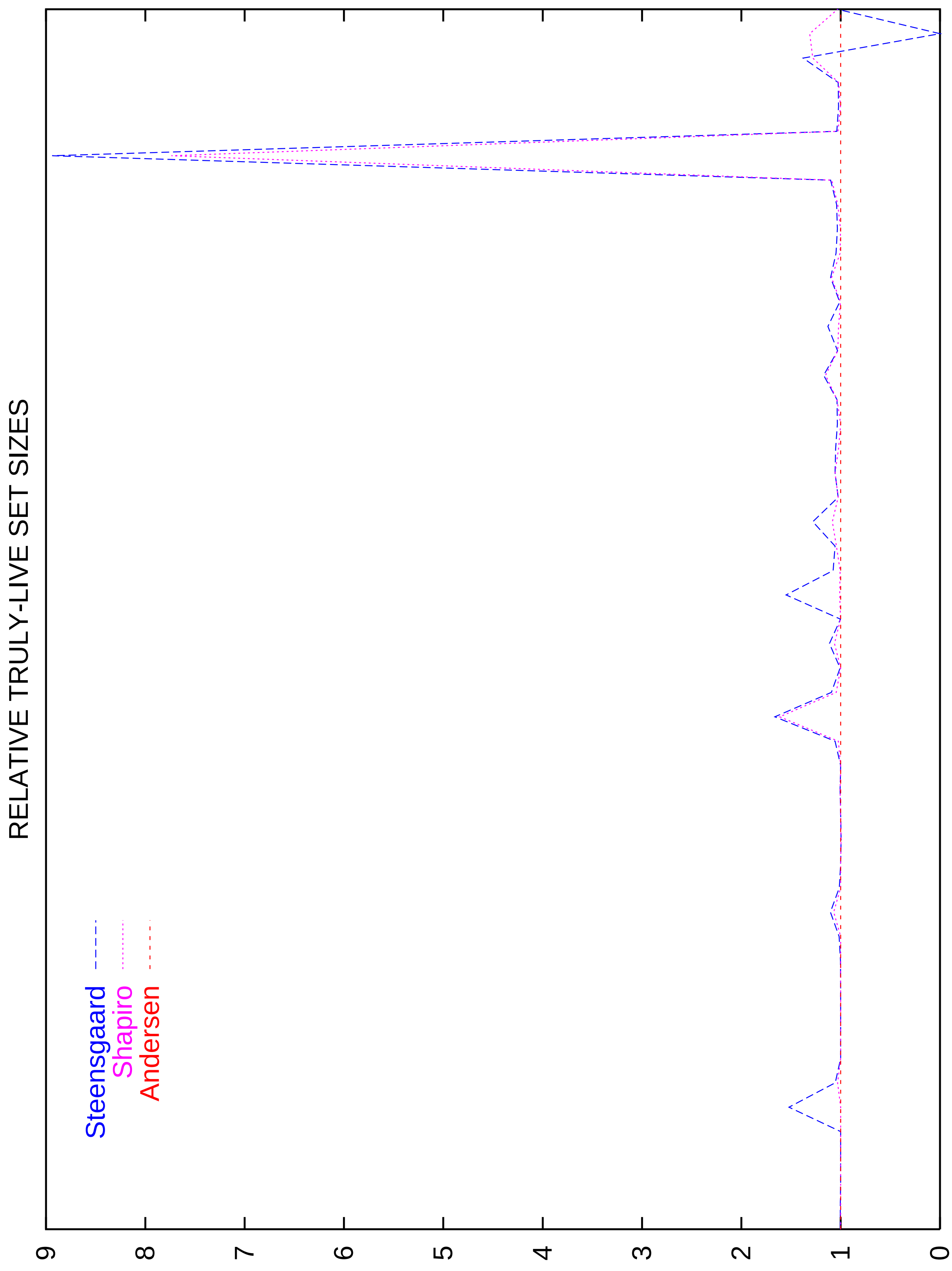

RELATIVE LIVE SET SIZES
naive
Steensgaard
Shapiro
Andersen
0
2
4
6
8
10
12
14

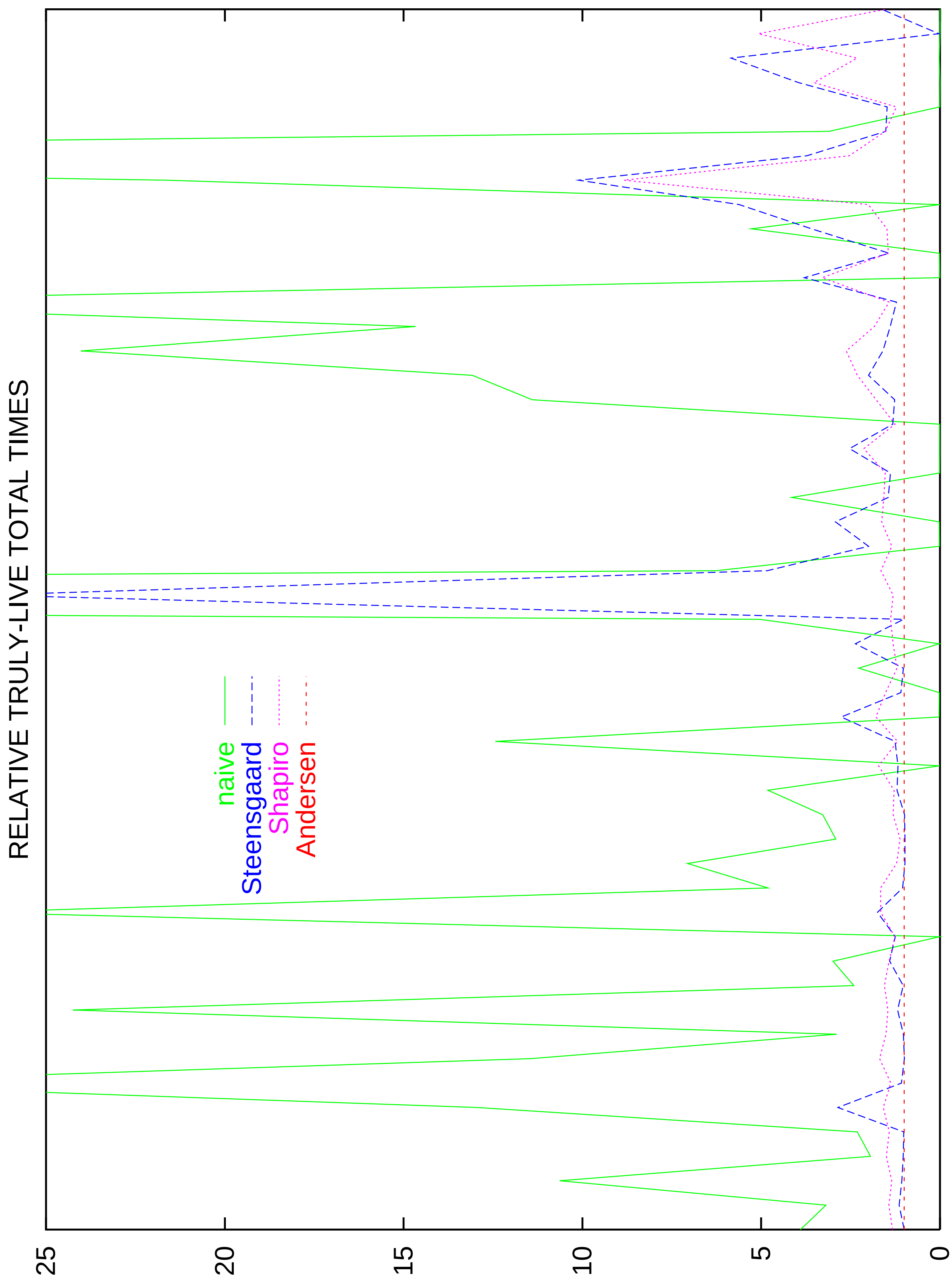

RELATIVE LIVE SET SIZES
Steensgaard
Shapiro
Andersen
4
3.5
3
2.5
2
1.5
1
0.5
0


RELATIVE LIVE TOTAL TIMES
naive
Steensgaard
Shapiro
Andersen
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5



RELATIVE TRULY-LIVE SET SIZES
naive
Steensgaard
Shapiro
Andersen
0
5
10
15
20
25
30


RELATIVE TRULY-LIVE SET SIZES
Steensgaard
Shapiro
Andersen
0
1
2
3
4
5
6
7
8
9


RELATIVE TRULY-LIVE TOTAL TIMES
naive
Steensgaard
Shapiro
Andersen
25
20
15
10
5
0

TRANSITIVE EFFECTS SUMMARY

- MORE PRECISE ANALYSIS
  MORE PRECISE DATAFLOW INFO

- FOR HARD DATAFLOW PROBLEMS:

  MORE PRECISE ANALYSIS

  LESS TOTAL TIME

FUTURE WORK

- Speed up Shapiro’s algorithm
- Improve precision of Shapiro’s algorithm
- Comparisons with flow-*sensitive* analyses
- Predicting best algorithm