Dependence Clusters

Dave Binkley

Loyola University Maryland

What’s Coming

- Dependence Defined
- Dependence Clusters
- Motivation
- Finding Dependence Clusters and Causes
- Finding Causes FAST!

Dependence

I’ll go if you go

Dependence Cluster

I'll go if you go

I'll go if you go

I'll go if you go

Thus you get
all or nothing!

Dependence

```
main()
{
   a = 42;

   if a > 10

      b = a / 2
}
```

Data Dependence (definition – use)

Control Dependence

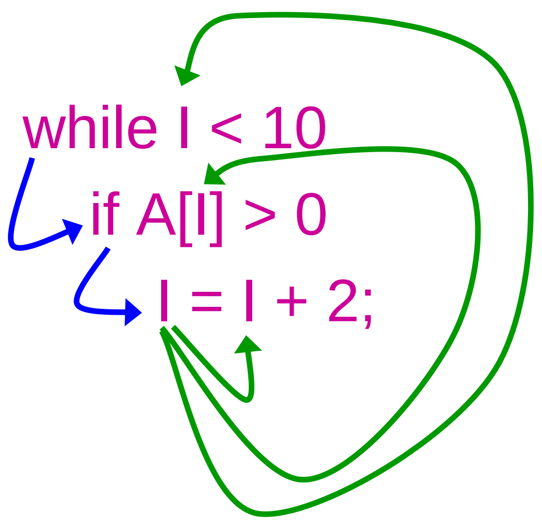
Dependence Cluster

- main()
- {
-     while I < 10
-         if **A[I] > 0**
-             I = I + 2;
- }

Dependence Cluster

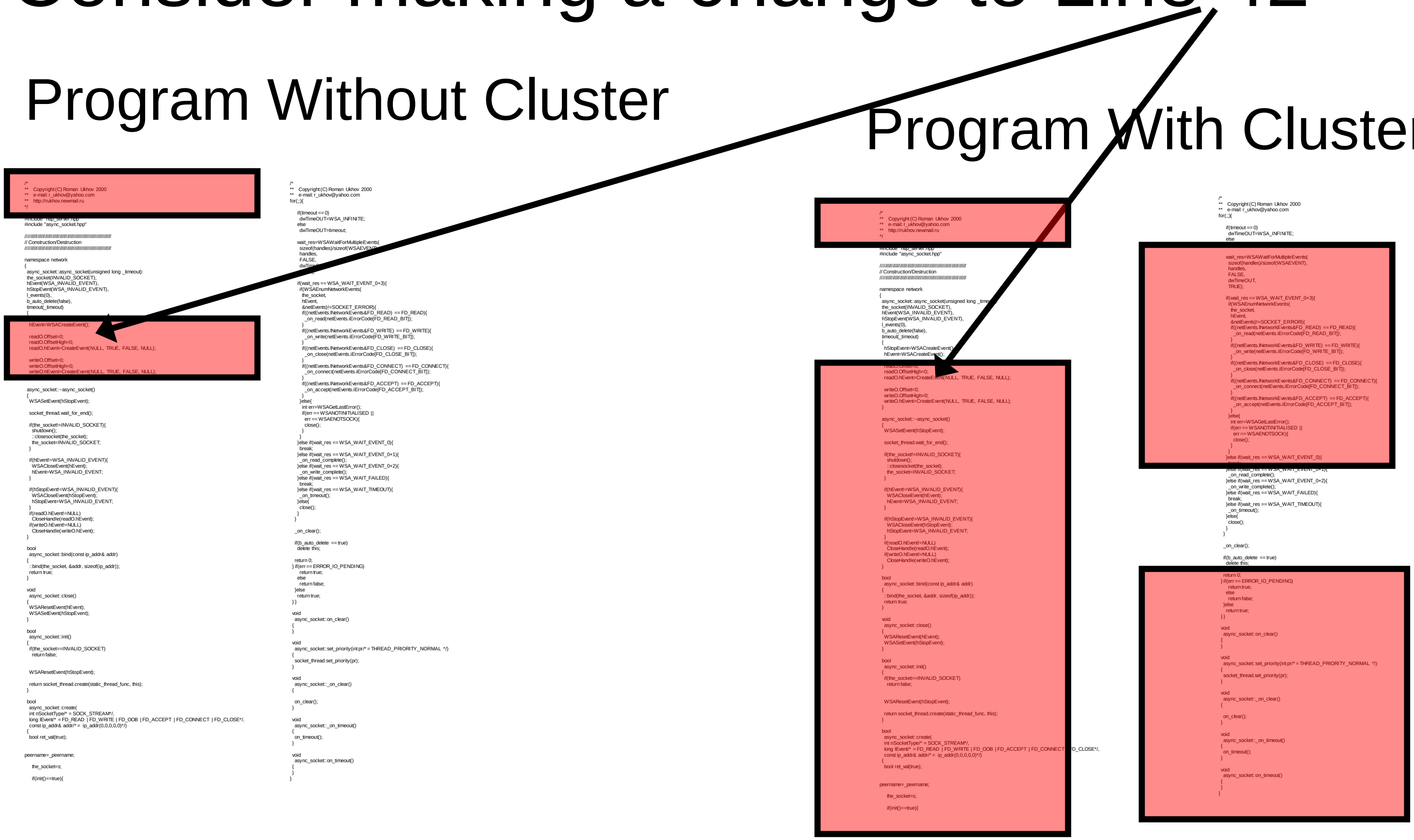
A set of statements where each (transitively) depends on the others

- main()
- {
- while I < 10
- if A[I] > 0
- I = I + 2;
- }



Impact of Dependence Clusters

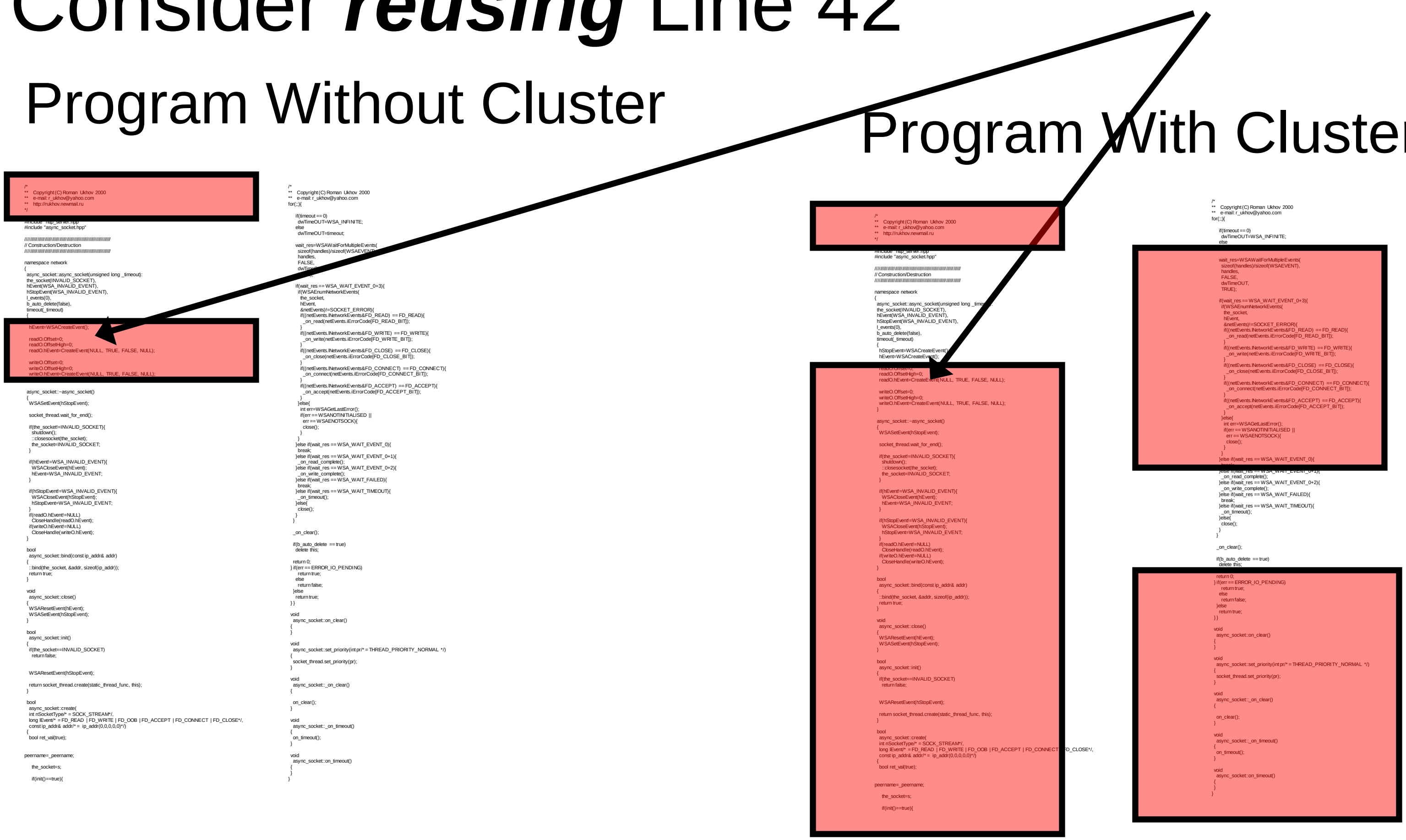
Consider making a change to Line 42



Impact of Dependence Clusters

Consider ***testing*** Line 42

Consider ***reusing*** Line 42



Finding Dependence Clusters

- *Definition: A set of statements where each depends on the others*

- Algorithm 1

  statements ‘s’ and ‘t’ are in a cluster if

  slice(s) = slice(t)

Finding Dependence Clusters

For statement ‘s’ of program ‘P’

$$\textbf{\textit{Cluster(s)}} = \{ t \in P \mid \text{slice}(t) = \text{slice}(s) \}$$

Note

$$t \in \text{Cluster}(s) \rightarrow \text{Cluster}(s) = \text{Cluster}(t)$$

thus, P’s clusters partition P’s statements

Finding Dependence Clusters

- Algorithm 1

    statements ‘s’ and ‘t’ are in a cluster if

    slice(s) = slice(t)

- Implementation 1
  - slice on each statement $O(n^2)$
  - compare the slices $O(n^3)$

Finding Dependence Clusters An Approximation

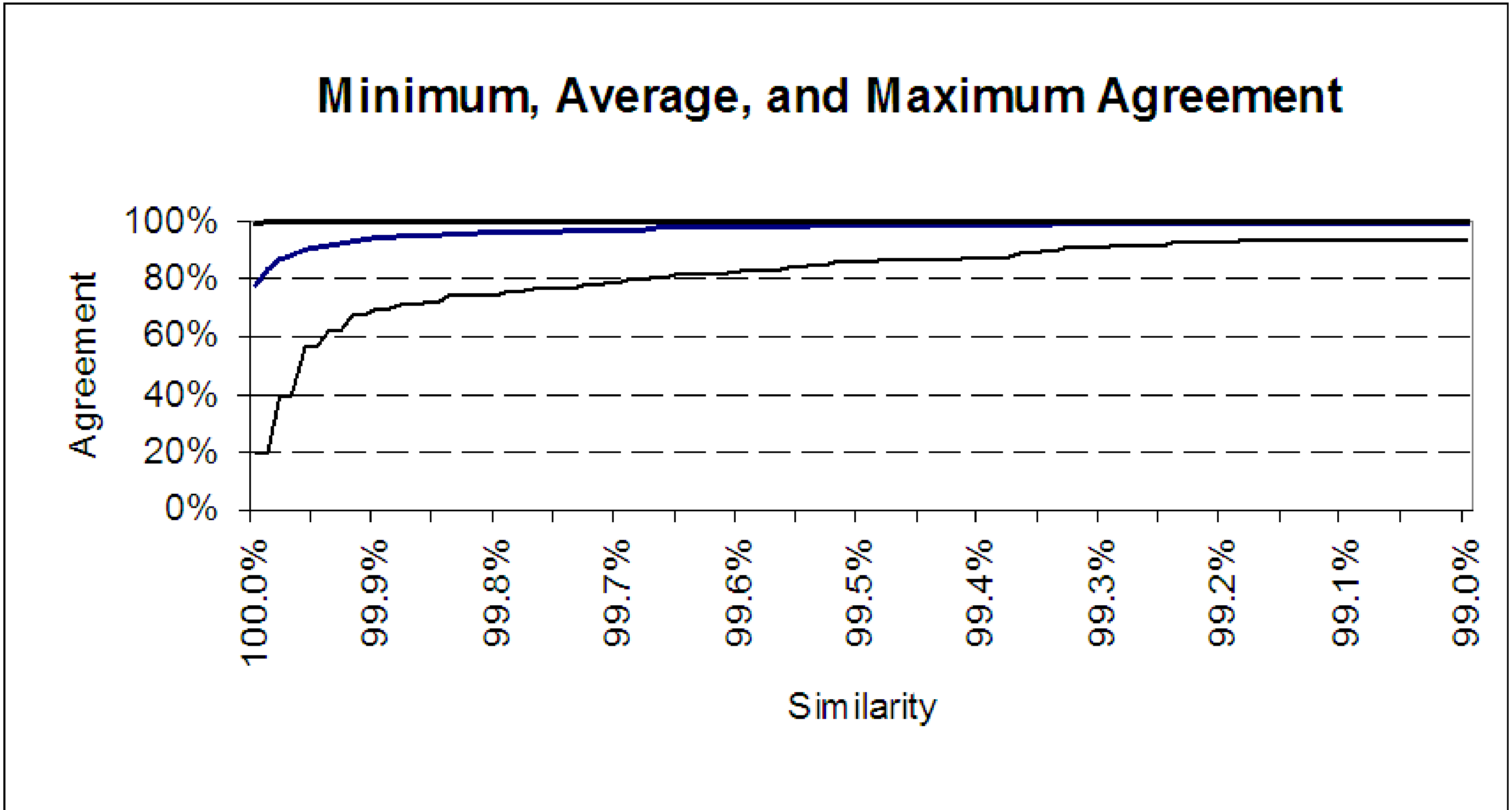
- Algorithm 1a

  statements ‘s’ and ‘t’ are in a cluster if

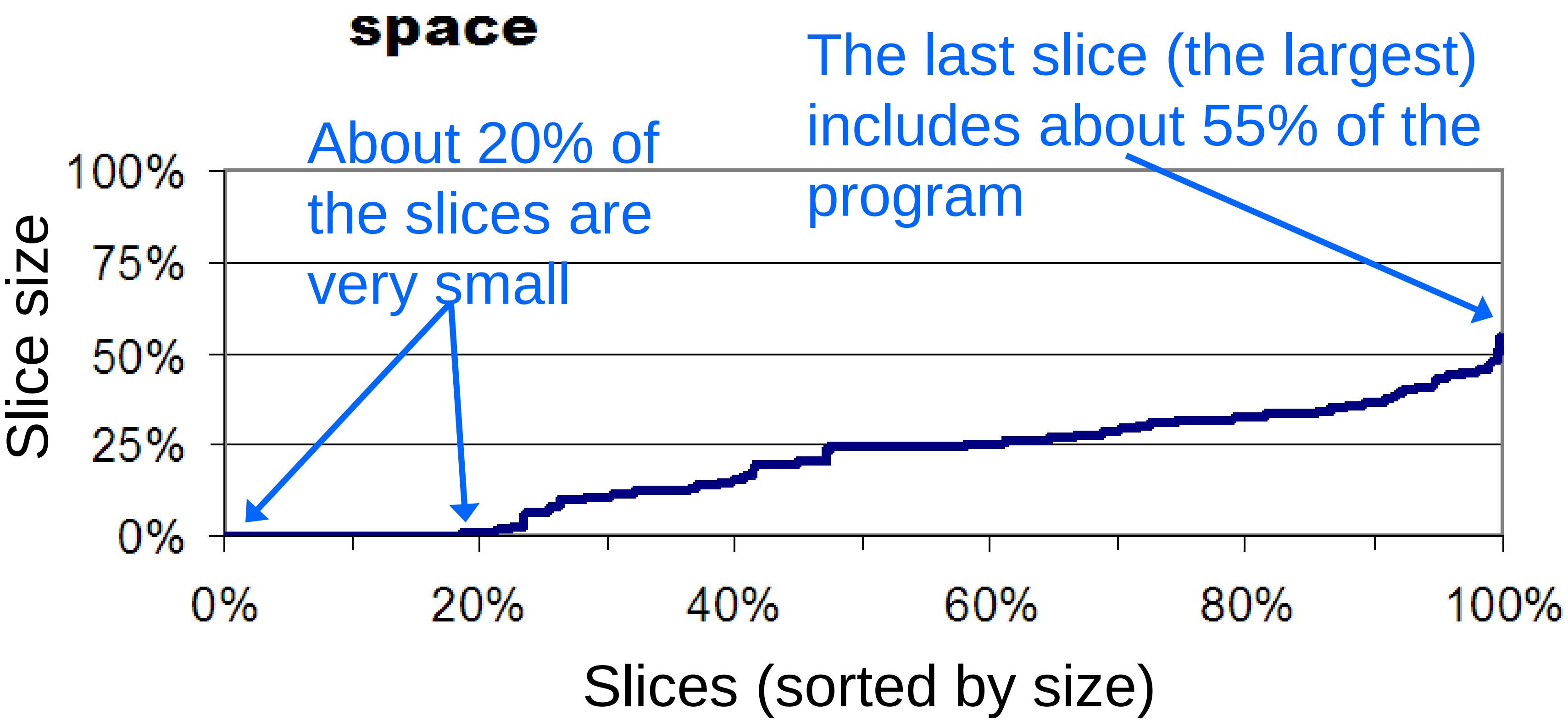
  slice-**size**(s) = slice-**size**(t)

- Implementation 1a
  - slice on each statement $O(n^2)$
  - partition based on size $O(n)$

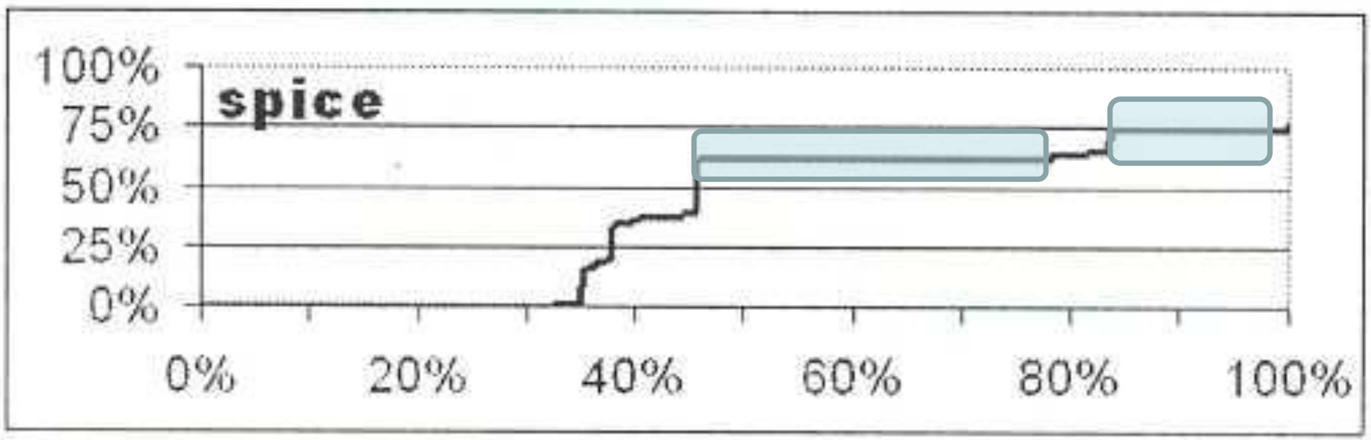
Is this a Good Approximation?



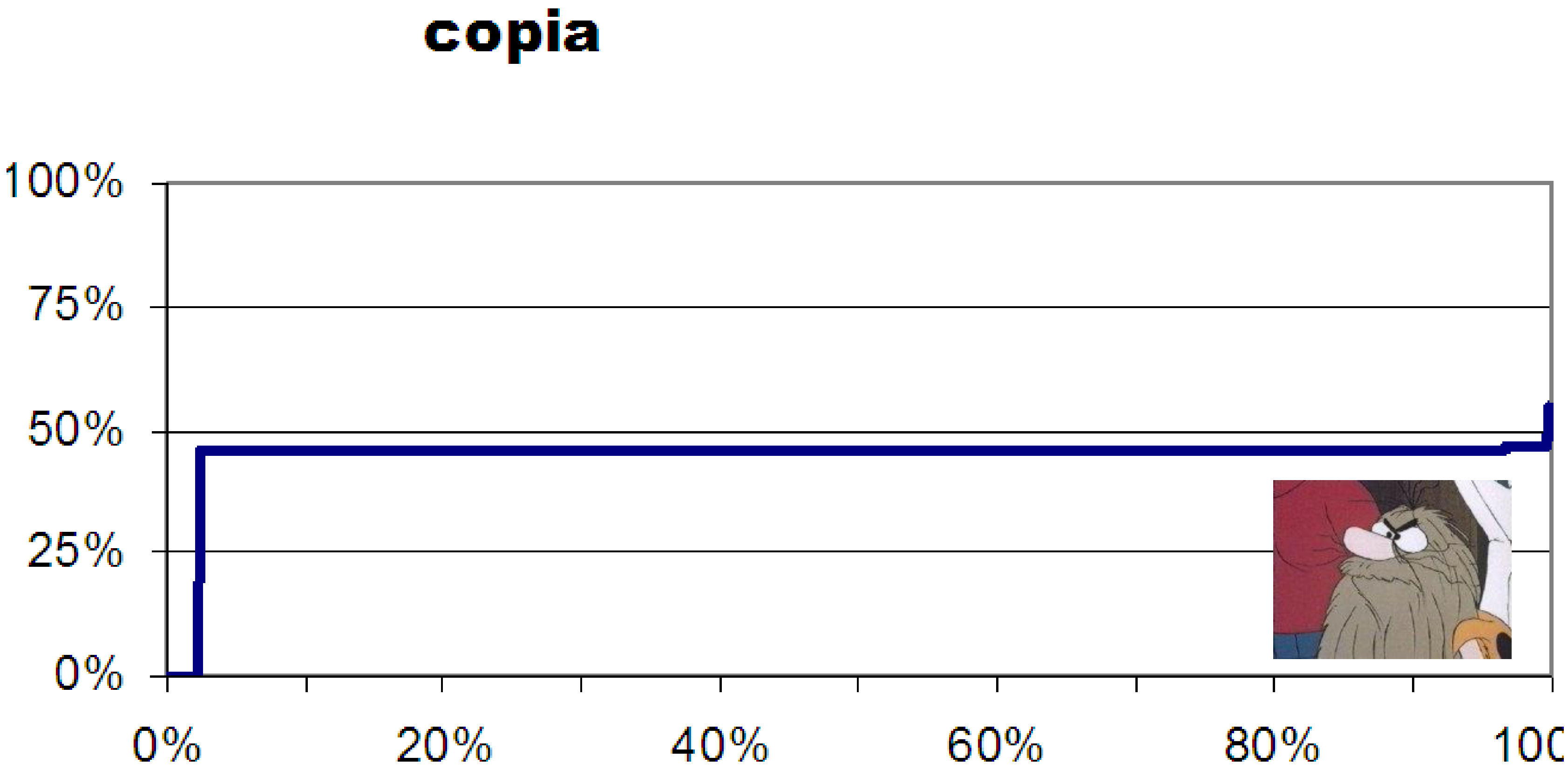
The Approximation Yields an Interesting Visualisation
the MSG
(the Monotone Slice-size Graph)



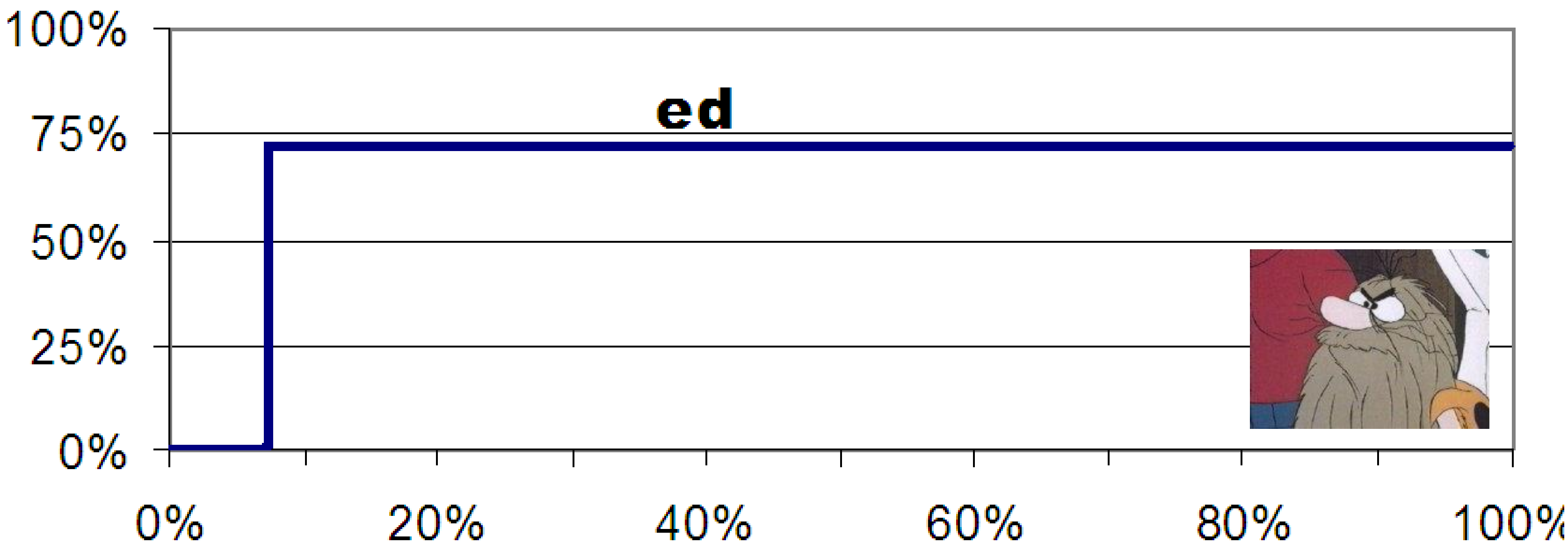
Cluster Containing MSG



And Another

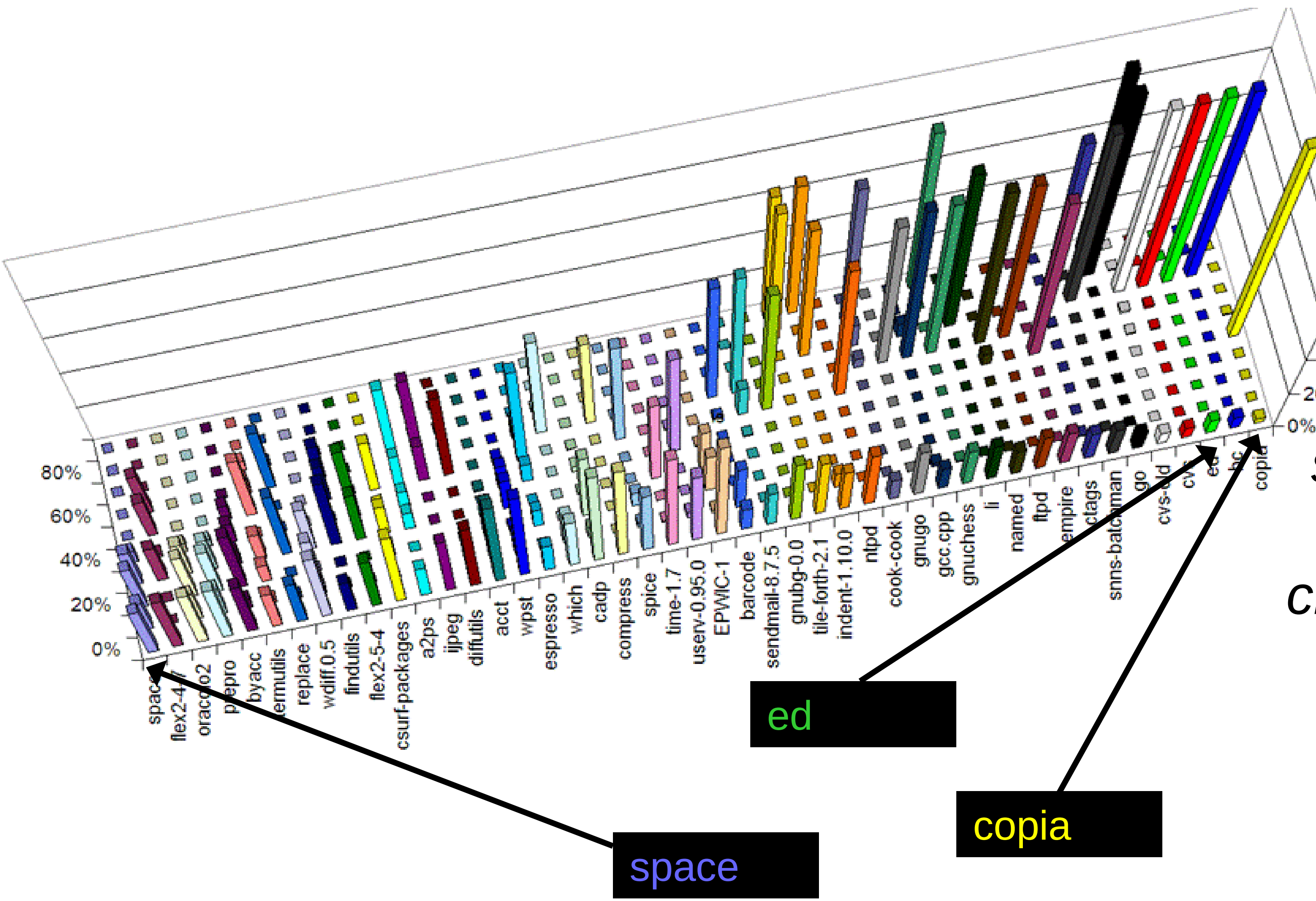


And Another



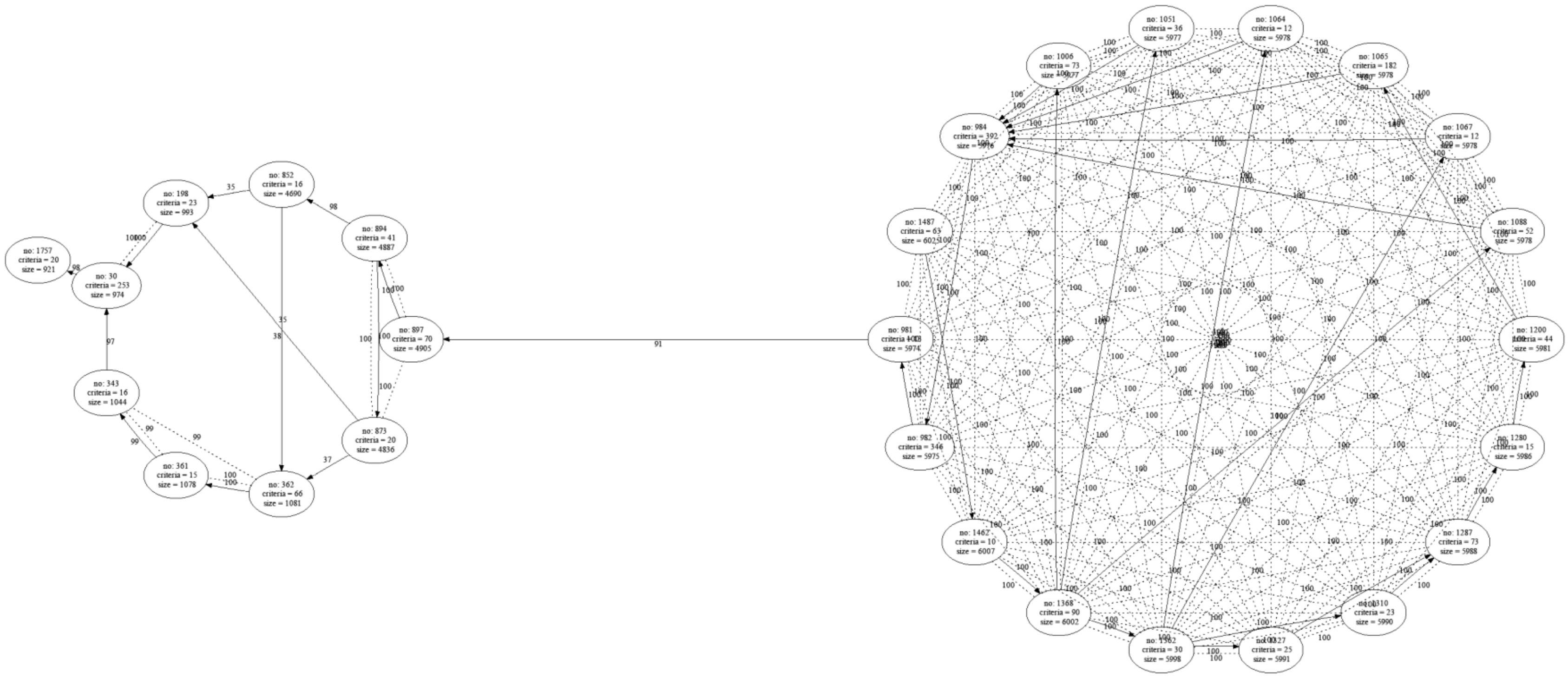
Do Clusters Exist? -- Yes

size of slices in cluster



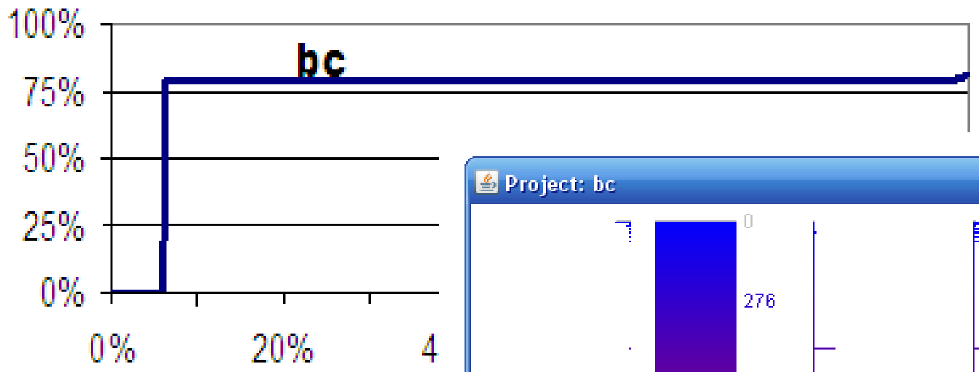
slices in cluster

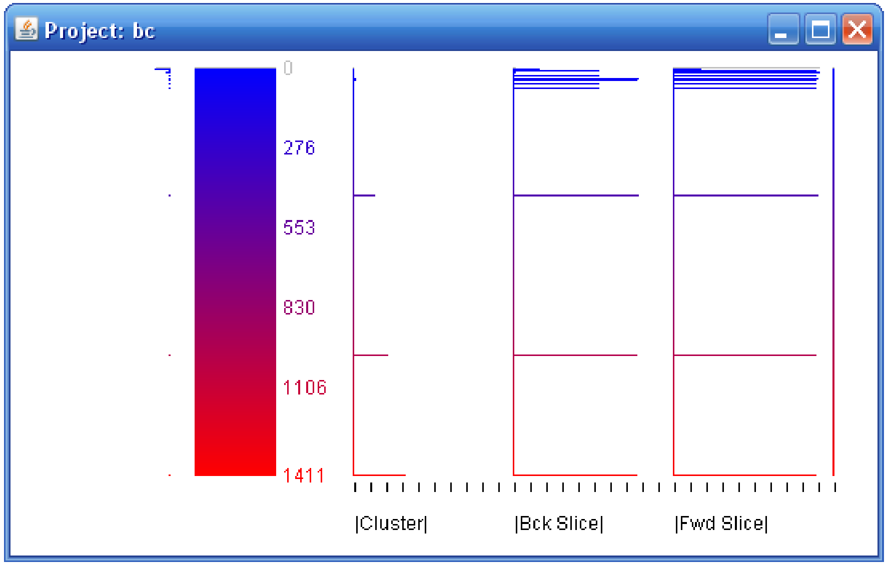
Visualizing Dependence Clusters

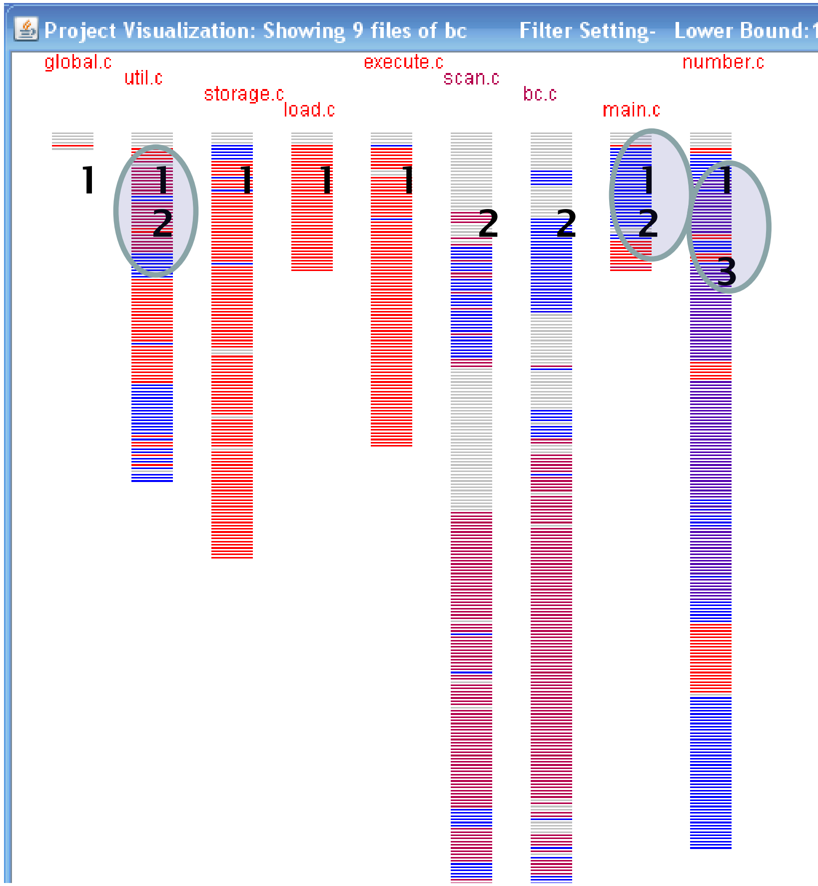


MSG for **bc** (basic calculator)

(Syed Islam)



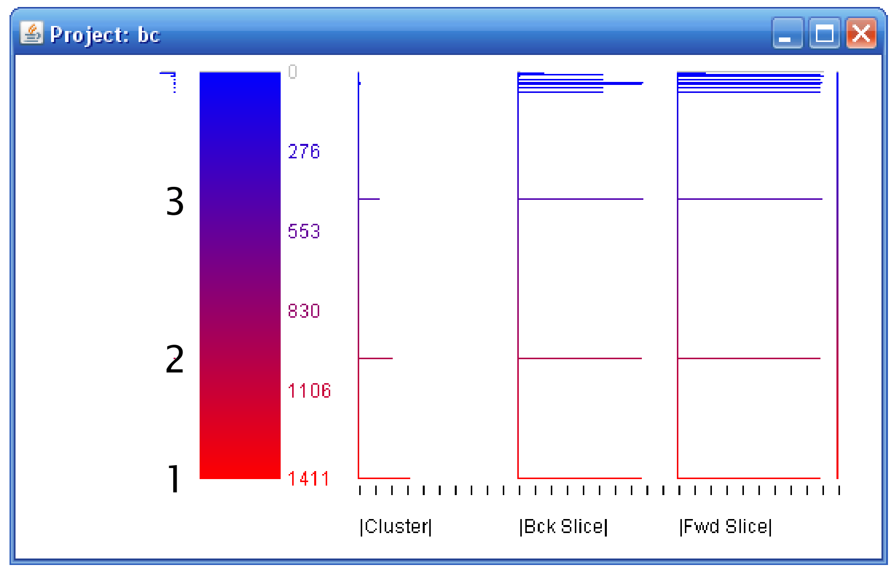


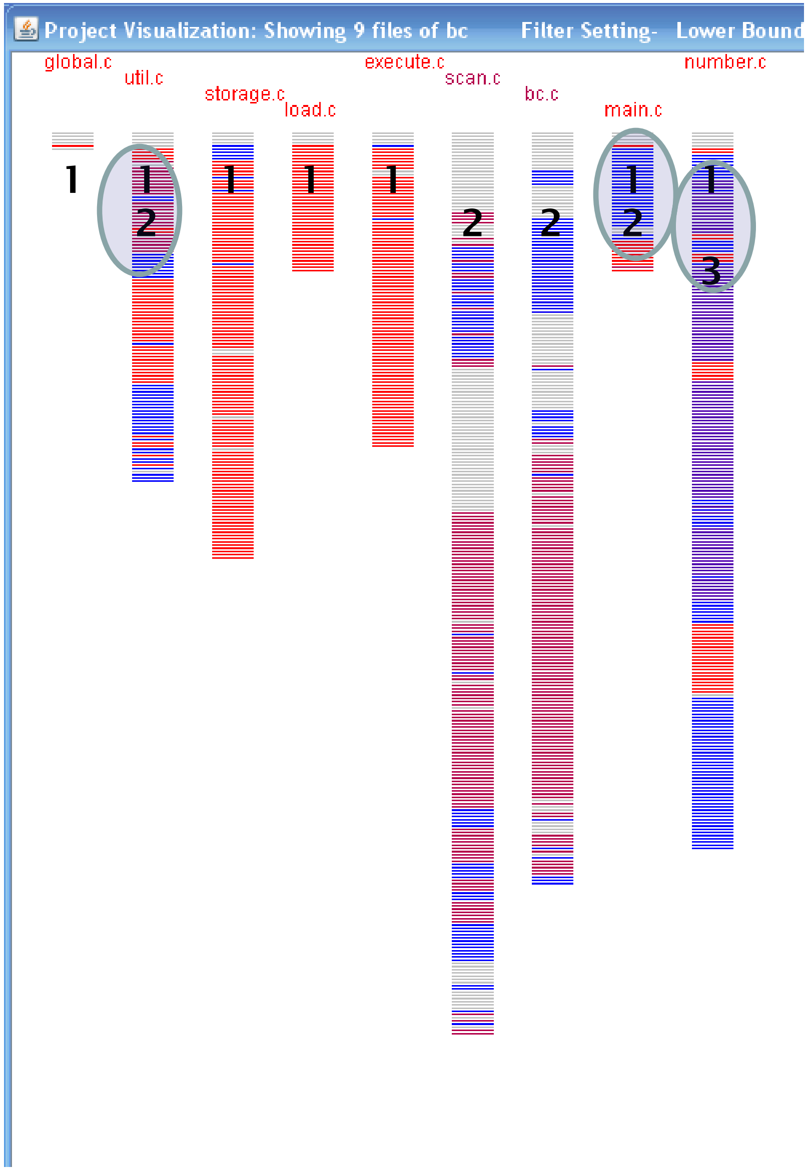


Cluster 2 covers 20% of the program. It consists of lexical analysis and parser functions found in ***main.c, scan.c***, ***bc.c*** and ***util.c****

Cluster 3 is a small isolated cluster spanning only one file (***number.c)***

Cluster 1 is the largest cluster covering 30% of the program. It contains the main functionality of the program and spans over all files except ***scan.c*** and ***bc.c*** . ***util.c****





main.c will almost always have combination of clusters.

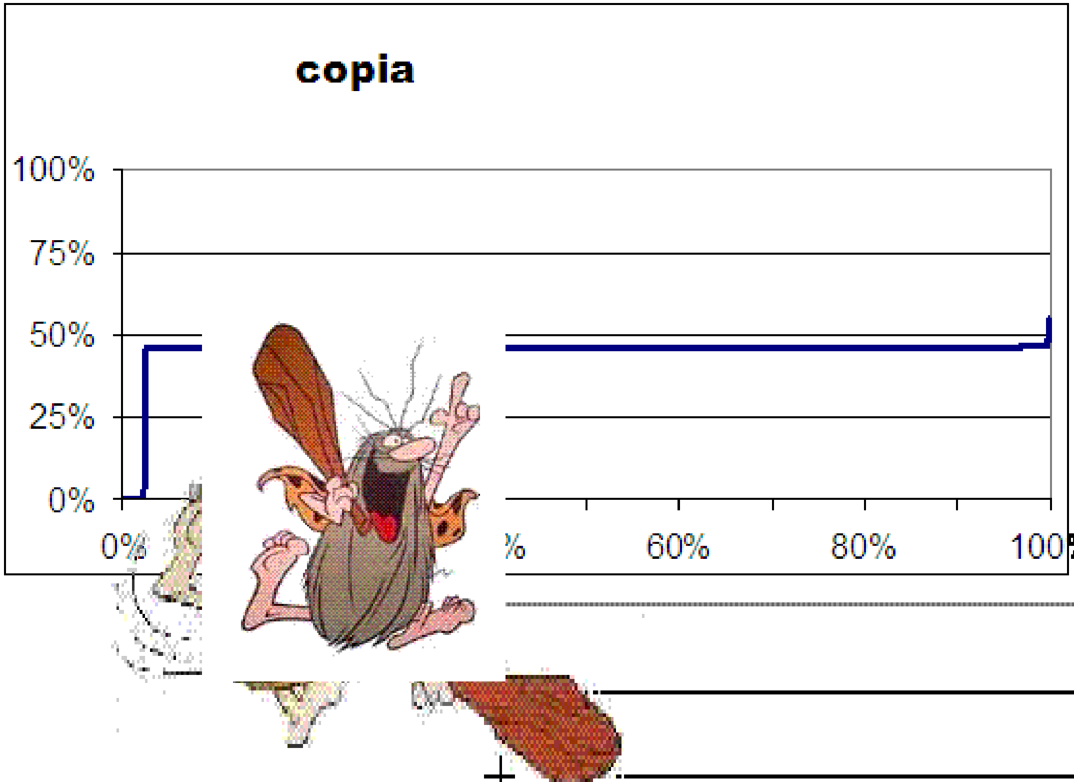
number.c includes Cluster 3, which implements a number formatter. While linked with the other functionality in the file some refactoring might take place.

util.c contains small functions used all over the program. Hence, changes in this file could have ripple effects.

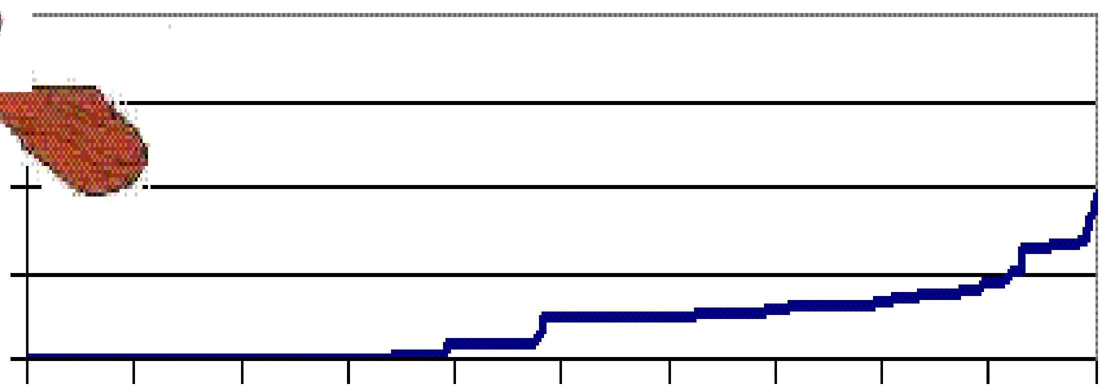
OK They Exist (and They are Bad), But, *Can **Causes** of Dependence Clusters be Identified*?

- Yes! *but how and at what cost ☺*
- How
  - By Hand
  - Automatically -- ACluB Project
- Cost
  - Semantics
  - Comprehension

Breaking Dependence Clusters First by Hand

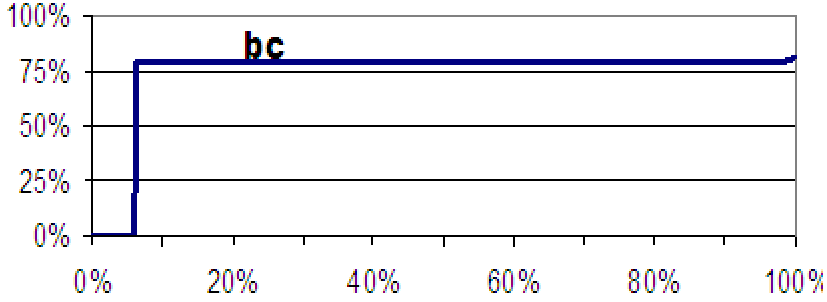


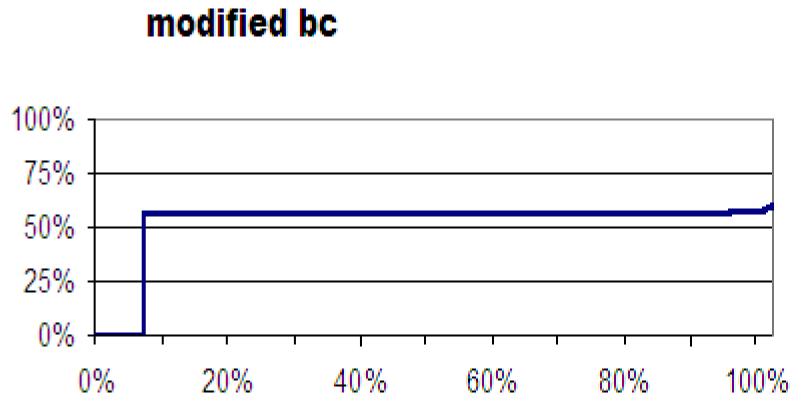
The Good



Breaking Dependence Clusters By Hand

The not-so Good



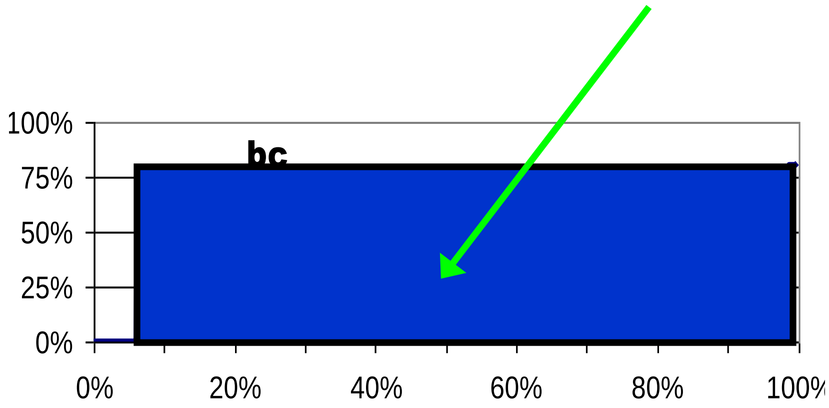


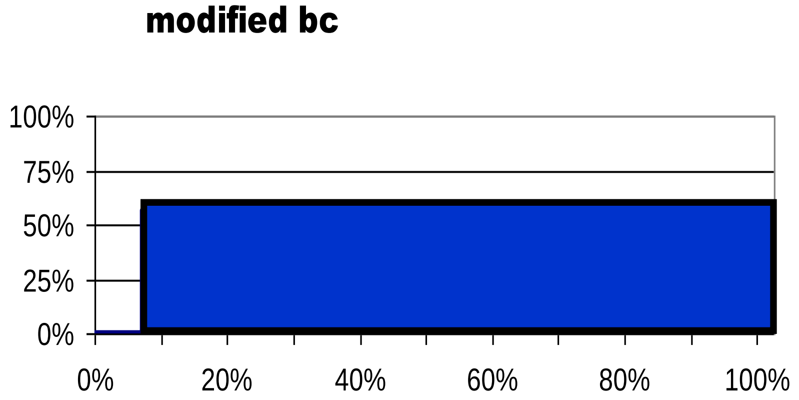
Automated Techniques

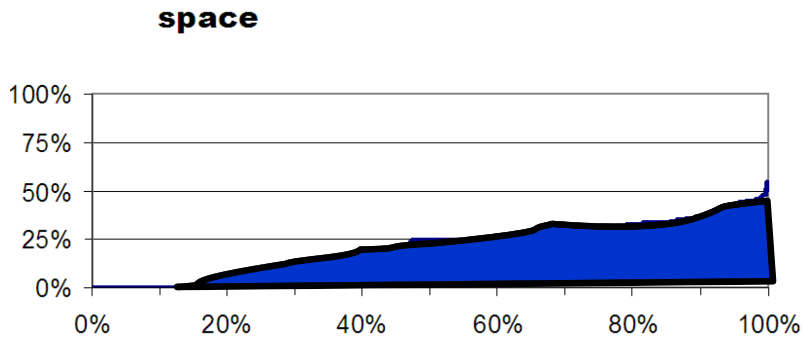
1. Ignore the dependences associated with each ***global variable*** then rebuild MSG
   - ☺ moving a pawn updates the *board*
     Surprised?
2. Ignore the dependences associated with each ***vertex*** then rebuild MSG
   - – a very small change to the “prograr

Measuring Impact

Area under MSG



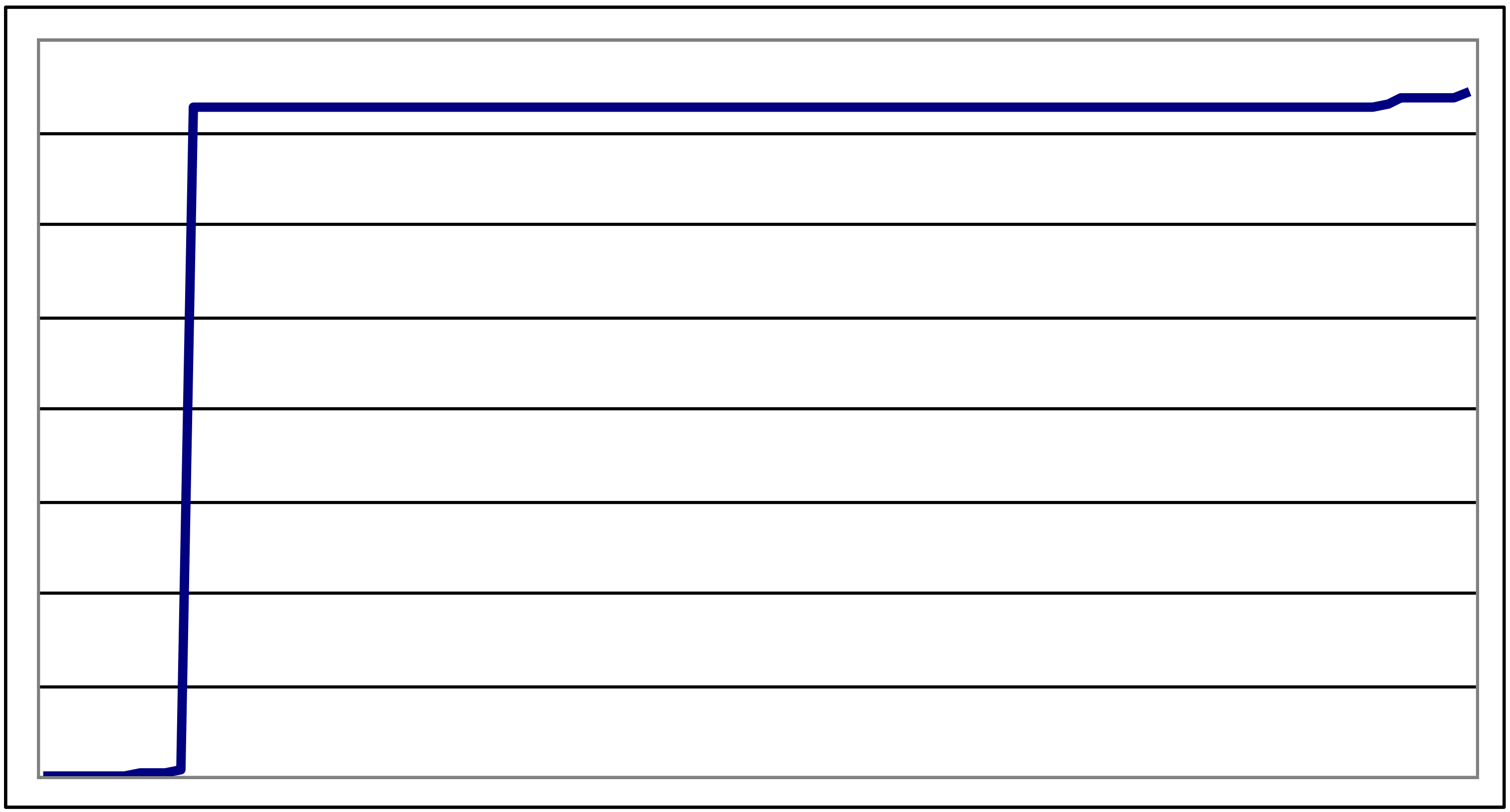




Automated Breaking

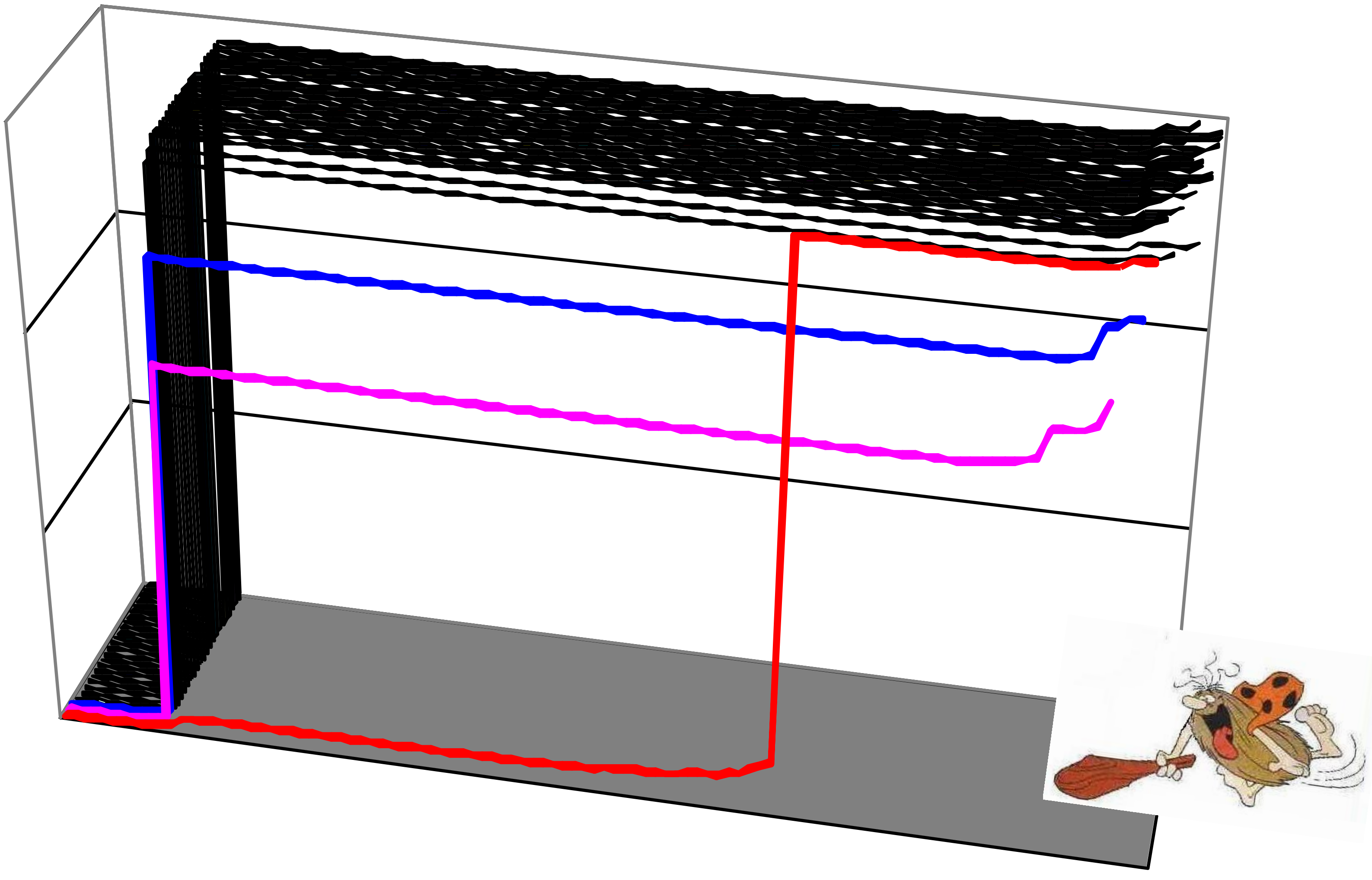
Ignoring Dependences Associated with a ***Global Variable***

MSG for PC2C

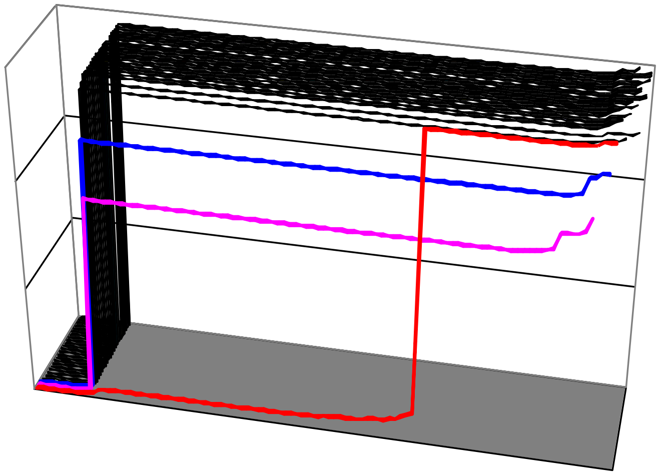


Automated Breaking

PC2C without each global



So Which Variable Is It?

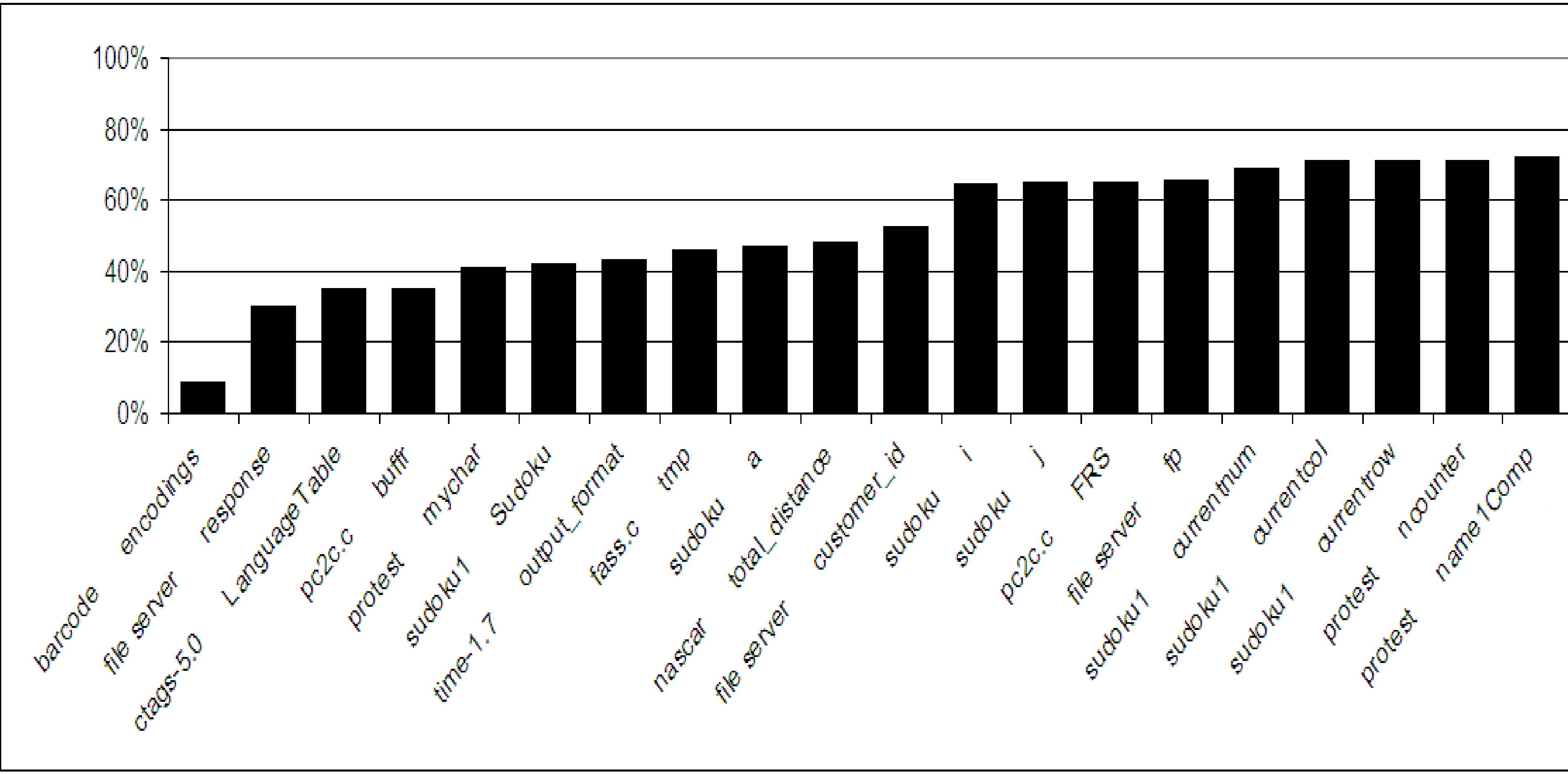


buffr – the input read buffer

```
if (InStrPos(buffr, ":=") >= 0)
{
   strcpy(buffr, ReplaceS(buffr, ":=", "="));
…
if (InStrPos(buffr, "CONST") >= 0)
    strcpy(buffw, "#define ");
```

Variables Causing Highly Significant Difference

20 of 849 Global Variables



Automated Breaking

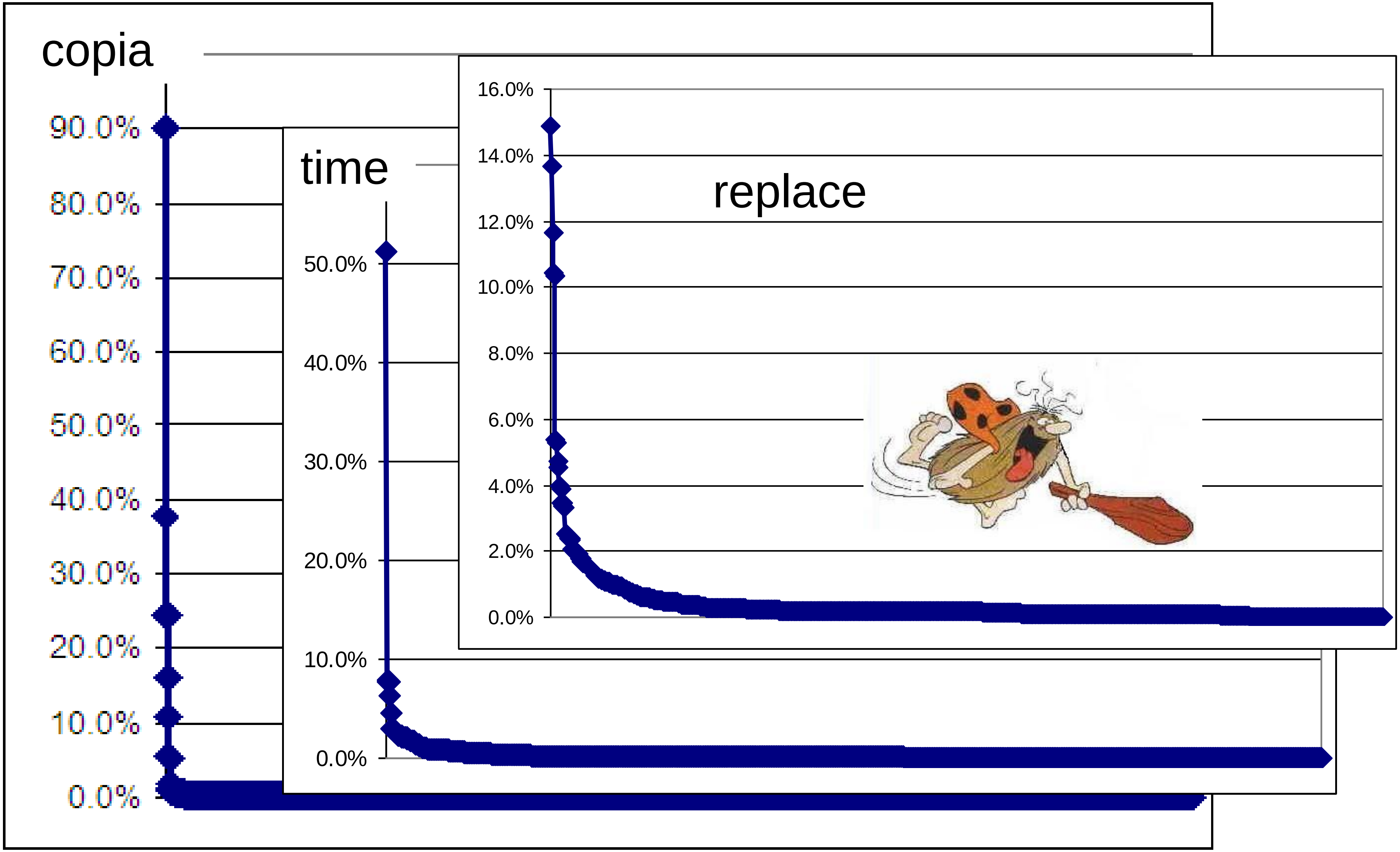
Ignore dependence associated with a ***statement (dependence graph vertex)***

- main()
- {
- while I < 10
- if A[I] > 0
- I = I + 2;
- }

Good News!

slices	program	reduc-tion	vertex type	source
4686	copia	89.96%	control-point	switch (a)
1044	time-1.7	51.23%	control-point	switch (*++fmt)
1077	conversion	27.04%	control-point	switch (pick_op)
747	driver	26.45%	control-point	switch(choice)
585	sudoku	22.87%	control-point	while(!check_completed())
11277	space	7.90%	control-point	if (error != 0)
9556	gnubg-0.0	7.32%	indirect-call	pc->pf()
3909	barcode	5.95%	indirect-call	cptr->encode()
12492	EPWIC-1	5.79%	control-point	while(state != DONE)
10151	byacc	1.54%	expression	k = keyword()

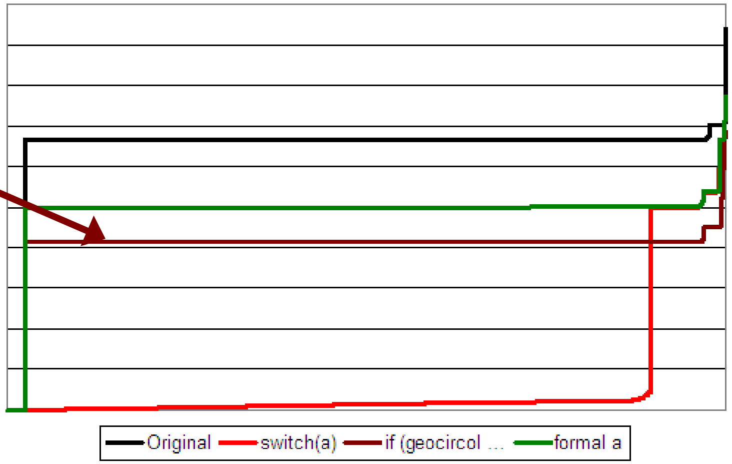
Linchpin Vertices?



MSGs for Copia's Top 3

Function "Confirm" from Copia

- conf()
- {
-     if (geocircol==1 && ... 10 more comparisons ...
-         grexc1();
-     else
-         grexc2();
- }



Function “Next-state” from Copia

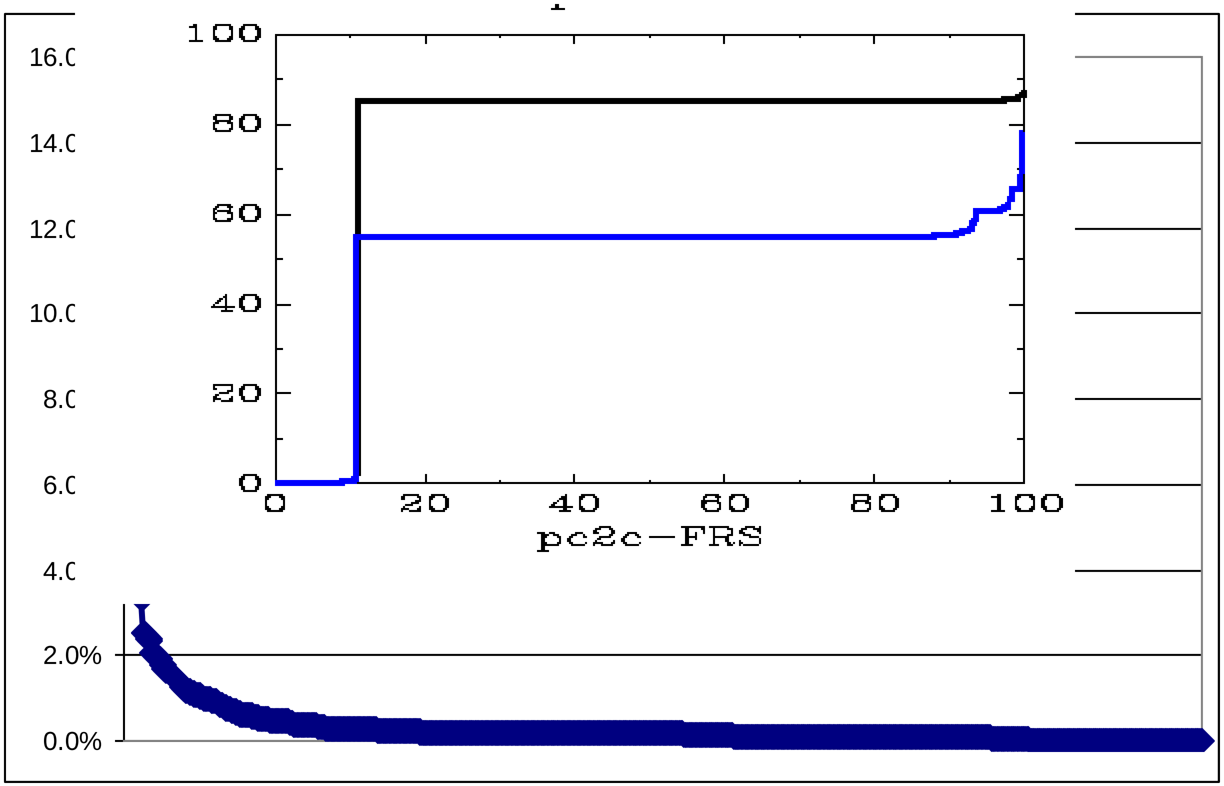
- void seleziona(int a)
- {
-     switch (a) {
-         case 0: grid(); break;
-         case 1: hex(); break;
-     …

Cost

- Computing the MSG takes $O(n^2)$ time
- Doing so for each vertex can take a couple of cups of tea!
- Might there be some efficiency improvements?

Efficiency Improvements

Goal: bound area change



Three Efficiency Improvements

- Small Impact
- Declarations
- Dual Paths

Small Impact

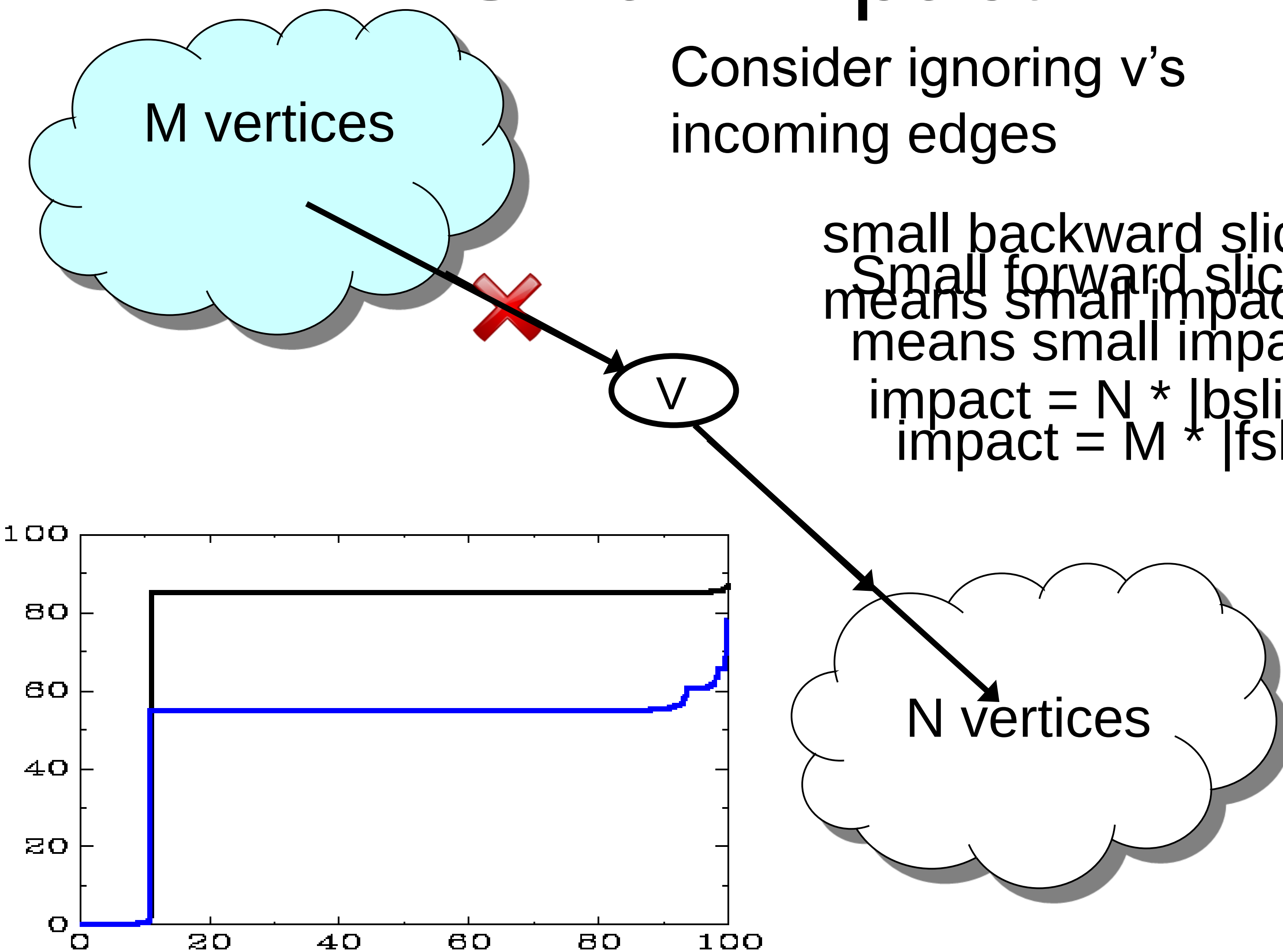
Consider ignoring v’s incoming edges

small backward slice means small impact:

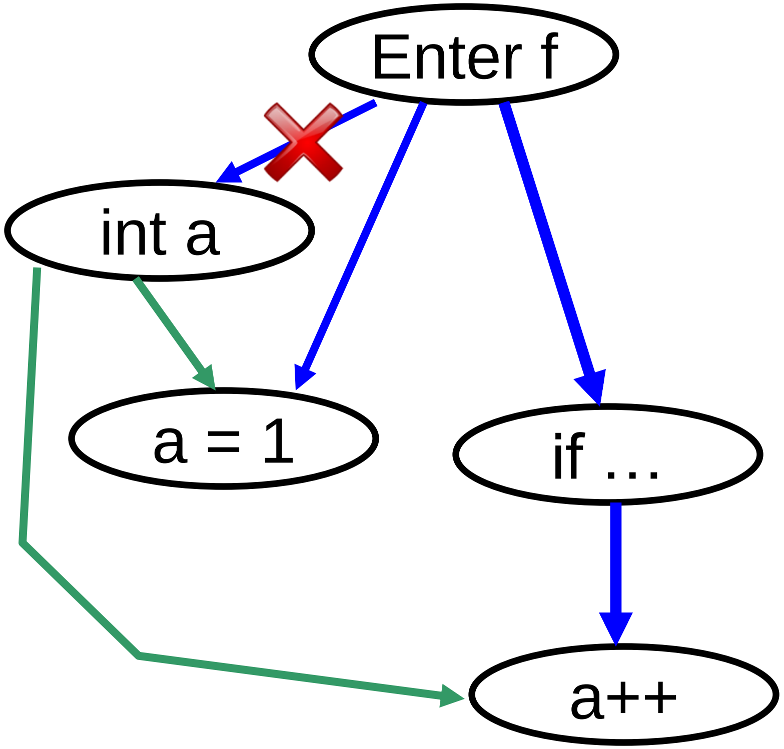
impact = N * |bslice(v)|

Small forward slice means small impact:

impact = M * |fslice(v)|



Declarations

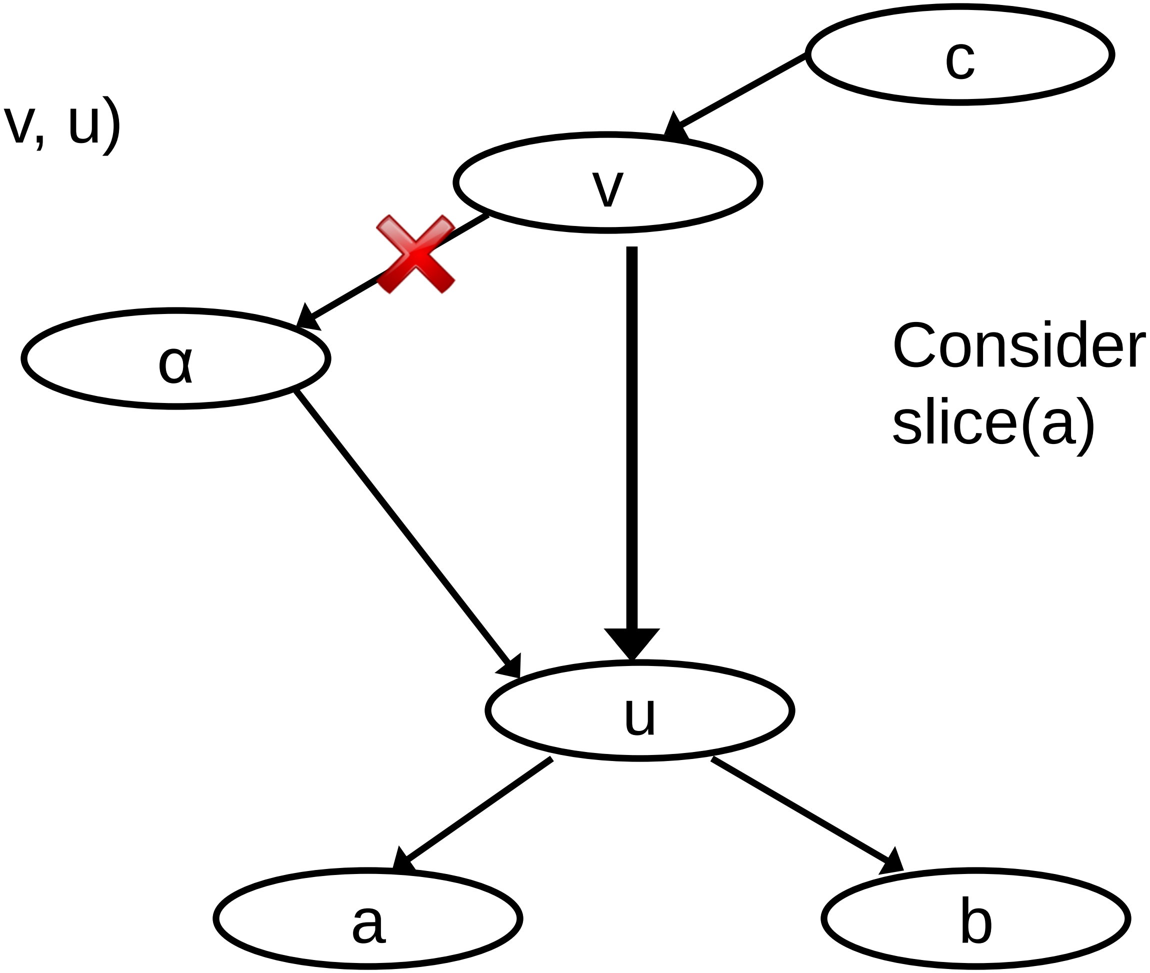


Control Dependence

Data Dependence (definition – use)

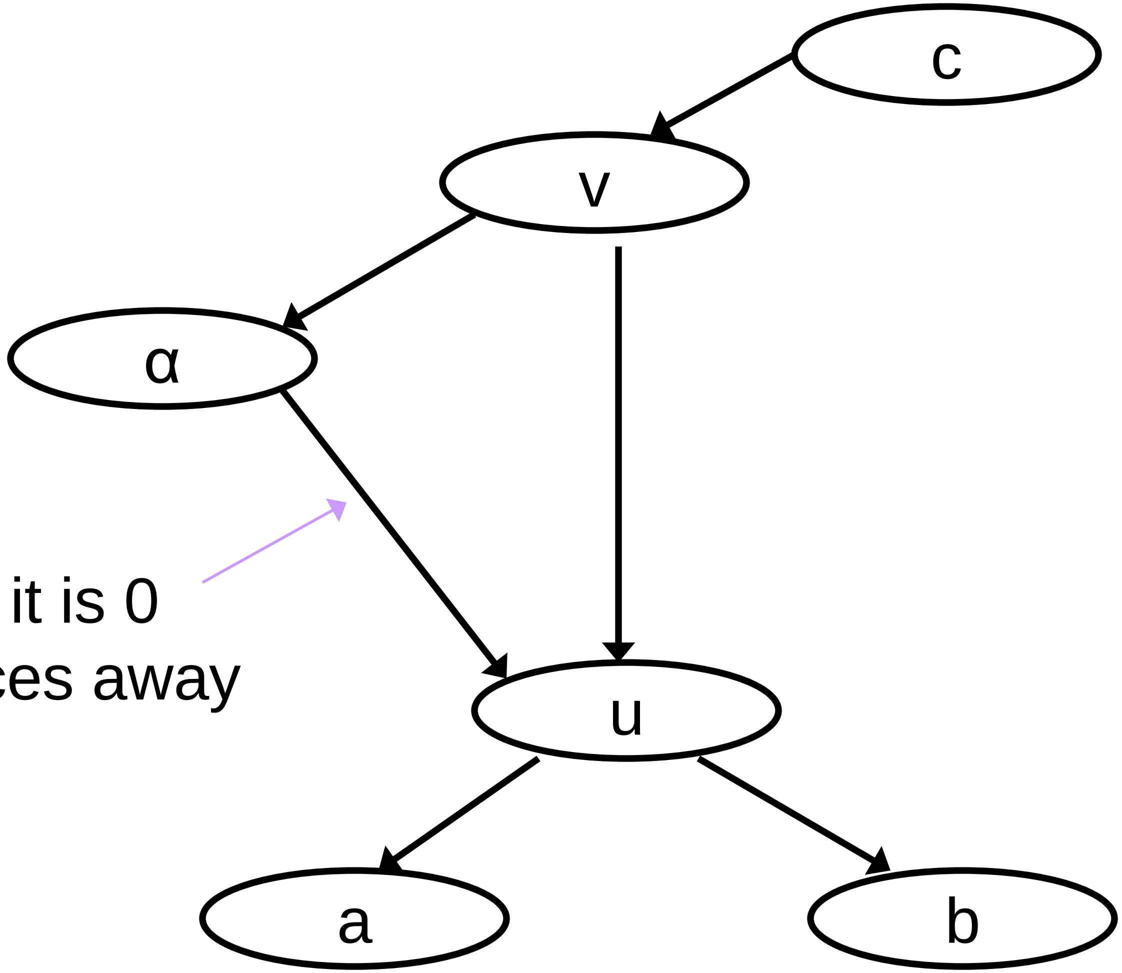
Dual Path Property

dpp(α, v, u)



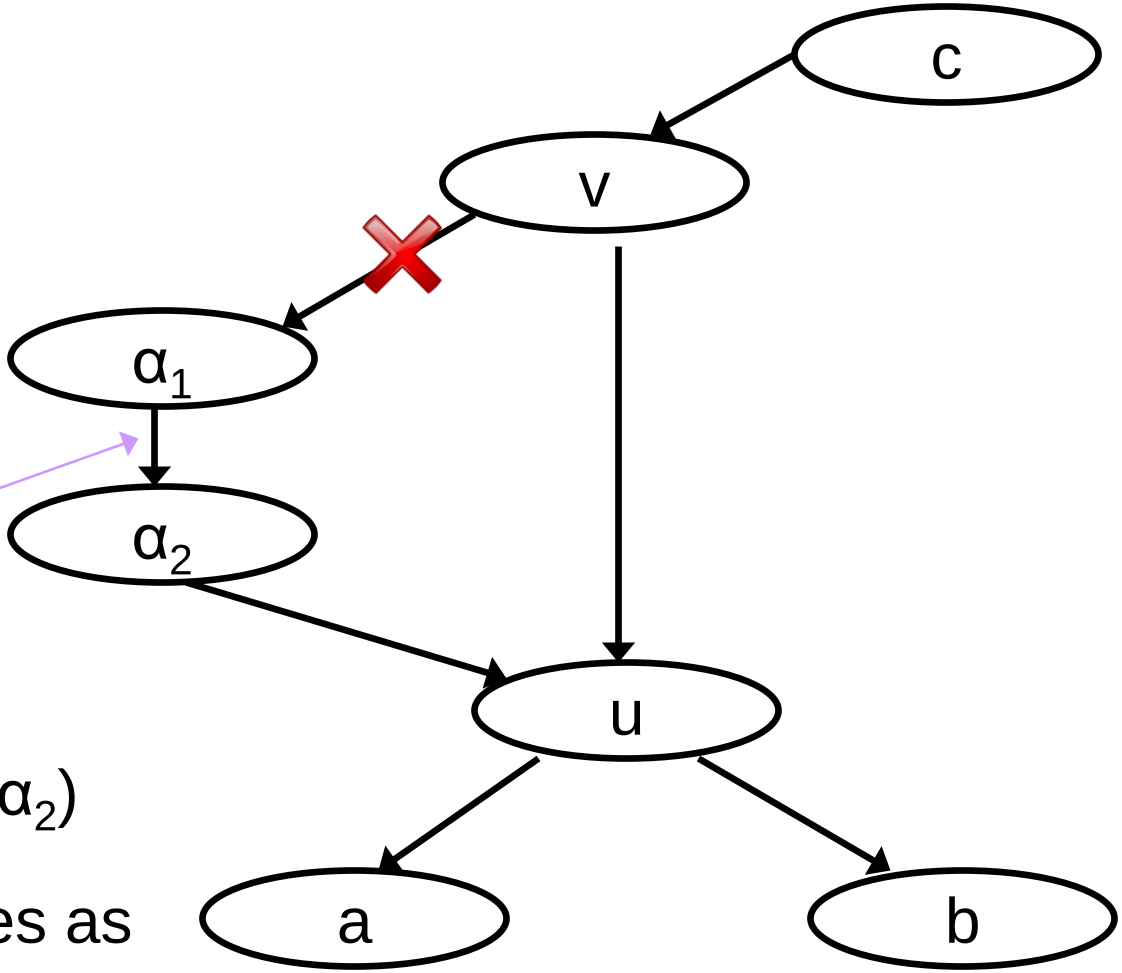
Consider slice(a)

Challenge is finding ‘u’ from α



Here it is 0 vertices away

Challenge is finding 'u'



Here 0 does not cut it

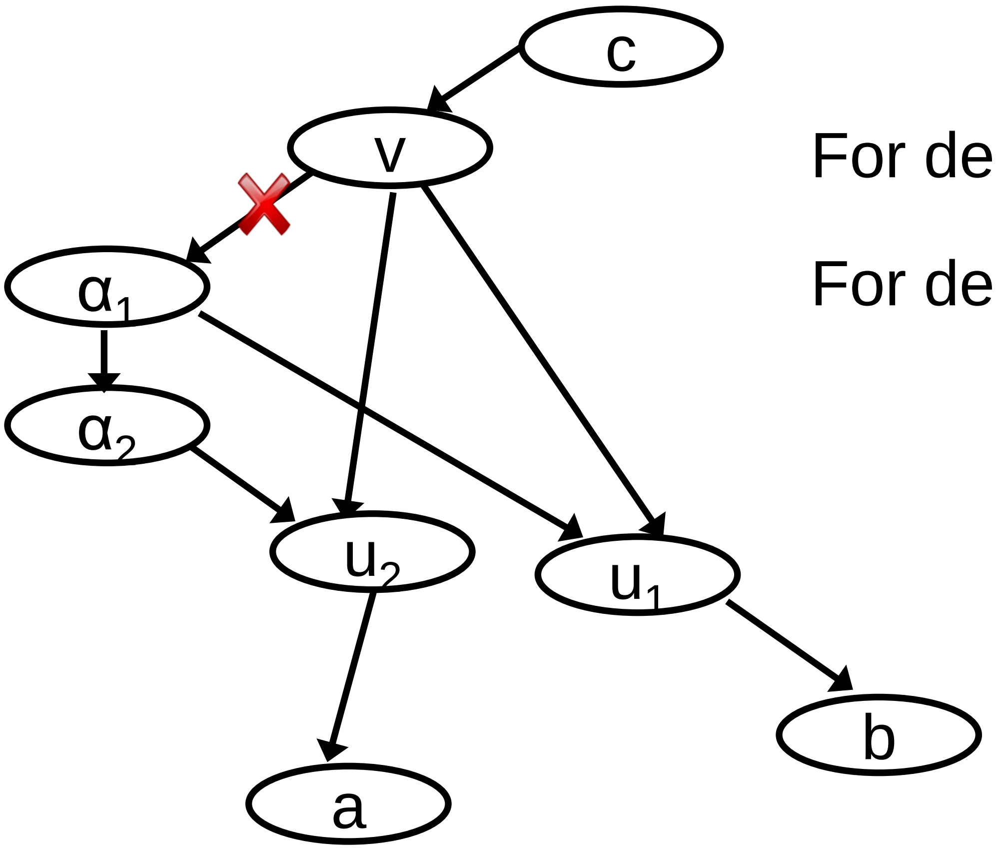
dpp(α_1, v, α_2)

But 1 does as dpp(α_1, v, u) holds!

Three Algorithms

- Static Depth
- Dynamic Depth
- Borderless

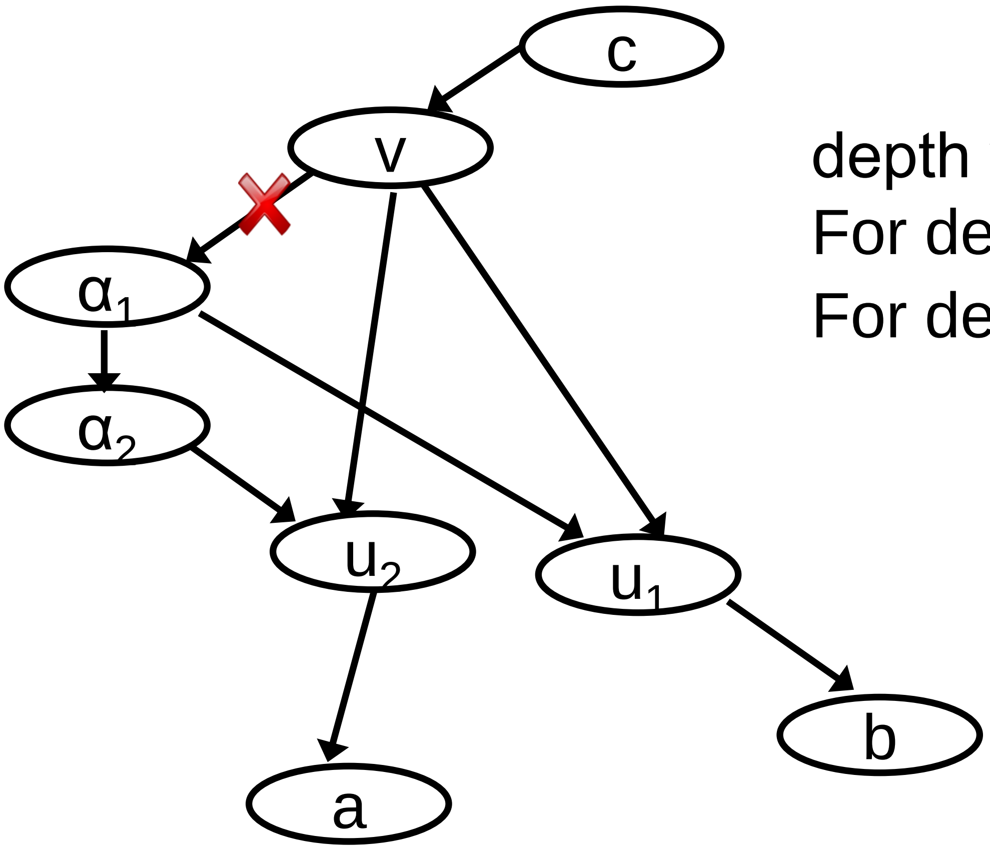
Static Depth
(a disaster ☹)



For depth=0 dpp(α_1, v, u_1)

For depth=1 dpp(α_2, v, u_2)

Dynamic Depth
way better ☺



depth varies by ‘u’

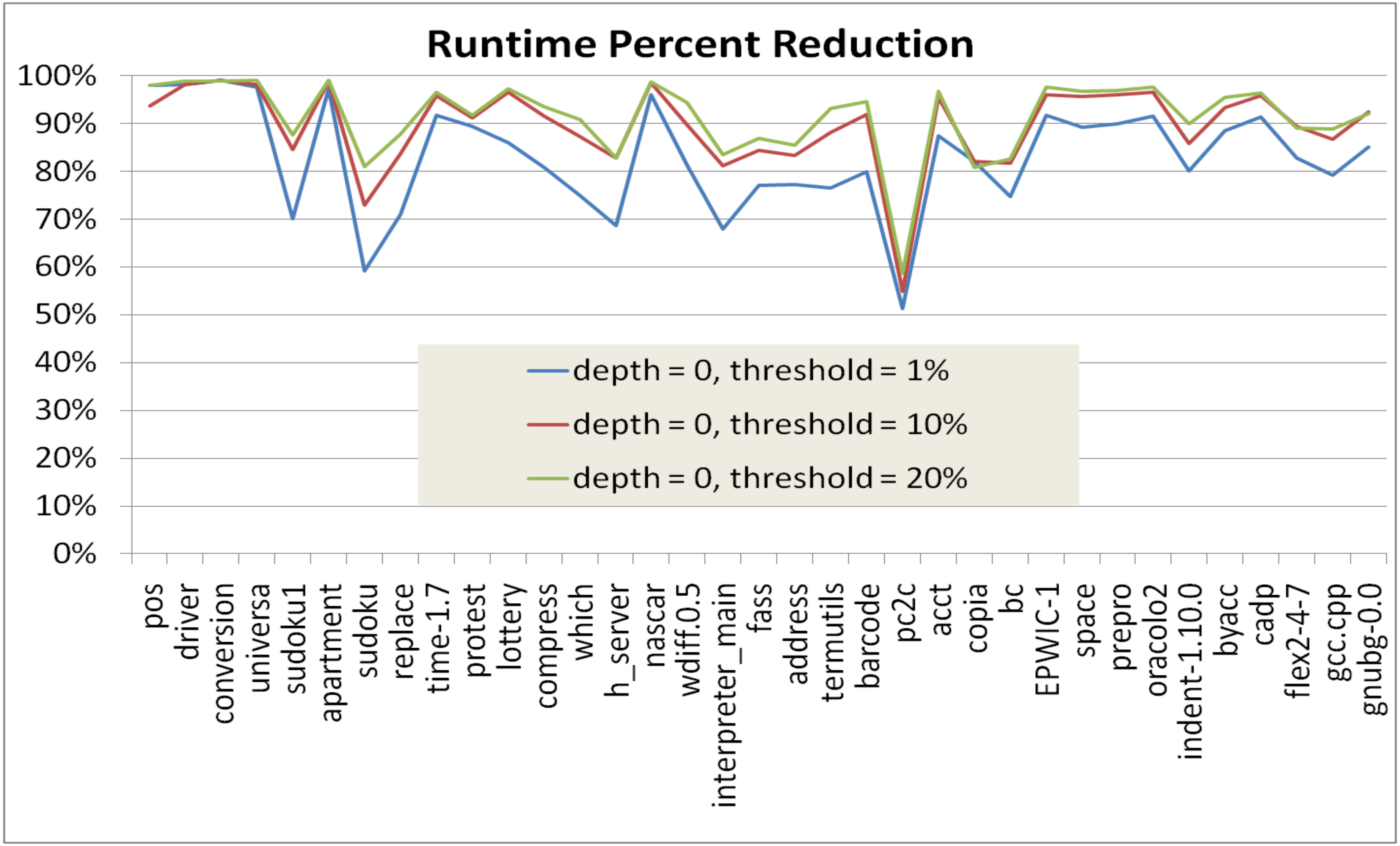
For depth=0 dpp(α_1, v, u_1)

For depth=1 dpp(α_2, v, u_2)

Borderless

- Original theory too weak to deal with crossing procedure borders
  - Therefore fringe (u's) had to be in same procedure as α
- Improved theory led to borderless search

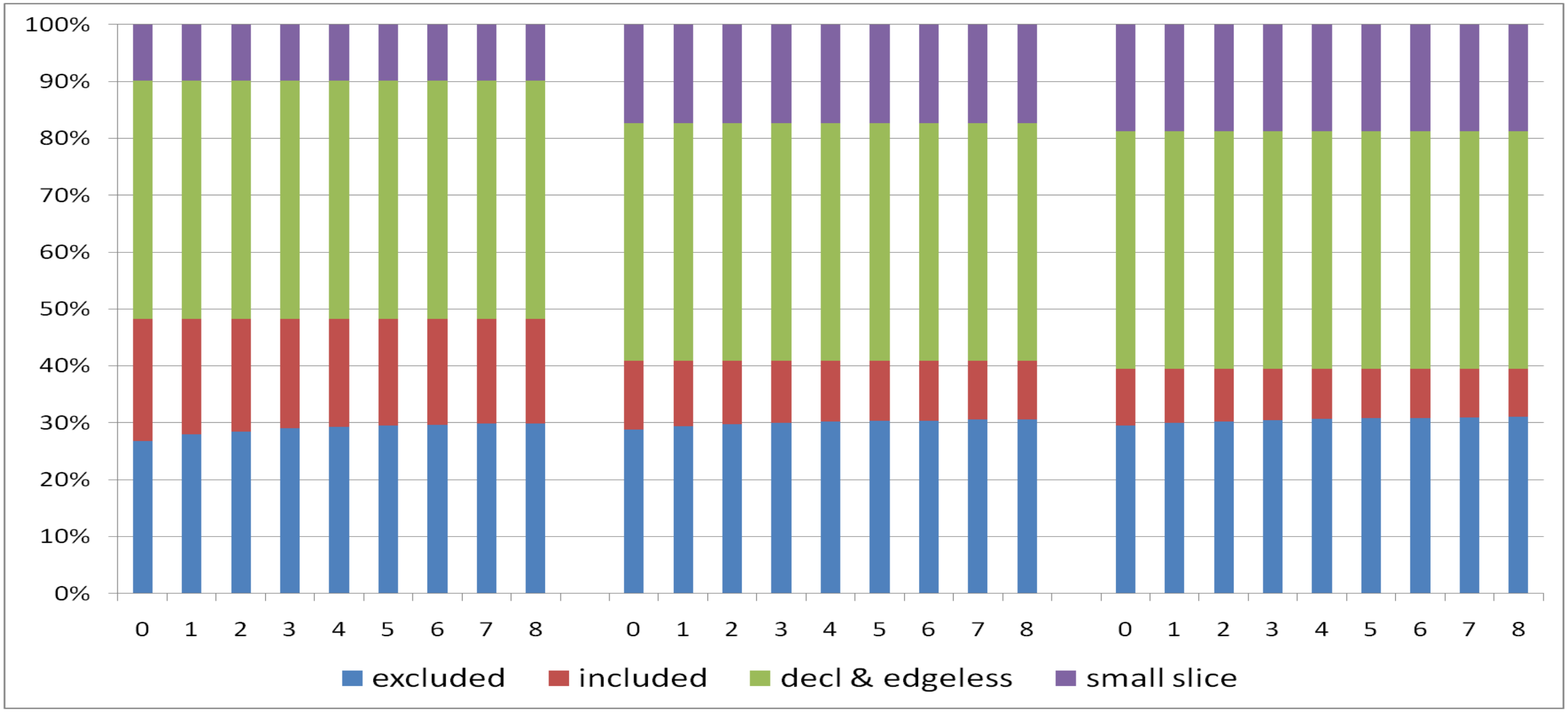
Great Theory, but
How about some Data ☺



Averages 83%, 90%, 91%

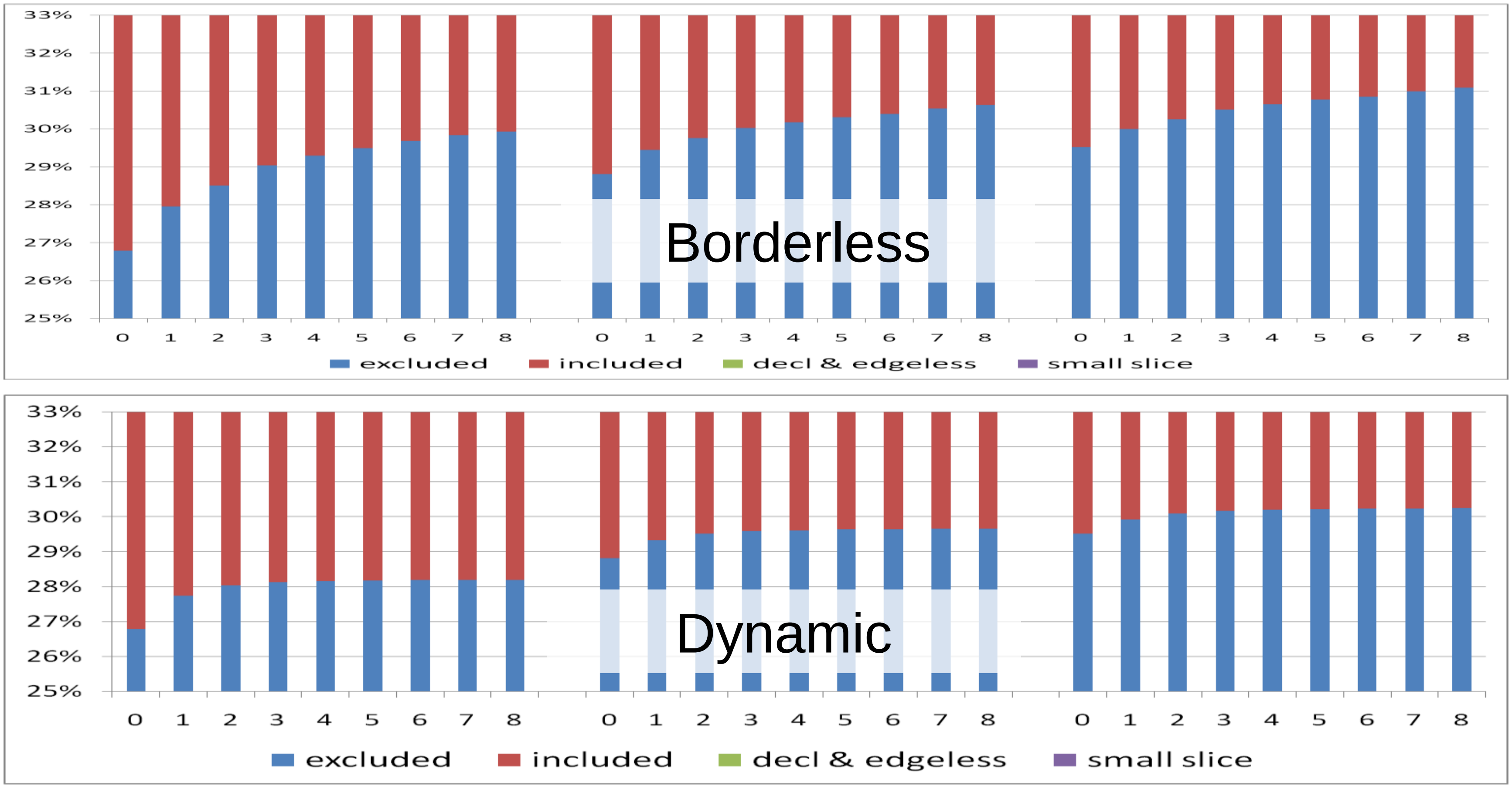
Vertex Classification - Borderless

Threshold = 1%　　Threshold = 10%　　Threshold = 20%

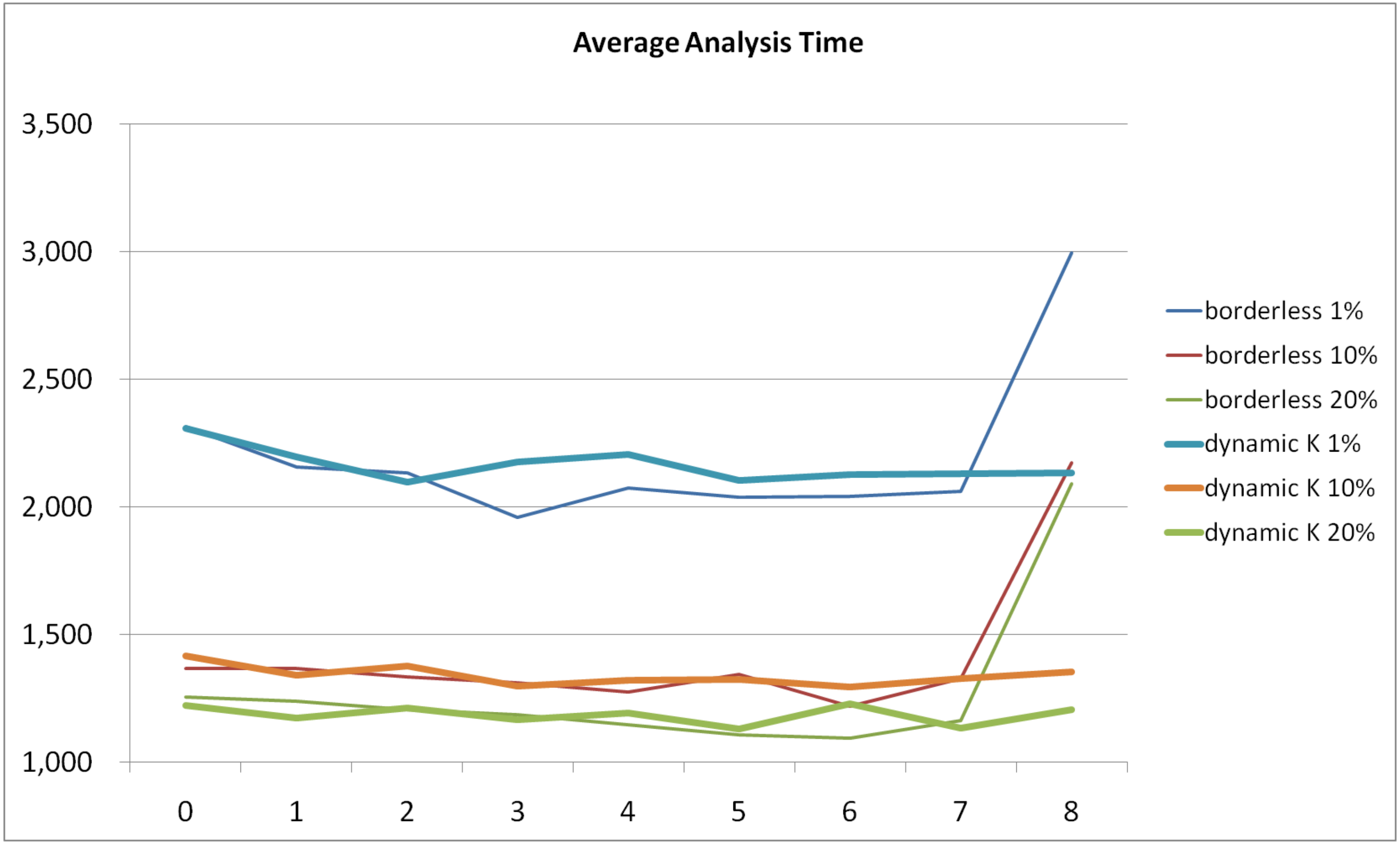


Zoom

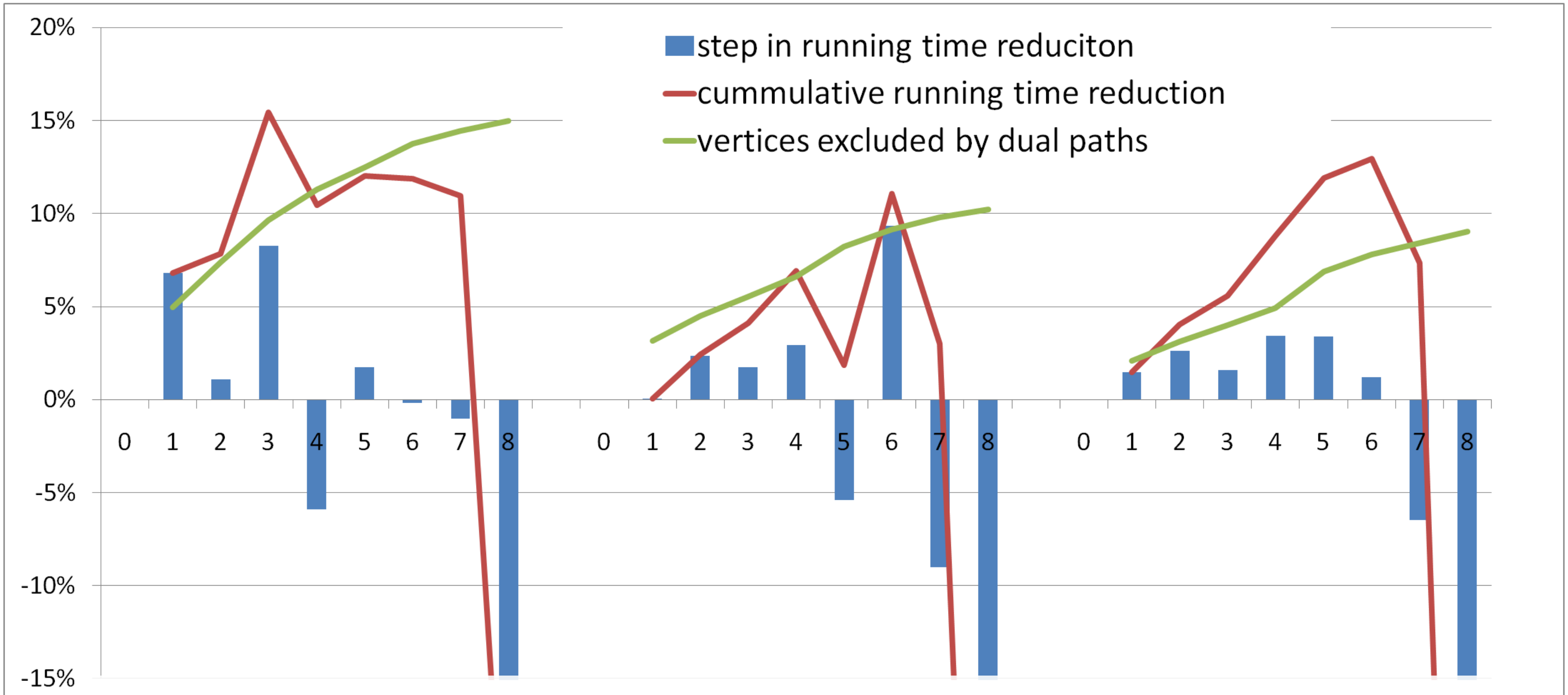
Threshold = 1%　　Threshold = 10%　　Threshold = 20%



Time Comparison



Time and Work Together Borderless



Threshold = 1% Threshold = 10% Threshold = 20%

Summary

- Dependence Clusters Exist
- They impact all (dependence based) static analysis
- Some are held together by a *single* vertex
- With a little work the search for these ***linchpin vertices*** can be made efficient

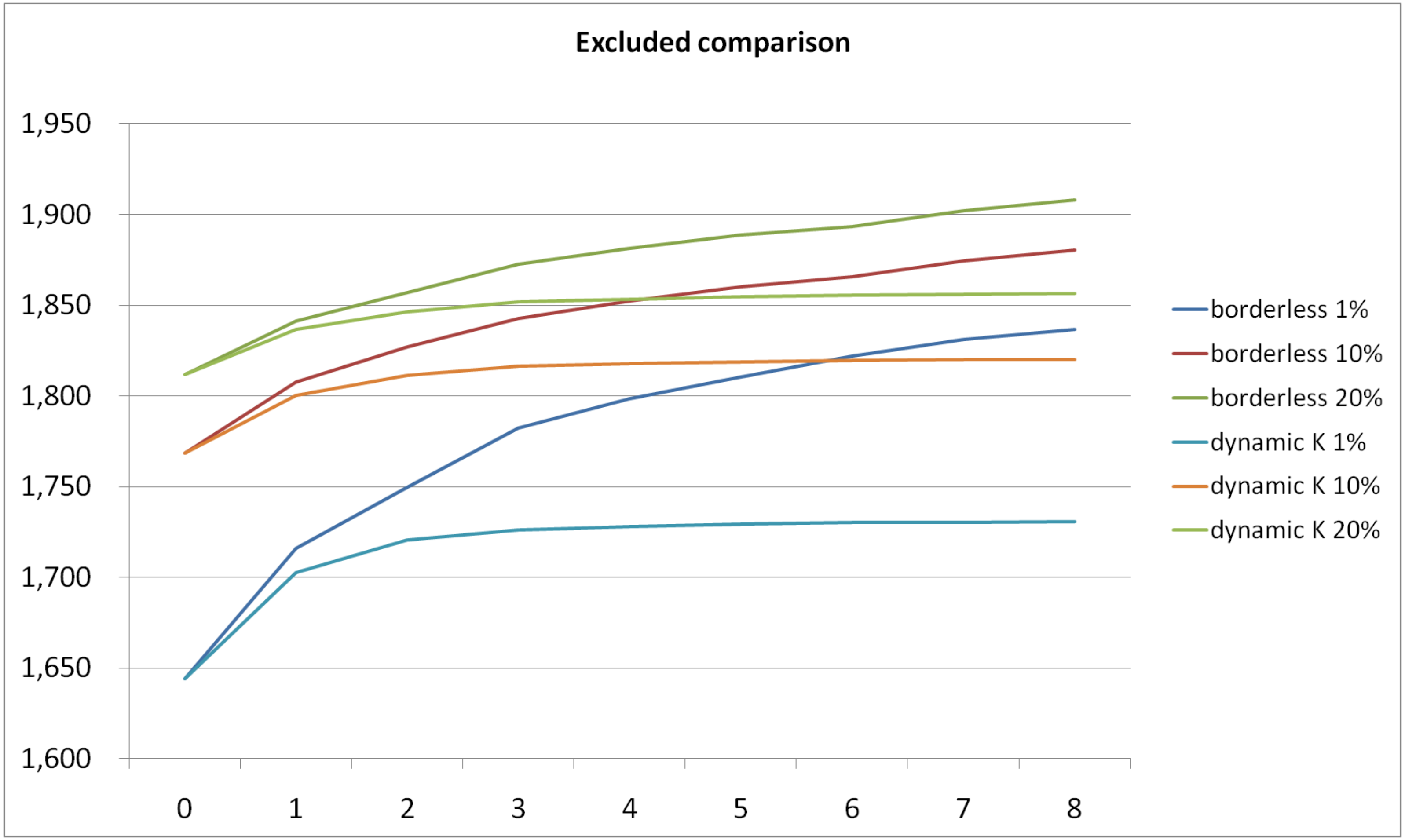
Thanks!

Questions ?

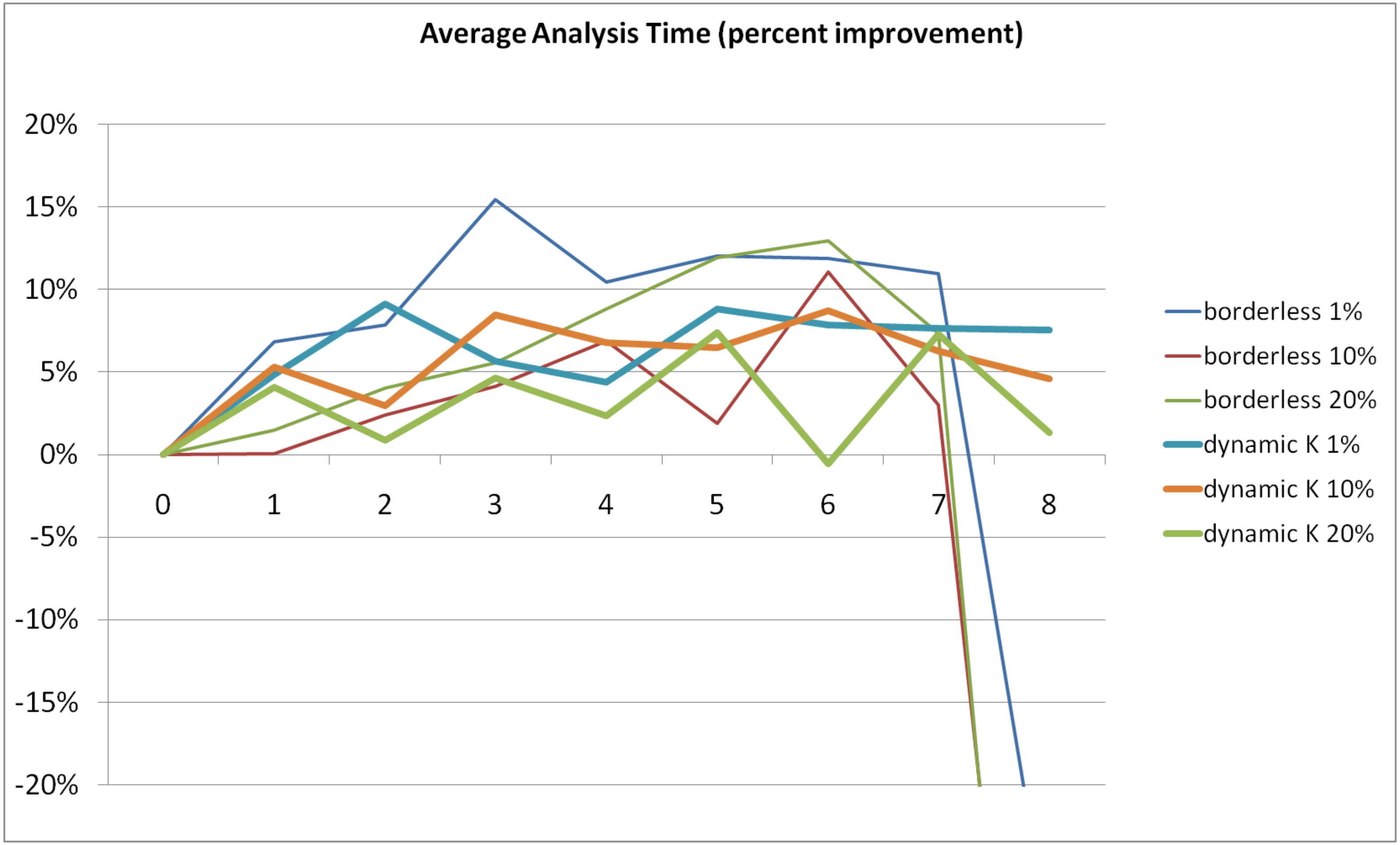
ACluB Web Page

www.dcs.kcl.ac.uk/staff/mark/aclub

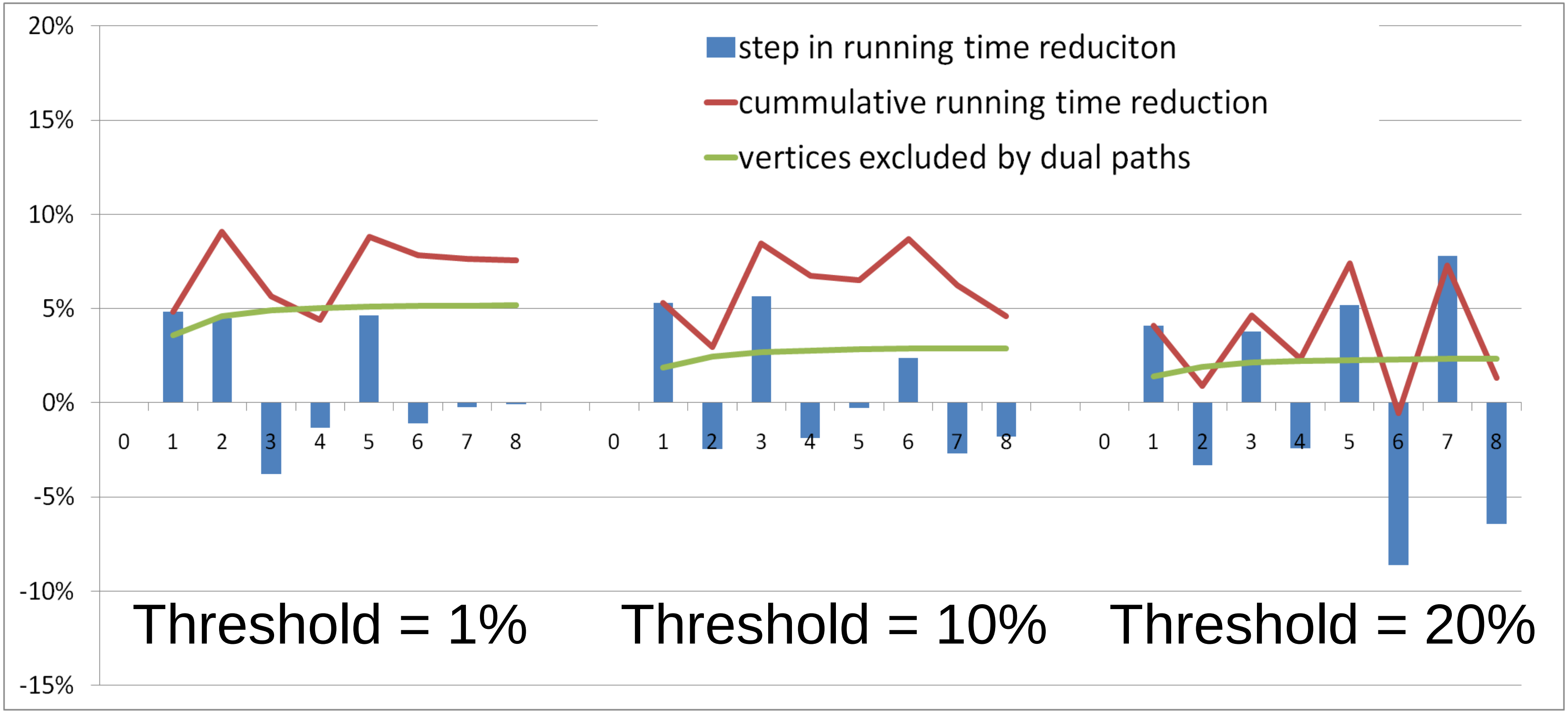
Vertex Classification - Comparison

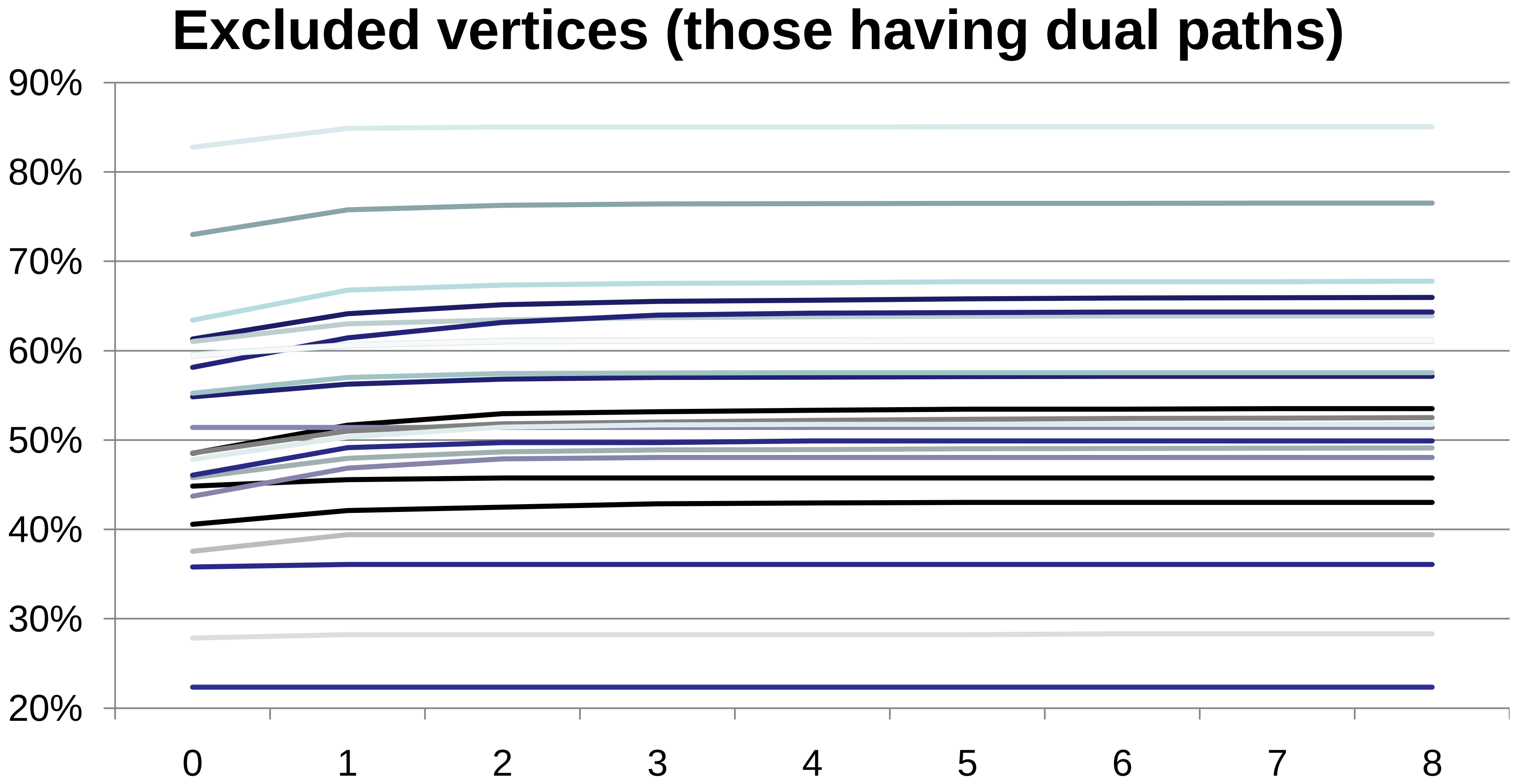


Time Comparison

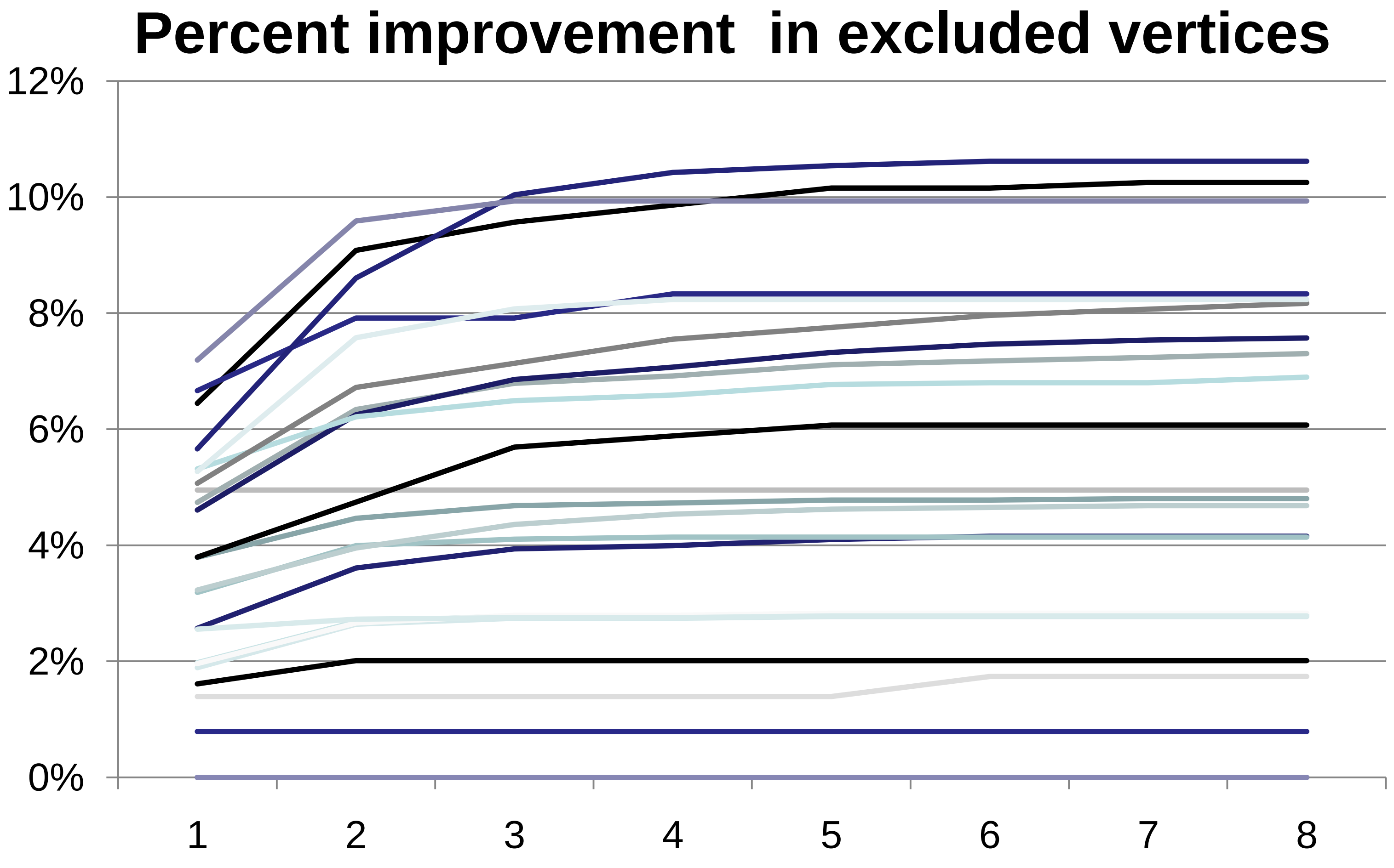


Time and Work Together - Dynamic



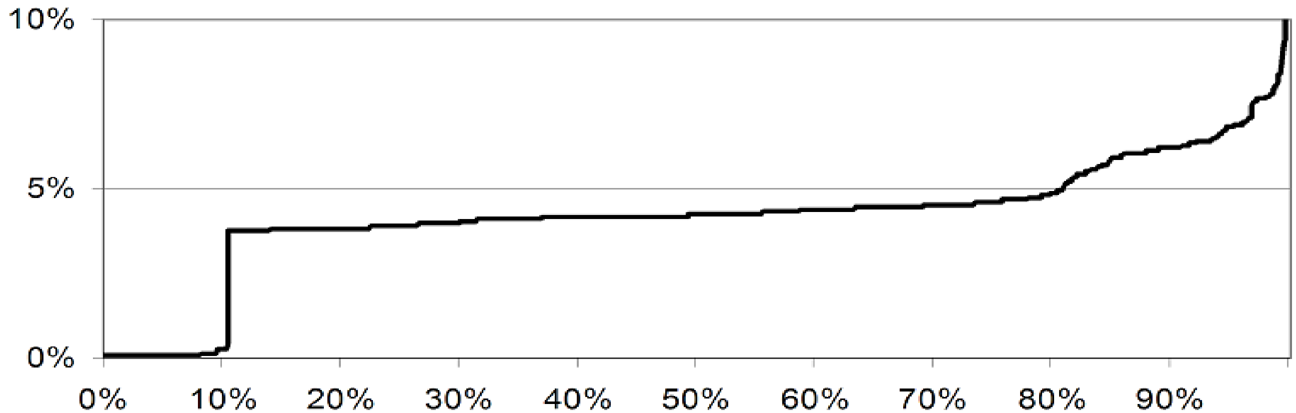


Excluded vertices (those having dual paths)
90%
80%
70%
60%
50%
40%
30%
20%
0
1
2
3
4
5
6
7
8

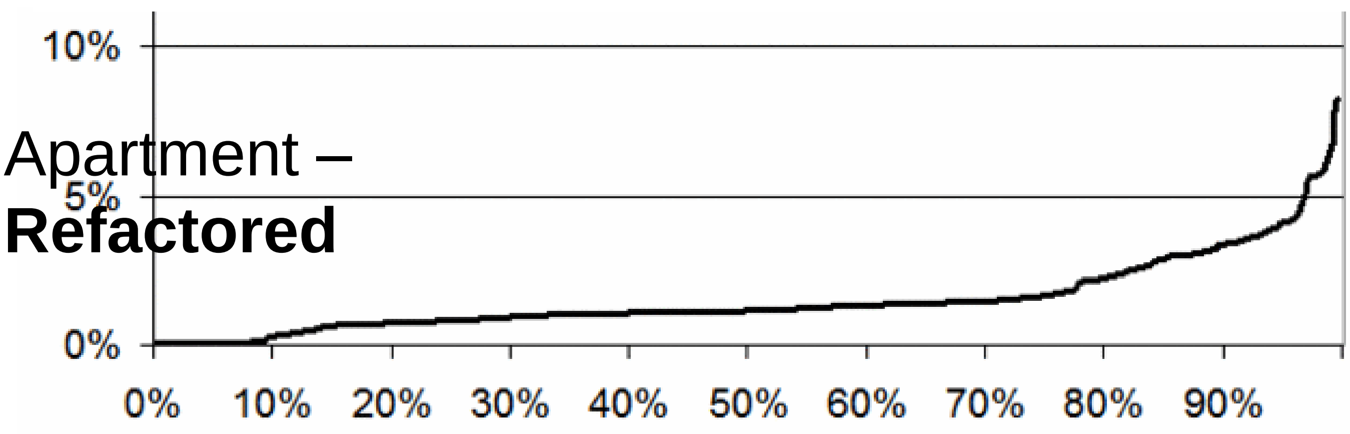


Percent improvement in excluded vertices
12%
10%
8%
6%
4%
2%
0%
1
2
3
4
5
6
7
8

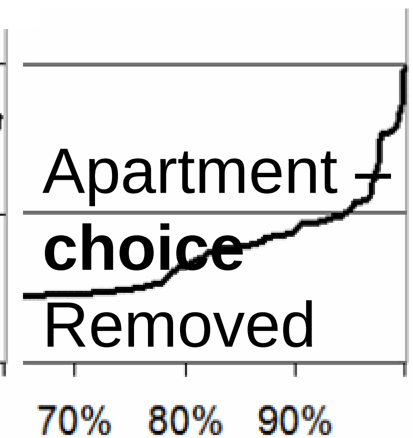
Semantics-preserving Refactoring



Apartment –
Original

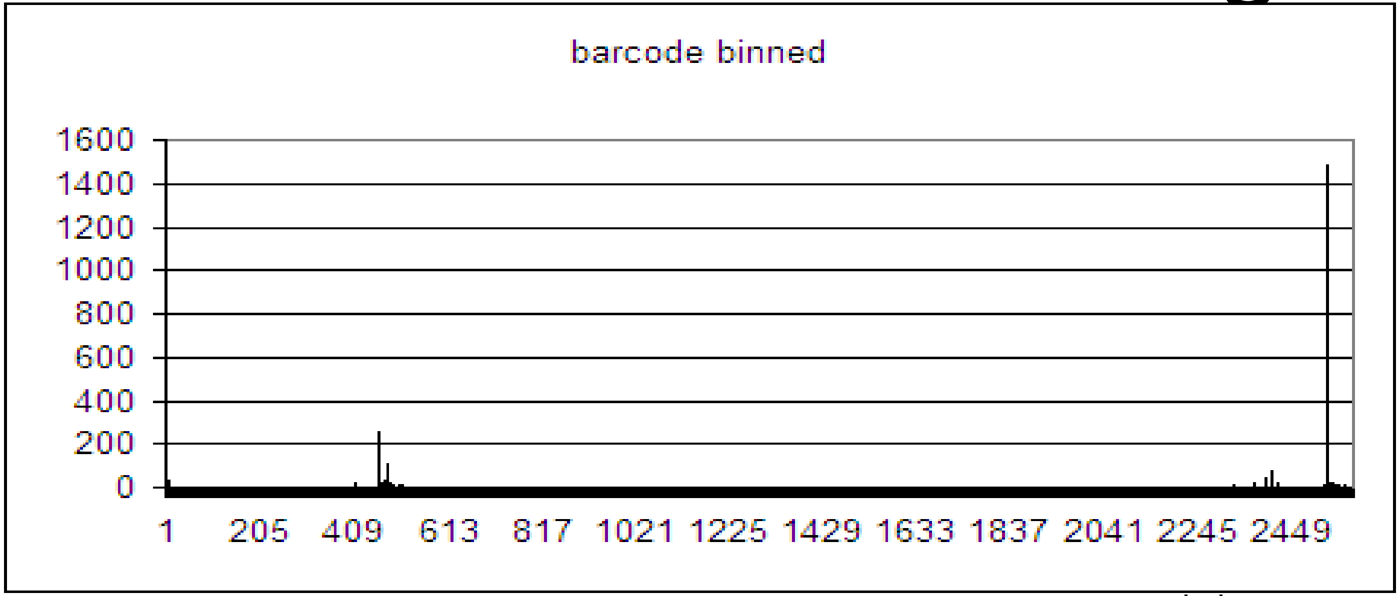


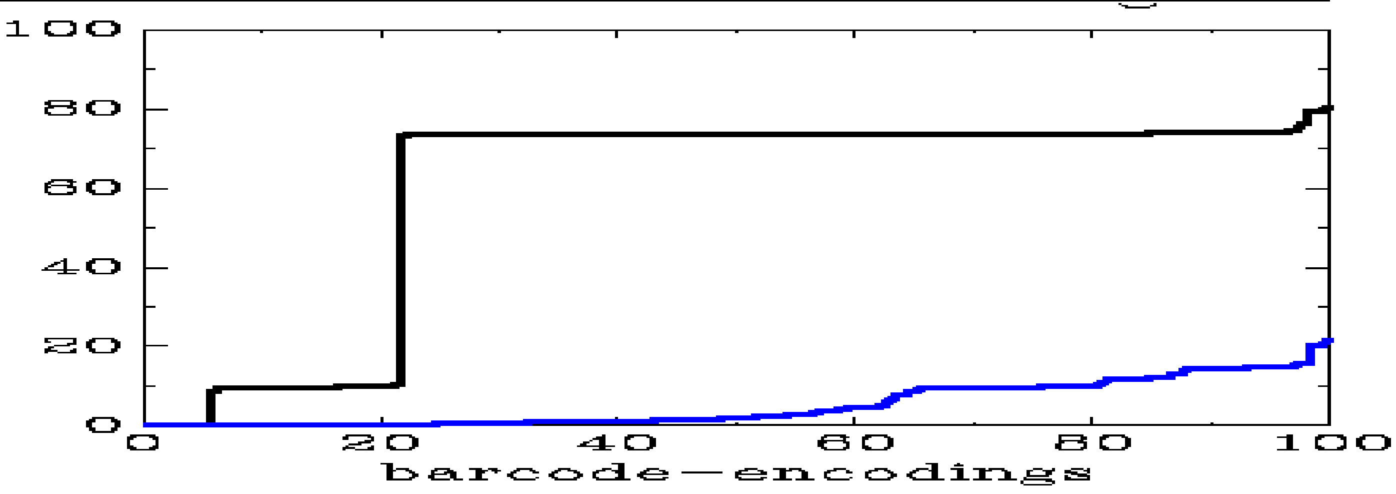
Apartment –
Refactored



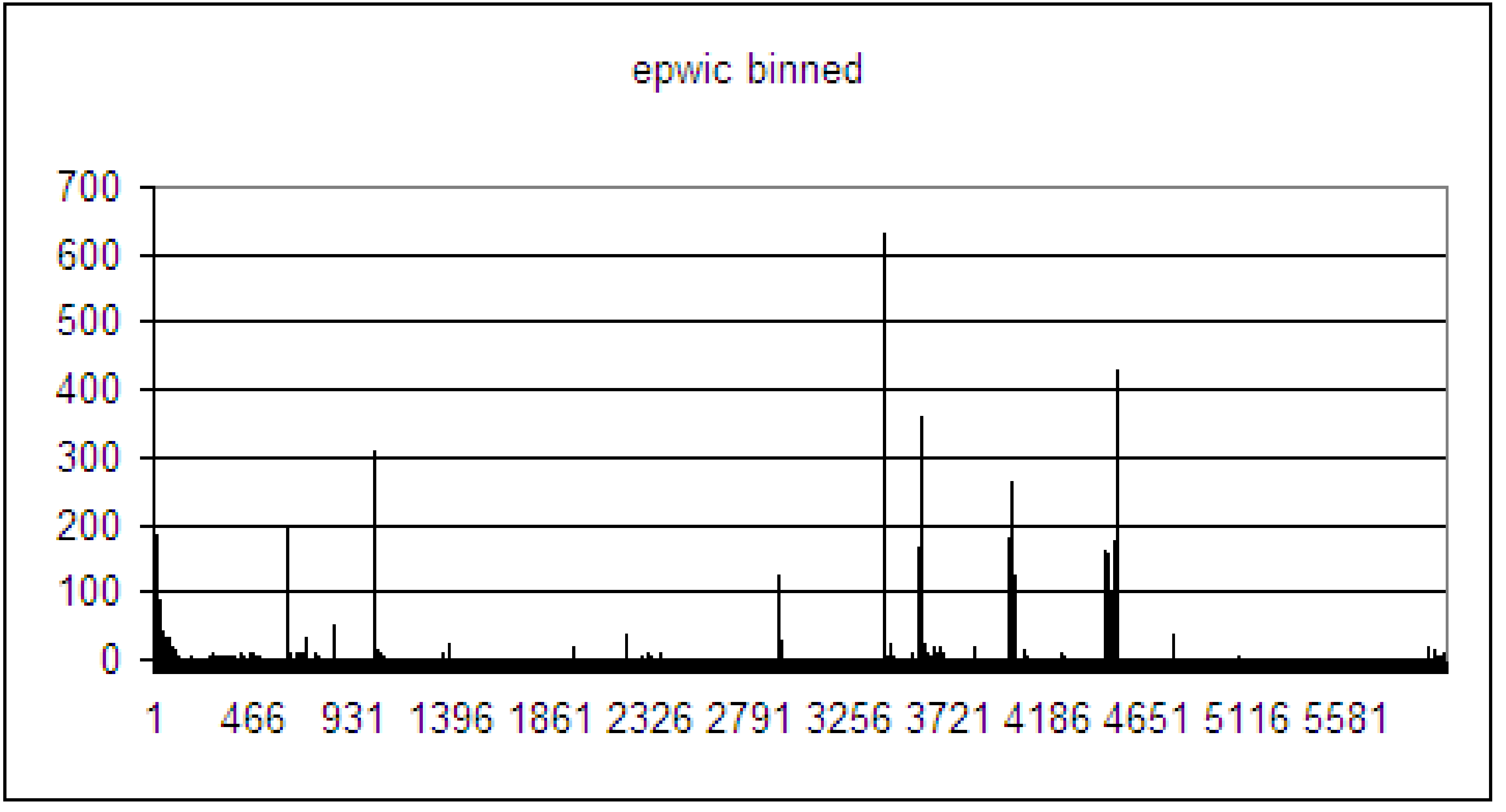
Apartment –
choice
Removed

Alternate Views - Histogram

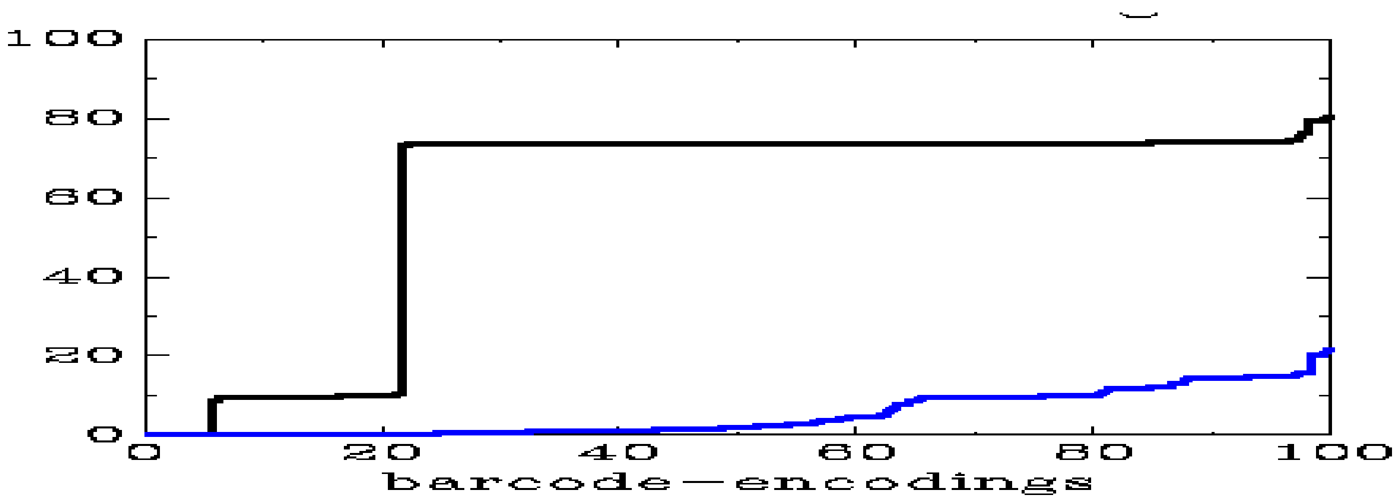


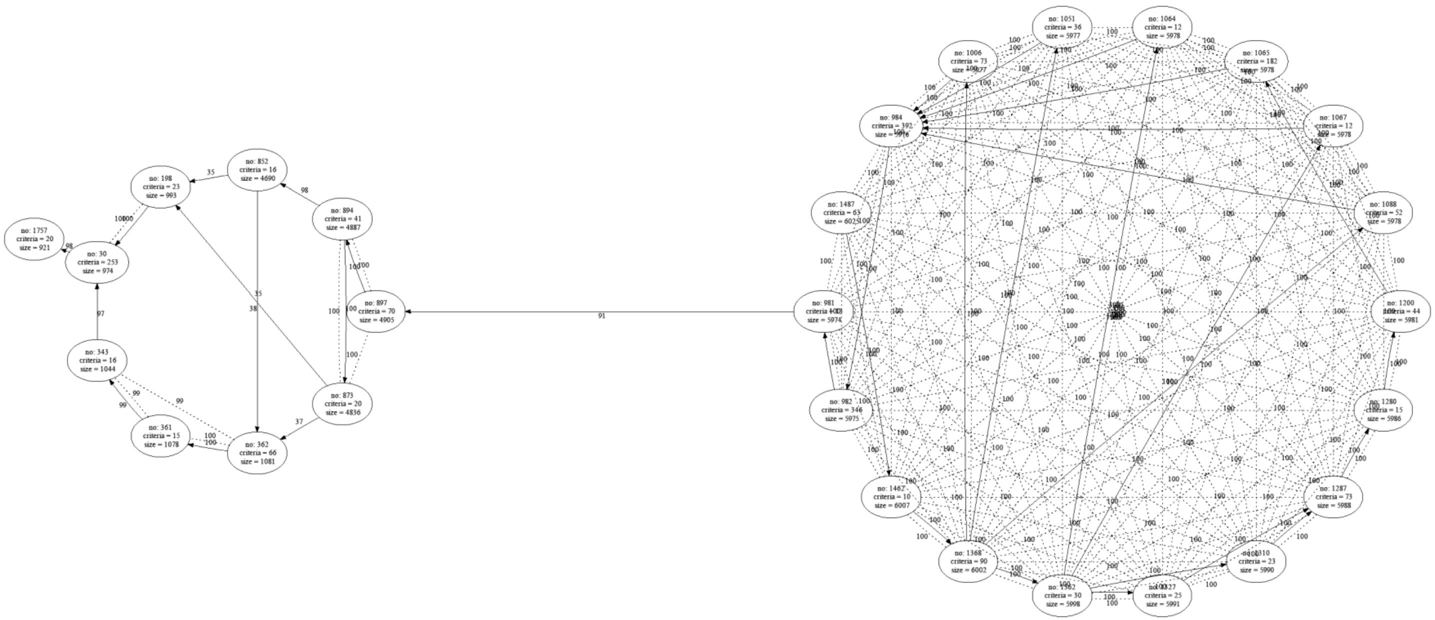


EPWIC Histogram

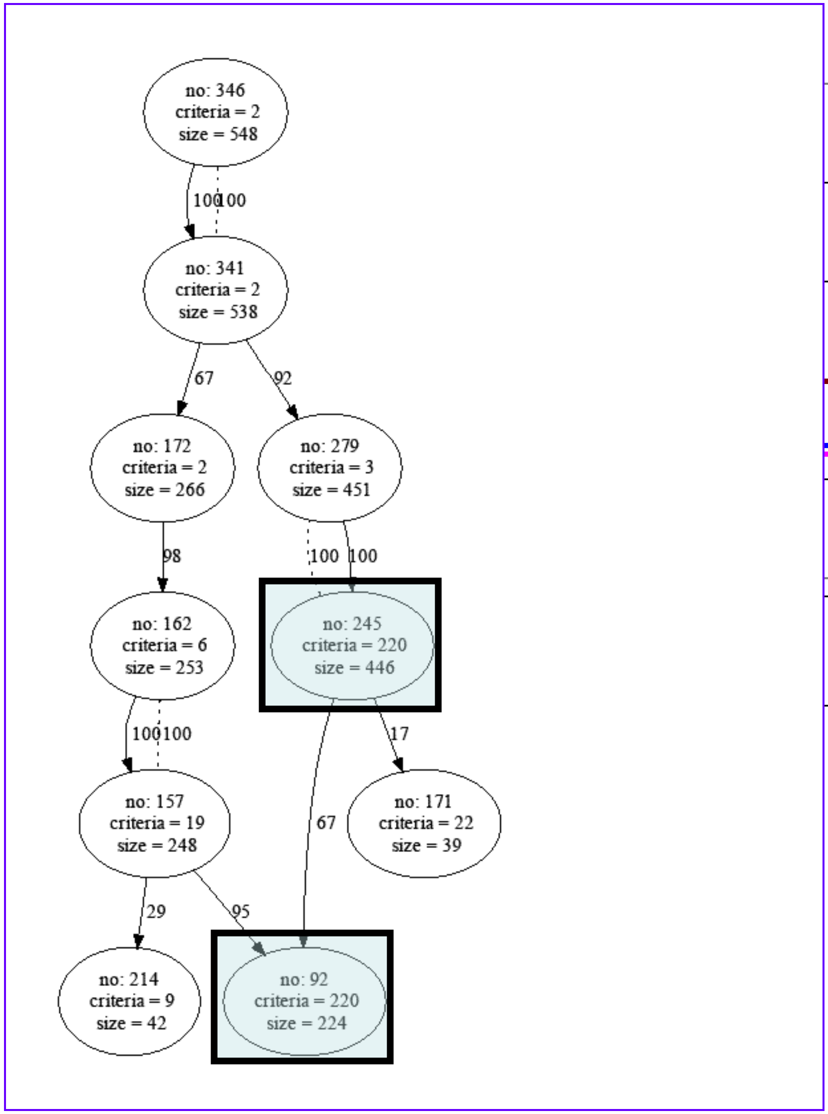


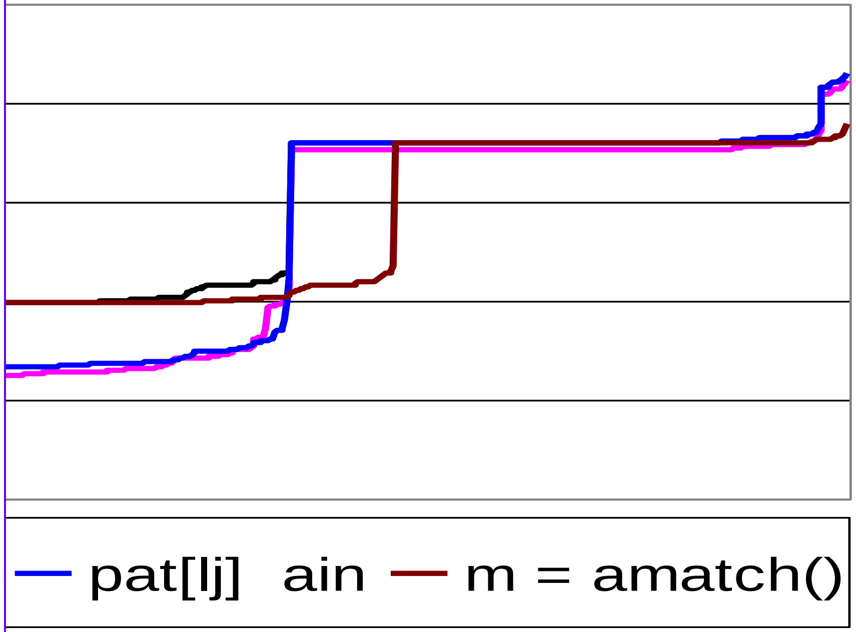
Alternate View 2 – sub-Clusters



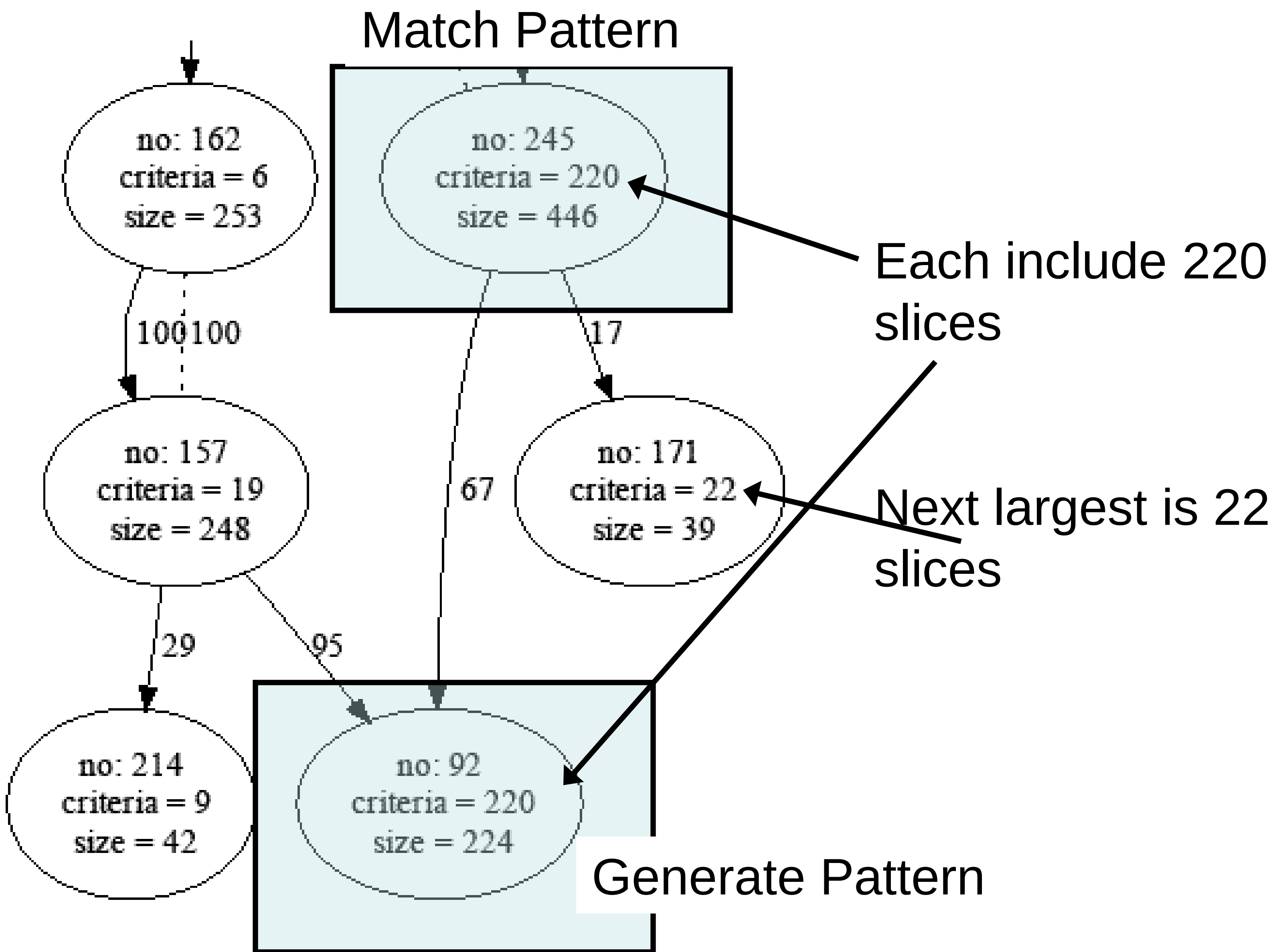


Alternate View 2 – sub-Clusters





Alternate View 2 – sub-Clusters



Remember

The definition of *Dependence Cluster*:

A set of statements where each depends on the others

Remember

The definition of *Dependence Cluster*:

A set of statements where each depends on the others

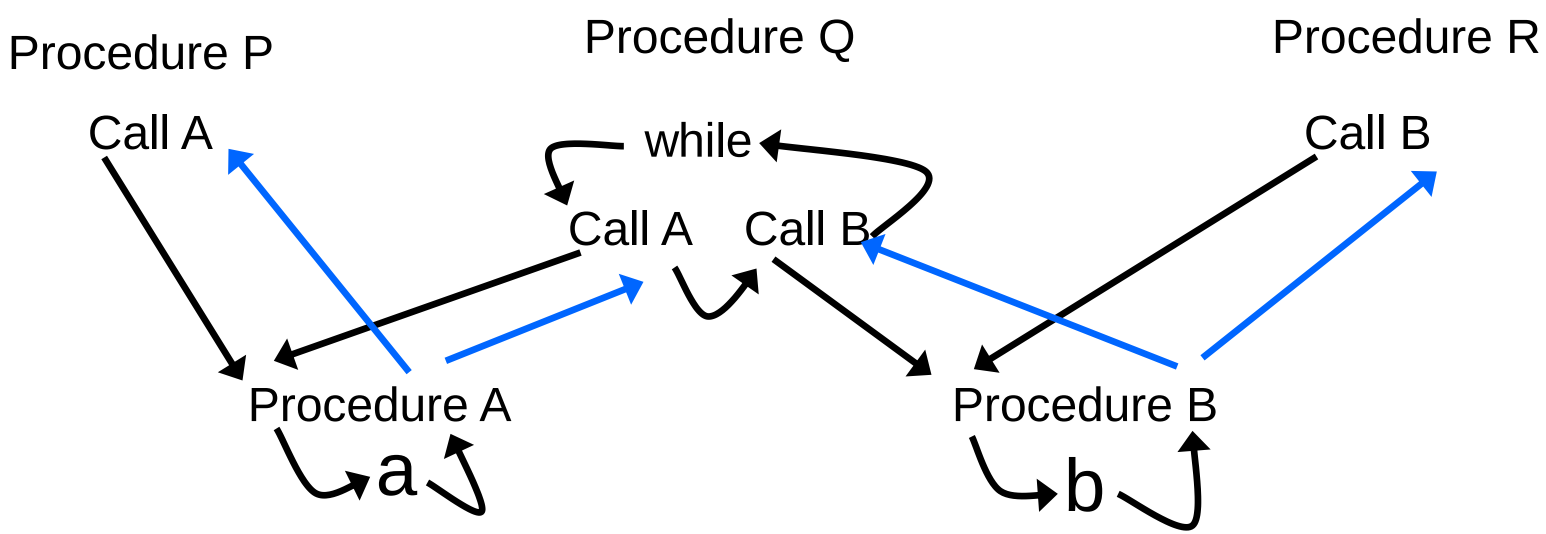
And

Algorithm 1 (an under approximation)
statements ‘s’ and ‘t’ are in a cluster if
slice(s) = slice(t)

Why it's an Under Approximation

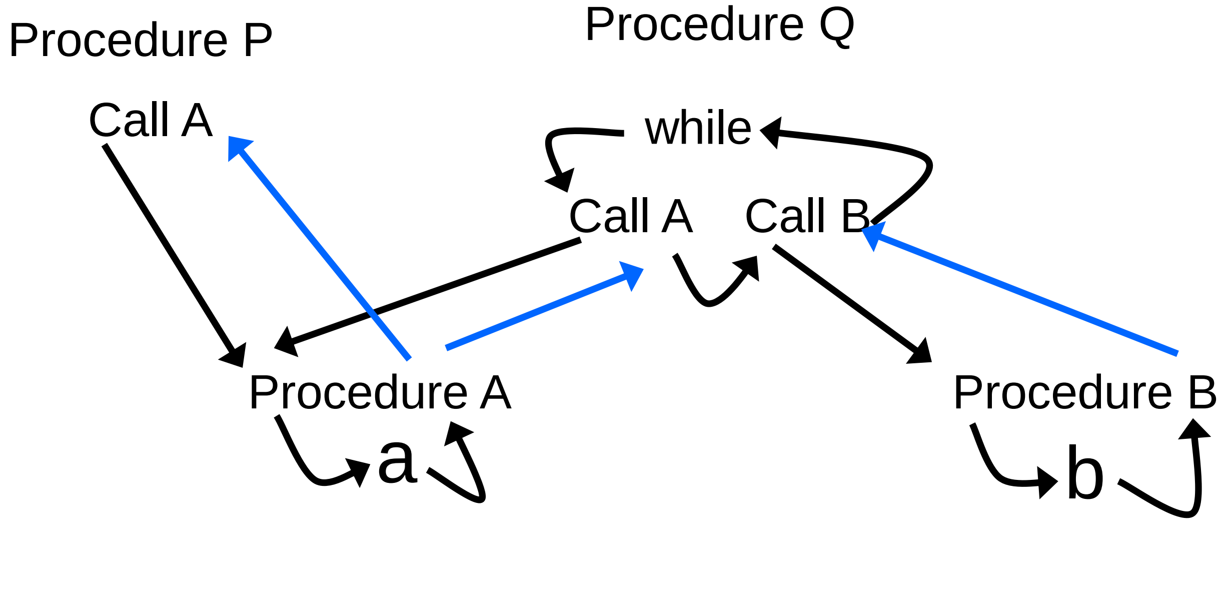
Dependence Cluster – A set of statements where each depends on the others

consider first the slice on **a**



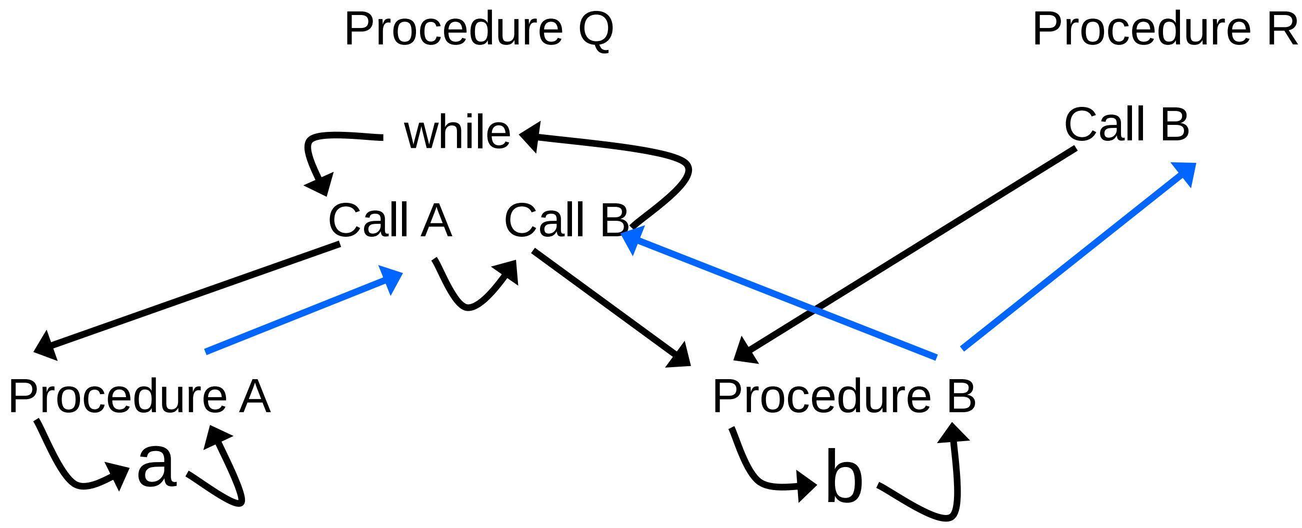
Why it's an Under Approximation

Slice on **a**



Why it's an Under Approximation

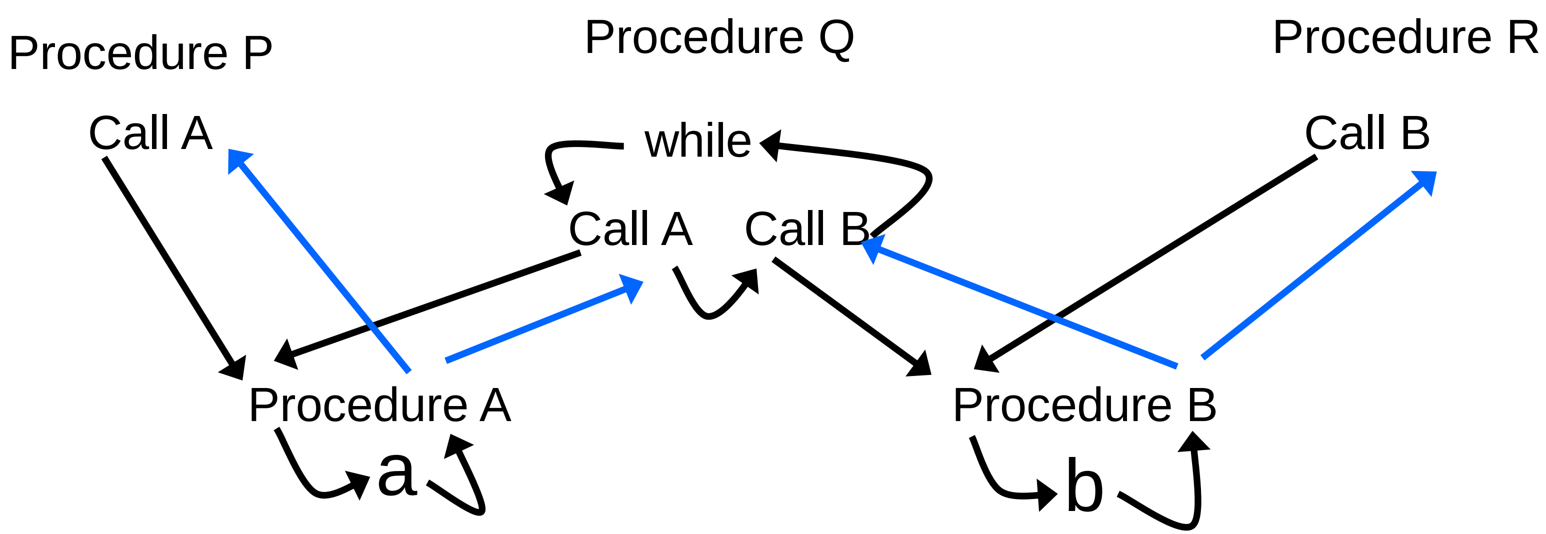
Slice on **b**



Why it's an Under Approximation

Thus a $\in$ slice(b) and b $\in$ slice(a)

but   slice(a) $\neq$ slice(b)



Algorithm 2

Dependence Cluster – A set of statements where each depends on the others

Algorithm 2
statements ‘s’ and ‘t’ are in a cluster if
$t \in$ *slice(s)* *and s* $\in$ *slice(t)*

Observation

if slice(a) = slice(b) then

$a \in$ slice(b)

$b \in$ slice(a)

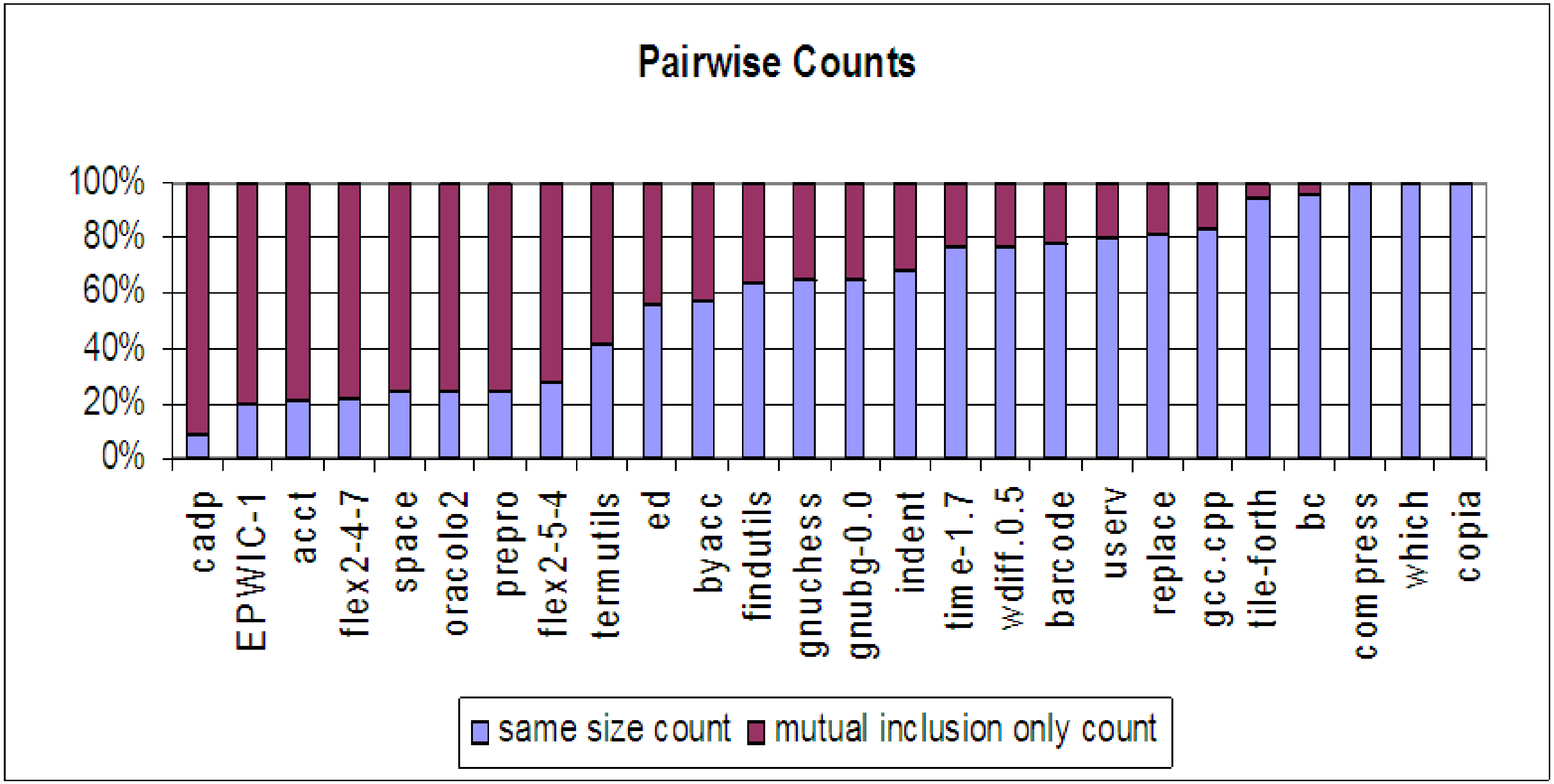
as

$a \in$ slice(a)

Thus

Algorithm 1 clusters are Algorithm 2 clusters

Do the Two Definitions Produce Different Answers?



Alas, Algorithm 2

- places a statement in multiple clusters
- is NP hard