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!
main()
{
    a = 42;
    if a > 10
    {
        b = a / 2
    }
}
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;
- }
Consider making a change to Line 42
Impact of Dependence Clusters

Consider testing Line 42

Consider reusing Line 42

Program Without Cluster

Program With Cluster
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’

\[ Cluster(s) = \{ t \in P \mid \text{slice}(t) = \text{slice}(s) \} \]

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
  \( \text{slice}(s) = \text{slice}(t) \)

• 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-$\text{size}(s) = \text{slice-size}(t)$

• Implementation 1a
  – slice on each statement   $O(n^2)$
  – partition based on size    $O(n)$
Is this a Good Approximation?

Minimum, Average, and Maximum Agreement

Agreement

Similarity

0%
20%
40%
60%
80%
100%

100.0% 99.9% 99.8% 99.7% 99.6% 99.5% 99.4% 99.3% 99.2% 99.1% 99.0%
The Approximation Yields an Interesting Visualisation

the MSG
(the Monotone Slice-size Graph)

About 20% of the slices are very small.

The last slice (the largest) includes about 55% of the program.
Cluster Containing MSG
And Another

[Graph image] copia
And Another
Do Clusters Exist? -- Yes
Visualizing Dependence Clusters
MSG for bc (basic calculator)  
(Syed Islam)
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*

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*).
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 “progra
Measuring Impact

Area under MSG

modified bc

space
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

```c
if (InStrPos(buffr, ":=") >= 0)
{
    strcpy(buffr, ReplaceS(buffr, ":=", "="));
}
```

```c
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;
-   }

Automated Breaking
Ignore dependence associated with a statement (dependence graph vertex)
<table>
<thead>
<tr>
<th>slices</th>
<th>program</th>
<th>reduction</th>
<th>vertex type</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>4686</td>
<td>copia</td>
<td>89.96%</td>
<td>control-point</td>
<td>switch (a)</td>
</tr>
<tr>
<td>1044</td>
<td>time-1.7</td>
<td>51.23%</td>
<td>control-point</td>
<td>switch (*++fmt)</td>
</tr>
<tr>
<td>1077</td>
<td>conversion</td>
<td>27.04%</td>
<td>control-point</td>
<td>switch (pick_op)</td>
</tr>
<tr>
<td>747</td>
<td>driver</td>
<td>26.45%</td>
<td>control-point</td>
<td>switch(choice)</td>
</tr>
<tr>
<td>585</td>
<td>sudoku</td>
<td>22.87%</td>
<td>control-point</td>
<td>while(!check_completed())</td>
</tr>
<tr>
<td>11277</td>
<td>space</td>
<td>7.90%</td>
<td>control-point</td>
<td>if (error != 0)</td>
</tr>
<tr>
<td>9556</td>
<td>gnubg-0.0</td>
<td>7.32%</td>
<td>indirect-call</td>
<td>pc-&gt;pf()</td>
</tr>
<tr>
<td>3909</td>
<td>barcode</td>
<td>5.95%</td>
<td>indirect-call</td>
<td>cp-&gt;encode()</td>
</tr>
<tr>
<td>12492</td>
<td>EPWIC-1</td>
<td>5.79%</td>
<td>control-point</td>
<td>while(state != DONE)</td>
</tr>
<tr>
<td>10151</td>
<td>byacc</td>
<td>1.54%</td>
<td>expression</td>
<td>k = keyword()</td>
</tr>
</tbody>
</table>
Linchpin Vertices?

Copia vs. Time

Replace
MSGs for Copia’s Top 3
Function “Confirm” from Copia

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

- 10 more comparisons ...

![Diagram showing the function flow with labels for Original, switch(s), if (geocircol ..., and formal a)]
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)|

M vertices

N vertices

Graph showing the impact as a function of time.
Declarations

Data Dependence (definition – use)

Control Dependence
Dual Path Property

dpp(\(\alpha, v, u\))

Consider slice(a)
Challenge is finding ‘u’ from $\alpha$
Challenge is finding ‘u’

Here 0 does not cut it

But 1 does as \( \text{dpp}(\alpha_1, v, u) \) holds!

3.1.2.3
Three Algorithms

- Static Depth
- Dynamic Depth
- Borderless
Static Depth
(a disaster 😞)

For depth=0 \( \text{dpp}(\alpha_1, v, u_1) \)

For depth=1 \( \text{dpp}(\alpha_2, v, u_2) \)
Dynamic Depth
way better 😊

depth varies by ‘u’
For depth=0 \( dpp(\alpha_1, v, u_1) \)
For depth=1 \( dpp(\alpha_2, v, u_2) \)
Borderless

• Original theory too weak to deal with crossing procedure borders
  – Therefore fringe (u’s) had to be in same procedure as \( \alpha \)

• 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%

Borderless

Dynamic
Time Comparison

Average Analysis Time

- borderless 1%
- borderless 10%
- borderless 20%
- dynamic K 1%
- dynamic K 10%
- dynamic K 20%
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

Excluded comparison

- blue line: borderless 1%
- red line: borderless 10%
- green line: borderless 20%
- cyan line: dynamic K 1%
- orange line: dynamic K 10%
- yellow line: dynamic K 20%
Time Comparison

Average Analysis Time (percent improvement)
Time and Work Together - Dynamic

Threshold = 1%  Threshold = 10%  Threshold = 20%
Excluded vertices (those having dual paths)
Percent improvement in excluded vertices
Semantics-preserving Refactoring
Alternate Views - Histogram
EPWIC Histogram
Alternate View 2 – sub-Clusters
Alternate View 2 – sub-Clusters

- no: 346
  criteria: 2
  size: 548

- no: 341
  criteria: 2
  size: 538

- no: 279
  criteria: 3
  size: 451

- no: 245
  criteria: 220
  size: 446

- no: 171
  criteria: 22
  size: 39

- no: 214
  criteria: 9
  size: 42

- no: 93
  criteria: 220
  size: 224

- pat[lj] ain
- m = amatch()
Alternate View 2 – sub-Clusters

Match Pattern

Each include 220 slices
Next largest is 22 slices

Generate Pattern
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

\[ \text{slice}(s) = \text{slice}(t) \]
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 $a \in \text{slice}(b)$ and $b \in \text{slice}(a)$
but $\text{slice}(a) \neq \text{slice}(b)$
Algorithm 2

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

Algorithm 2


\[
\text{statements ‘s’ and ‘t’ are in a cluster if } \quad t \in \text{slice}(s) \text{ and } s \in \text{slice}(t)
\]
if \( \text{slice}(a) = \text{slice}(b) \) then

\[
 a \in \text{slice}(b) \\
 b \in \text{slice}(a)
\]

as

\[
 a \in \text{slice}(a)
\]

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