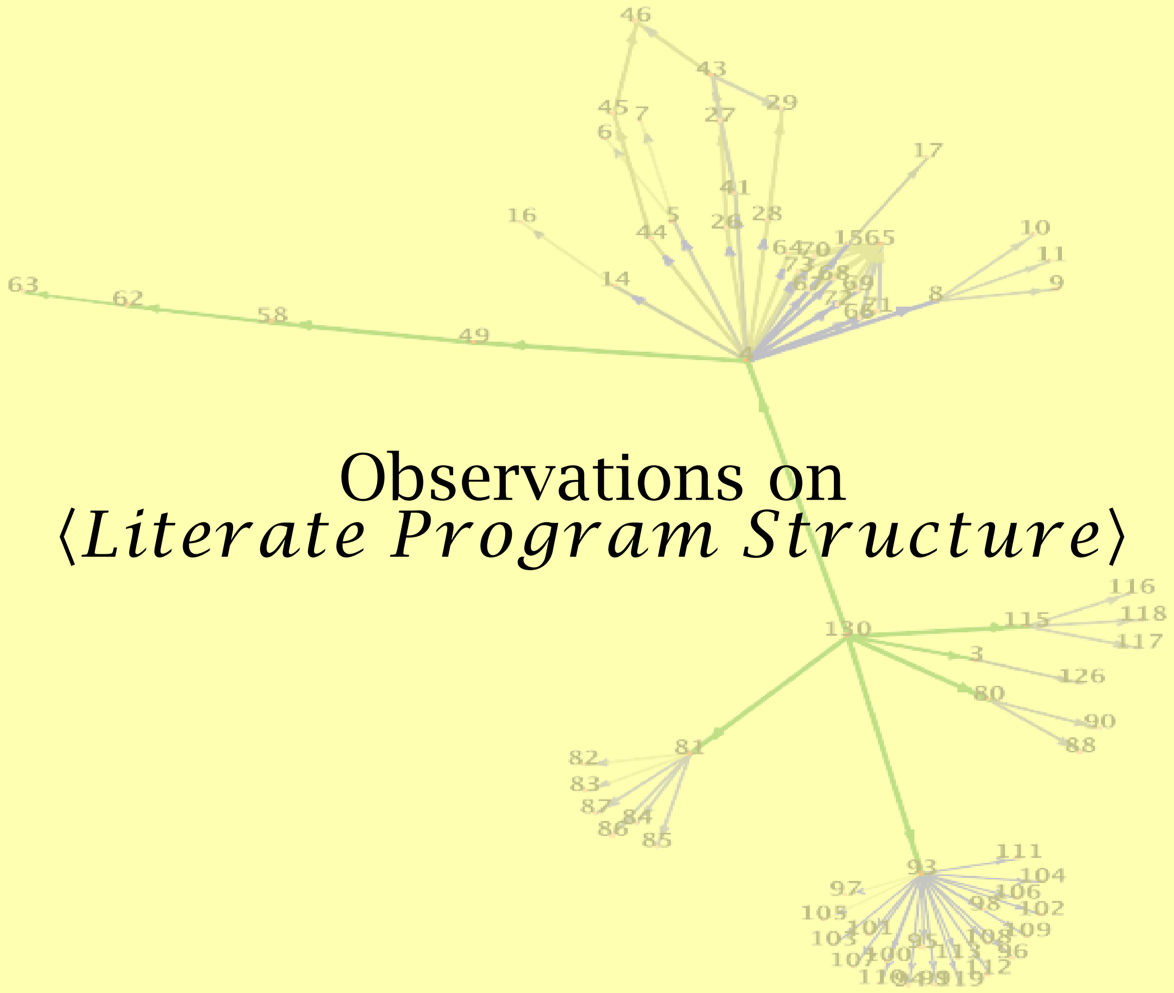


Observations on ⟨*Literate Program Structure*⟩

The Challenge

If it is true, as Professor Knuth says, that a literate program is a “web” of interconnections, can the pattern of connections particular to literate programs be *modeled*, *analyzed*, and *interpreted*?

- What is an appropriate representation for the structure of literate programs?
- What measurements can we extract from our representation?
- How do we interpret what we measure?
- Can we compare the structure of literate programs to the call graph structure of traditional programs?

Literate Programming Review

Characteristics of a Literate Program

Appearance of a Literate Program

“A complex piece of software consists of simple parts and simple relations between those parts; the programmer’s task is to state those parts and those relationships, in whatever order is best for human comprehension.” – D.E.K.

Characteristics of a Literate Program

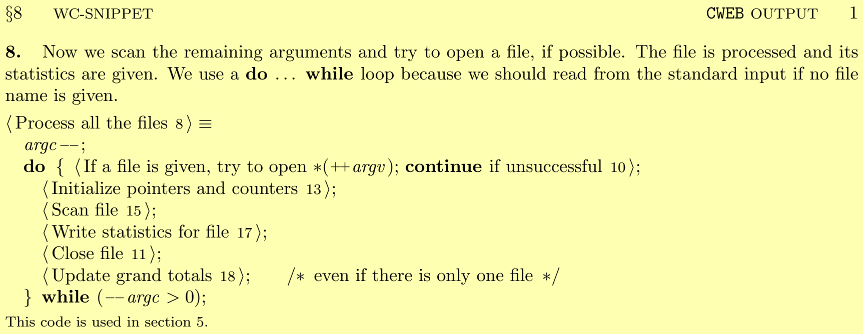
Literate programs are written with an emphasis on *exposition*. They are differentiated from programs written in other programming styles by three characteristics:

- A single literate source file produces both a compiled program and a nicely typeset document.
- The order of presentation is independent of the order of compilation.
- Cross-references, indices, and navigational hints are automatically generated.

Appearance of a Literate Program

Here is a section of a literate source file and its typeset result.

```
@ Now we scan the remaining arguments and try to open a file, if
possible.  The file is processed and its statistics are given.
We use a |do|~\dots~|while| loop because we should read from the
standard input if no file name is given.
@<Process...@>=
argc--;
do@+{
  @<If a file is given, try to open |*(++argv)|; |continue| if unsuccessful@>;
  @<Initialize pointers and counters@>;
  @<Scan file@>;
  @<Write statistics for file@>;
  @<Close file@>;
  @<Update grand totals@>; /* even if there is only one file */
}@+while (--argc>0);
```



§8 WC-SNIPPET CWEB OUTPUT 1

8. Now we scan the remaining arguments and try to open a file, if possible. The file is processed and its statistics are given. We use a **do** ... **while** loop because we should read from the standard input if no file name is given.

⟨Process all the files 8⟩ ≡
  argc −−;
  do { ⟨If a file is given, try to open $*(\mathord{+}\mathord{+}\,argv)$; **continue** if unsuccessful 10⟩;
    ⟨Initialize pointers and counters 13⟩;
    ⟨Scan file 15⟩;
    ⟨Write statistics for file 17⟩;
    ⟨Close file 11⟩;
    ⟨Update grand totals 18⟩; /∗ even if there is only one file ∗/
  } **while** $(-- argc > 0)$;

This code is used in section 5.

Compare & Contrast

The programs chosen for this study were the thirteen demonstration programs and the eighteen library modules that together comprise *The Stanford GraphBase*.

Literate programs:

- The unit of oganization is a *section* – analogous to a paragraph in prose.
- The organizing principle is clarity of *exposition*.
- The scope of a literate program written in **cweb** is a *single file*.

Traditional programs:

- The unit of organization is a *subroutine*.
- The organizing principle is *execution*, i.e. a hierarchy of subroutine calls.
- The scope of a traditional program is an *executable*.

GraphXML

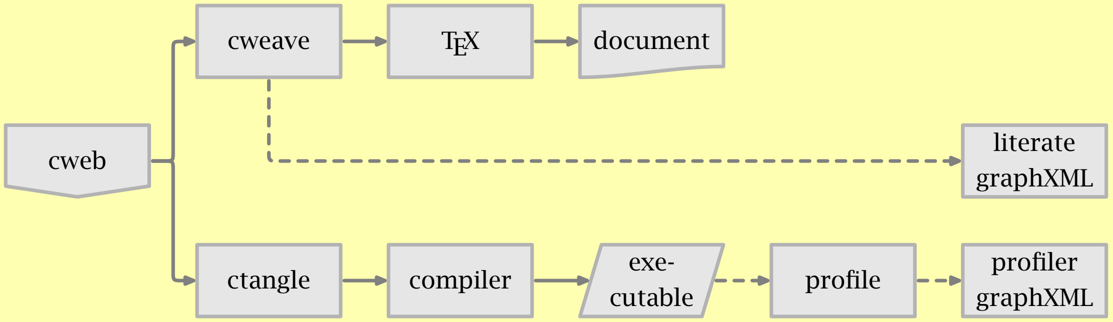
The structure of literate programs and the call graphs of traditional programs are represented as *directed acyclic graphs* (DAGs).

GraphXML is a graph description language in XML, whose goal is to provide a general interchange format for graph drawing and visualization systems.

```
<?xml version="1.0"?>
<!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd">
<GraphXML>
  <graph>
    <node name="first"/>
    <node name="second"/>
    <edge source="first"  target="second"/>
  </graph>
</GraphXML>
```

Extracting the Graphs

- The literate program graphs were extracted from auxiliary files created by **cweave.**
- The call graphs were extracted from profiling information created by **gprof.**



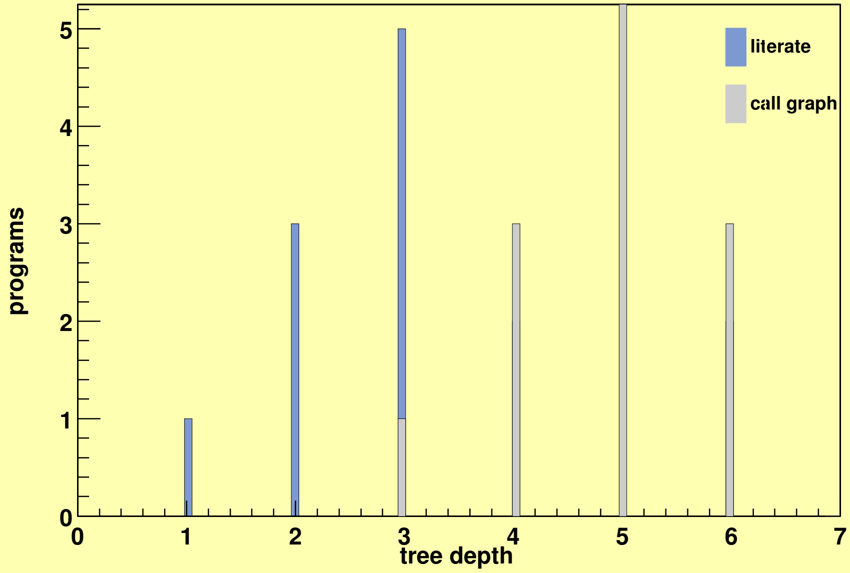
Structure Metrics

A *metric* is a measure of some aspect of the graph that is associated its nodes or edges. A metric can consider the graph's structure, include domain-specific information, or a combination of both. A metric is *structure-based* if it only uses information about the structure of the graph.

- Tree Depth
- Tree Impurity
- Spreading Activation
- Recursive Citation

Tree Depth

Tree Depth counts how many nodes lie along the longest path from the root to a leaf. We expected that call graphs would be deeper than their literate counterparts. The average literate depth is 3.23, whereas the average call graph depth is 4.85.



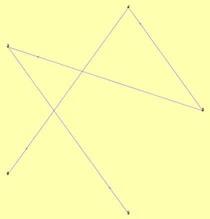
Tree Impurity

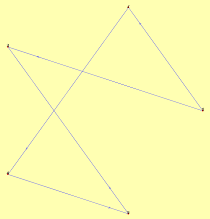
Tree Impurity measures the degree to which a graph's structure deviates from being a pure tree, i.e. no cycles. Its value is considered inversely proportional to the quality of the structure's design.

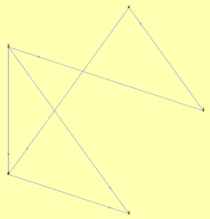
Tree Impurity in Theory

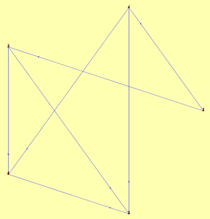
A graph's *Tree Impurity* value ranges between zero and one

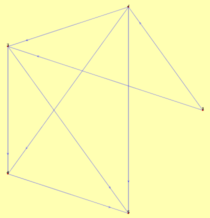
where: $m(G) = \frac{\textit{number of edges more than the spanning tree}}{\textit{maximal number of edges more than the spanning tree}} = \frac{2(e-n+1)}{(n-1)(n-2)}$

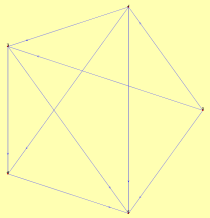

0

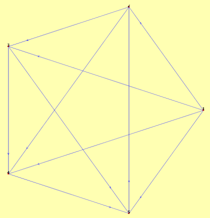

1/6


1/3


1/2

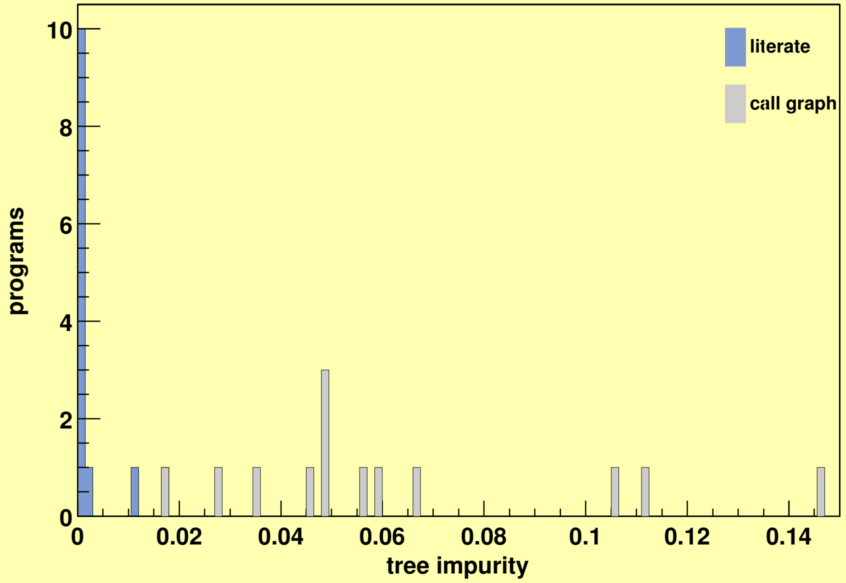

2/3


5/6


1

Tree Impurity in Practice

It appears that the literate organizations are very clean compared to the call graphs of their corresponding implementations. The average literate impurity is 0.00246, whereas the average call graph impurity is 0.06292.



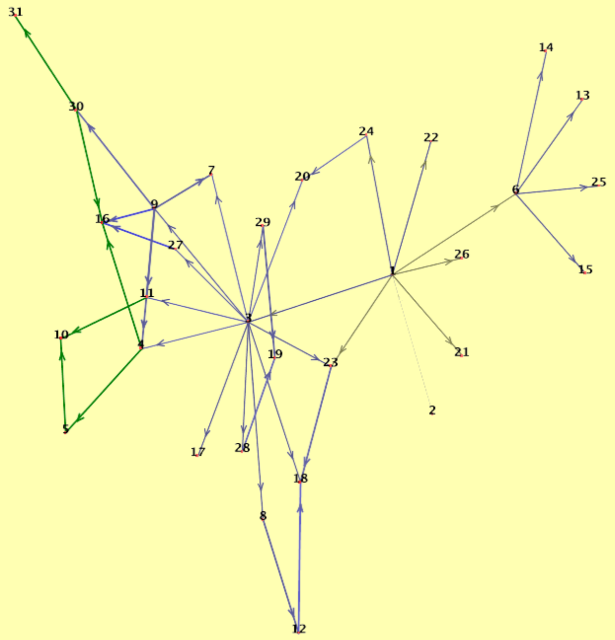
Word Ladders

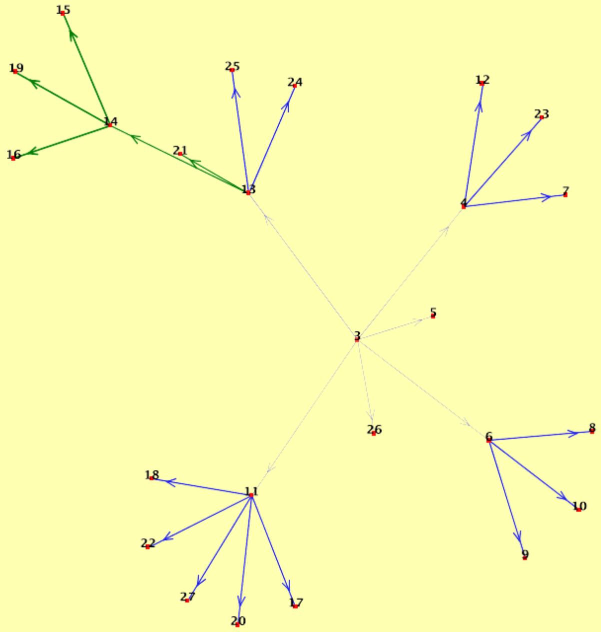
Recursive Citation
Spreading Activation

The next two metrics, *Recursive Citation* and *Spreading Activation*, are from the *Stanford GraphBase* example program named *ladders*, which first produces a graph of five-letter words, and then uses Dijkstra's algorithm to find the shortest path between a pair of word solicited from the user.

Recursive Citation

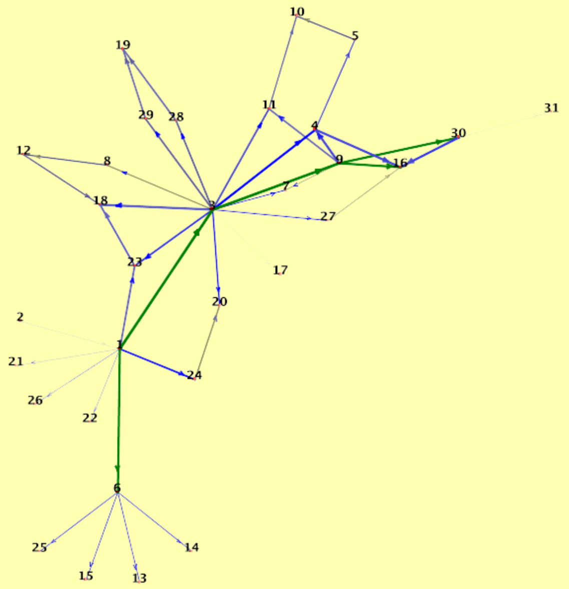
Recursive Citation is a measure of how many times a node is referenced from above. This measure reveals the most heavily referenced nodes.

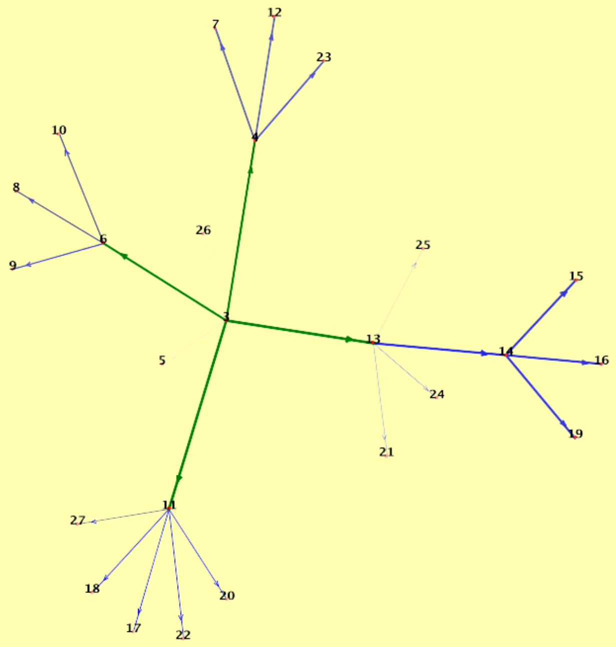




Spreading Activation

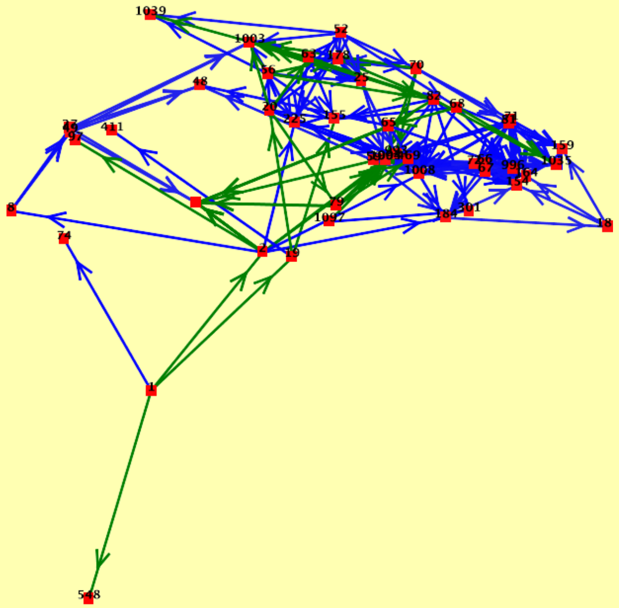
Spreading Activation is a measure of global connectivity. The value for a node is obtained by summing the contributions over the set of all its neighbors $n_1, \ldots, n_q (q \geq 1)$. This measure reveals a graph's *spine* or *center of mass*.

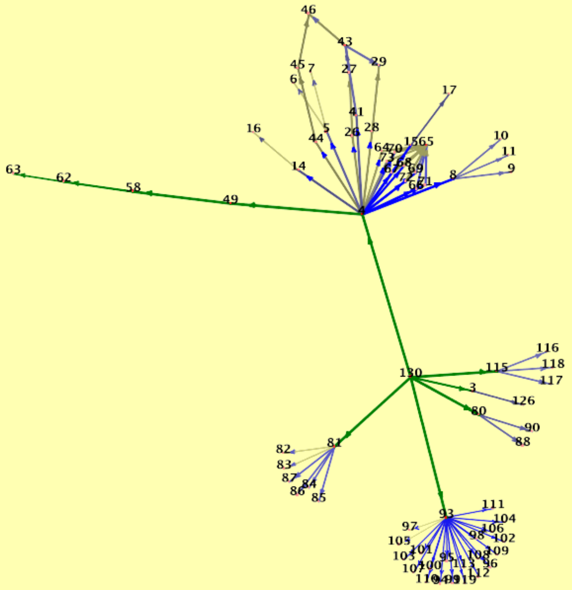




A Real Live Example

This is a visualization of the Taxiway Colision Monitor (TCM) employing the *Spreading Activation* metric. The literate graph has 124 nodes and 134 edges. The call graph has 851 nodes and 1493 edges.





Conclusions

- The fundamental difference between literate programs and their traditional counterparts is that the former are hierarchies of ideas *(exposition)*, while the latter are hierarchies of subroutine calls *(execution)*.
- The literate hierarchies appear routinely to be cleaner and less deep then the corresponding call graph hierarchies.
- The metrics for traditional programs would be enhanced by the addition of domain specific information such as: execution time, execution counts, etc.
- Lots of empirical evidence is needed to make sense of these metrics in practice.

Future Work

- Metrics like *tree impurity* should be calculated for each subtree in the graphs.
- the **gprof** profiler generates more data than we currently use. Information about execution time and execution counts could become edge attributes used to reveal the most often used or most expensive portions of a program.
- Empirical evidence needs to be gathered to know how to interpret structure metrics together with other software metrics.
- The **noweb** literate programming system can process multiple literate source files. It would be interesting to investigate larger literate systems comprised of many individual literate modules, e.g. libraries.

References

Here are a few links to the software used in the preperation of this presentation.

- The CWEB System: http://www-cs-staff.Stanford.EDU/ knuth/cweb.html
- The Stanford GraphBase: http://www-cs-staff.Stanford.EDU/ knuth/sgb.html
- Literate Programming Website: http://www.literateprogramming.com/
- The Royere Graph Visulation Software: http://www.cwi.nl/InfoVisu/