Graph-Theoretic Analysis of the Relationships in Discrete Data

Mathematics Honours Thesis

Ivan Lazar Miljenovic

Department of Mathematics, University of Queensland

6 October, 2008

Thank you to:

Supervisor Professor Peter Adams

Second Reader Andriy Kvyatkovskyy

My family For putting up with me

The people on #haskell For the programming assistance

About my thesis

The Information Age

- We live in the Information Age
- Can process large amounts of raw data
- We do so with *maths*!

Data Processing

Type of Processing	Mathematics Usage
Databases	Relational Algebra
Numeric Calculations	Linear Algebra
Correlations, etc.	Statistics
Computer Science	Originally *all* math!

Discrete Data

- Data that is made up of discrete pieces of information
- Have inherent *relationships* that characterise the data
- Use *Graph Theory* to analyse these relationships

Goal of my thesis

1. To develop a software library to aid programmers in utilising graph theory to analyse the relationships in their discrete data: GRAPHALYZE
2. To use this library to analyse the static complexity of source code: SOURCEGRAPH

Previous work

- Network Theory/Analysis
- Program and library visualisation
- Graph-based optimisations

Graph Theory

2. Graph Theory
   - Non-general definitions

Non-general definitions

- I use the term *Node* rather than *Vertex*
- Empty graphs allowed
- A *Clustering* is a logical grouping of nodes in a graph.

Implementation Specifics

Haskell

- Possibly the only language successfully designed by a committee
- *"...a pure functional programming language with non-strict semantics."*

Graph Implementation

- Most graph libraries use node/edge lists or some type of matrix.
- *Inductive Graphs* provide a much more applicable representation.

Algorithm Implementation

- Easy to read
- Follow the graph structure
- Graph-invariant
- No outside input
- Most are from scratch

Code produced

- Worked on the *graphviz* library
- The Graphalyze library
- The SourceGraph program

Analysing Source Code

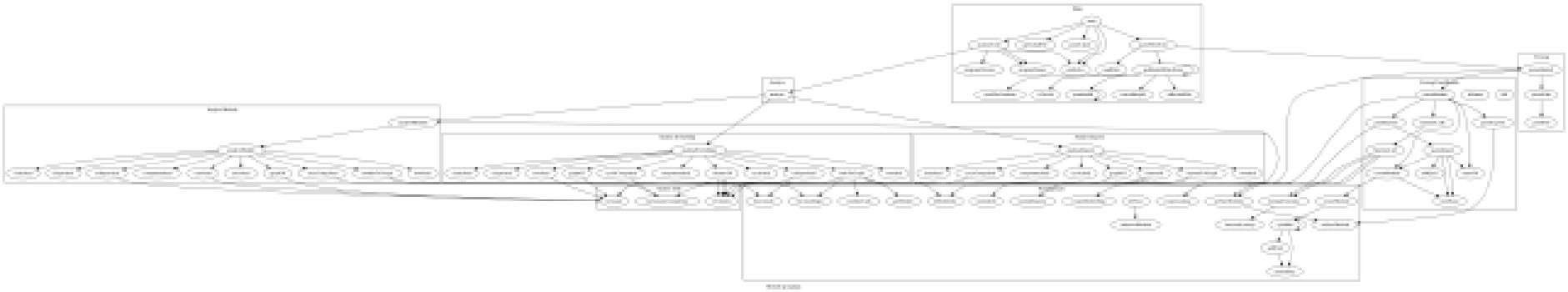
4. Analysing Source Code
   - Haskell code structure
   - Sample Analysis

Haskell code structure

- Lots of small functions
- Higher number of function calls
- Higher function re-use
- Recursion

SourceGraph

Original files:



Suggested files:

Where to from here?

- Better report generation
- More customizable analysis
- Extend language support
- Improve efficiency