Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification

Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, Václav Rajlich

14th IEEE International Conference on Program Comprehension (ICPC’06)

Athens, Greece
Motivation

• Feature/concept identification (location)

• Concept location – identifying parts of the source code implementing domain concepts

• Reduces search space

• Uses static and/or dynamic analysis
Concept Location in Practice

- **Static**
  - Dependency based search [Rajlich’00]
  - IR methods [Marcus’04]

- **Dynamic**
  - Execution traces - Reconnaissance [Wilde’92]
  - Scenario based probabilistic ranking [Antoniol’05]

- **Combined**
  - Profiling with concept analysis [Eisenbarth’03]
  - Feature dependencies [Salah’05]
  - Feature evolution [Greevy’05]
Shortcomings

• Static analysis:
  – sometimes does not identify all entities implementing a specific concept
  – recall is impacted

• Dynamic analysis:
  – sometimes unable to distinguish between overlapping features
  – precision is impacted
Our Combination

• Static
  – Dependency based search [Rajlich’00]
  – IR methods [Marcus’04]

• Dynamic
  – Execution traces - Reconnaissance [Wilde’92]
  – Scenario based probabilistic ranking [Antoniol’05]

• Combined
  – Profiling with concept analysis [Eisenbarth’03]
  – Feature dependencies [Salah’05]
  – Feature evolution [Greevy’05]
Novel Hybrid Technique

• Feature identification – decision making problem in presence of uncertainty

• Static (LSI) and dynamic (SBP) experts:
  – LSI queries static documents
  – SBP analyzes dynamic traces of execution scenarios

• Complementary results are combined via affine transformation
Scenario Based Probabilistic Ranking

• Building a model of a program architecture

• Identifying a feature of interest
  – Subset of a program architecture (micro-architectures -> variables, classes, functions, methods)

• Comparing features modeled as micro-architectures
SBP - Feature Identification

• Program model creation
  – static analysis, C++, AOL

• Trace collection
  – (ir) relevant scenarios are executed to collect traces
  – processor emulation (VALGRIND) to improve the precision of data collection

• Knowledge-based filtering

• Probabilistic ranking
  – events are re-weighted (Wilde’s equation is renormalized)
Latent Semantic Indexing

- Vector space model based IR method
  [Dumais’94, Berry’95, Deerwester’90]

- Applied to text retrieval, pattern recognition, natural language understanding

- Known application: Google
Concept Location with LSI

- User defined queries
  - Based on user experience and domain knowledge, little known about querying patterns
- Semi-automated query generation
  - Starts with a user defined query and adds synonyms from the source code, identified by LSI
Combining the Experts

- SBP and LSI – our experts
- SBP – constructing overlapping scenarios
- LSI – formulate a query that captures semantic characteristics of the feature
- Combining judgments of experts :

\[ r_{combined}(x) = \lambda r_{sbp}(x) + (1 - \lambda) r_{coli}(x) \]
Case Study Objectives

• Assess the precision of the novel hybrid technique

• Compare hybrid technique with standalone results for SBP and LSI

• Evaluate the influence of dimensionality reduction factor on the corpus size
Case Study – Mozilla Sizes

- Mozilla v1.6 size related statistics

<table>
<thead>
<tr>
<th>Item</th>
<th>Count (MLOC)</th>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header files</td>
<td>8,055 (1.50)</td>
<td>Classes</td>
<td>4,853</td>
</tr>
<tr>
<td>C files</td>
<td>1,762 (0.90)</td>
<td>Methods</td>
<td>53,617</td>
</tr>
<tr>
<td><strong>C++ files</strong></td>
<td>4,204 (2.00)</td>
<td>Specializations</td>
<td>5,314</td>
</tr>
<tr>
<td>IDL files</td>
<td>2,399 (0.20)</td>
<td>Associations</td>
<td>17,362</td>
</tr>
<tr>
<td>XML files</td>
<td>283 (0.12)</td>
<td>Aggregations</td>
<td>6,727</td>
</tr>
<tr>
<td>HTML files</td>
<td>2,231 (0.19)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java files</td>
<td>56 (0.06)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
First Case Study

- Feature: “Add a bookmark in Mozilla”

- Find the methods and functions, which implement the feature in Mozilla

- Replicated case study to compare with previous results
SBP Results

- Scenario 1: “A user visits an URL, opens Mozilla, clicks on bookmarked URL, loads page and closes Mozilla”

- Scenario 2: “The user acts like in Scenario 1, but once the page is loaded, she saves URL”

- SBP provides 274 methods ranked with probability of 1.0
# LSI Results

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MLOC</strong></td>
<td>4.4</td>
</tr>
<tr>
<td><strong>Vocabulary</strong></td>
<td>85,439</td>
</tr>
<tr>
<td><strong>Number of parsed documents</strong></td>
<td>68,190</td>
</tr>
<tr>
<td><strong>Number of methods</strong></td>
<td>48,267</td>
</tr>
<tr>
<td><strong>Number of functions</strong></td>
<td>19,923</td>
</tr>
</tbody>
</table>

- **LSI Query:** “*bookmark newbookmark bookmarkname bookmarkresource bookmarkadddate createbookmark insertbookmarkitem deletebookmark bookmarknode*”
## Combined Results

<table>
<thead>
<tr>
<th>R</th>
<th>300</th>
<th>500</th>
<th>750</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CreateB (3)</td>
<td>CreateB (6)</td>
<td>AddB (1)</td>
<td>AddB (1)</td>
</tr>
<tr>
<td>2</td>
<td>AddB (4)</td>
<td>AddB (2)</td>
<td>CreateB (14)</td>
<td>CreateB (8)</td>
</tr>
<tr>
<td>3</td>
<td>CreateBC (64)</td>
<td>Flush</td>
<td>Flush</td>
<td>CreateBC (19)</td>
</tr>
<tr>
<td>4</td>
<td>InsertResource</td>
<td>CreateBC (57)</td>
<td>CreateBC (36)</td>
<td>WriteBookmarks</td>
</tr>
<tr>
<td>5</td>
<td>ListenToEventQueue</td>
<td>InsertResource</td>
<td>WriteBookmarks</td>
<td>getFolderViaHint</td>
</tr>
<tr>
<td>6</td>
<td>Flush</td>
<td>WriteBookmarks</td>
<td>Observe</td>
<td>InsertResource</td>
</tr>
</tbody>
</table>

CreateB – CreateBookmark
AddB – AddBookmarkImmediately
CreateBC - CreateBookmarkInContainer
A Bug / Unwanted Feature

• Bug # 182192 from BugZilla: “quotes (") are not removed from collected e-mail addresses”

• From: "First Last" <first.last@example.org>
  – First: "First
  – Last: Last“
  – Difficult to search the address book

• We use official Bugzilla reports to verify the results: CollectAddress and CollectUnicodeAddress are fixed
SBP Results

- Scenario 1: “A user replies to an e-mail”

- Scenario 2: “A user performs the same action as in Scenario 1 and, using the mouse, the user forces to collect e-mail address of the sender”

- SBP returned 206 methods with score of 1.0
Combined Results

• LSI query: “collect collected sender recipient email name names address addresses addressbook”

<table>
<thead>
<tr>
<th>Rank</th>
<th>Method name</th>
<th>Rlisi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ParseHeadersWithArray</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>ParseHeaderAddresses</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>CollectAddress</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>OpenInternal</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>CollectUnicodeAddress</td>
<td>46</td>
</tr>
</tbody>
</table>
Discussion

• Combination of SBP and LSI is better than SBP and LSI standalone

• The results tend to improve when increasing the dimensions of LSI space

• The case studies reveal great potential of this combination
Future Work

- Combine with other feature location techniques
  - dependency search
  - clustering

- Determine heuristics to identify the best $\lambda$