Fine-grained and Accurate Source Code Differencing: GumTree
ASE 2014. (citation#;46)

Kijin An, CS6704
Introduction

• Unix diff tool takes as input two versions of a source code file and performs the Myers algorithm [24]

• Limitations of diff like tools
  • First, it only computes additions and deletions and does not consider actions such as update and move
  • Second, it works at a granularity (the text line) that is both coarse grain and not aligned with the source code structure: the abstract syntax tree (AST)
Examples

Diff tools over Java codes

AST differencing tools over Java codes

goals: to find a sequence “edit actions” in AST levels
Edit Actions in AST Differencing

• Abstract Syntax Tree (AST)
  • Labeled ordered rooted tree: labels and values
    • Label: the name of their production rule in the grammar

  ```
  compilationUnit
  /   \
  variable    print
  /   \
  var x = value print x var dupa = value print dupa

  5
  “dupa”
  ```

• Four AST edit actions in AST Differencing
  • \textit{updateValue}(t, v_n)
  • \textit{add}(t, t_p, i, l, v)
  • \textit{delete}(t)
  • \textit{move}(t, t_p, i)
Edit Actions in AST Differencing

• Four AST edit actions
  • \textit{updateValue}(t, v_n) : replaces the old value of t by the new value $v_n$
  • \textit{add}(t, t_p, i, l, v) : adds a new node t in the AST. If $t_p$ is not null and $i$ is specified then t is the $i^{th}$ child of $t_p$. Finally, l is the label of t and v is the value of t
  • \textit{delete}(t) : removes a leaf node of the AST
  • \textit{move}(t, t_p, i) : moves a node t and make it the $i^{th}$ child of $t_p$. Note that all children of t are moved as well, therefore this actions moves a whole subtree
Edit Actions in AST Differencing

• Edit Actions Examples

Test.java (source)

```java
public class Test {
    public String foo(int i) {
        if (i == 0) return "Foo!";
    }
}
```

Test.java (destination)

```java
public class Test {
    private String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Foo!";
    }
}
```

• Which one is
  • `updateValue`?
  • `add`?
  • `move`?
  • `remove`?
Overall GumTree Algorithm

- AST differencing algorithms work in two steps:
  - 1. Establishing **mappings** of **src** and **dst**
    > only explain in detail how we look for the mappings between two ASTs
  - 2. then deducing an **edit script**
    > an optimal and quadratic algorithm has already been developed by the Chawathe et al. [6]
GumTree Algorithm for mapping

- GumTree **Mapping** Algorithms
  - A greedy **top-down** algorithm
  - A **bottom-up** algorithm
  - **recovery mappings**
GumTree Algorithm for mapping

- A greedy top-down algorithm: anchors mappings
- A bottom-up algorithm: called a container mapping
- Recovery mappings: to search for additional mappings among their descendants

- First they search for the biggest unmodified pieces of code
- Then they deduce which container of code can be mapped together
- Finally they look at precise differences in what is leftover in each container
AST Differencing

1) Top-down
2) Bottom-up
3) Recovery
AST Differencing mapping with top-down

- Top-down
  - start with the roots and to check if they are isomorphic (or identical)
  - If they are not, their children are then tested.

(descendents of these nodes are also mapped but it is omitted to enhance readability)

\[ \text{minHeight} = 2 \]
AST Differencing mapping with top-down

Algorithm 1: The algorithm of the top-down phase.

Data: A source tree $T_1$ and a destination tree $T_2$, a minimum height $minHeight$, two empty height-indexed priority lists $L_1$ and $L_2$, an empty list $A$ of candidate mappings, and an empty set of mappings $\mathcal{M}$

Result: The set of mappings $\mathcal{M}$

1. $push(root(T_1), L_1)$;
2. $push(root(T_2), L_2)$;
3. while $min(peekMax(L_1), peekMax(L_2)) > minHeight$ do
   4. if $peekMax(L_1) \neq peekMax(L_2)$ then
      5. if $peekMax(L_1) > peekMax(L_2)$ then
         6. foreach $t \in pop(L_1)$ do open($t$, $L_1$);
      7. else
         8. foreach $t \in pop(L_2)$ do open($t$, $L_2$);
   9. else
      10. $H_1 \leftarrow pop(L_1)$;
      11. $H_2 \leftarrow pop(L_2)$;
      12. foreach $(t_1, t_2) \in H_1 \times H_2$ do
         13. if isomorphic($t_1, t_2$) then
            14. if $\exists t_3 \in T_2 \mid isomorphic(t_1, t_3) \land t_3 \neq t_2$
               or $\exists t_3 \in T_1 \mid isomorphic(t_3, t_2) \land t_3 \neq t_1$
               then
                  15. add($A$, $(t_1, t_2)$);
               else
                  16. add all pairs of isomorphic nodes of $s(t_1)$
                     and $s(t_2)$ to $\mathcal{M}$;
            17. foreach $t_1 \in H_1 \mid (t_1, t_2) \notin A \cup \mathcal{M}$ do open($t_1$, $L_1$);
            18. foreach $t_2 \in H_2 \mid (t_2, t_2) \notin A \cup \mathcal{M}$ do open($t_2$, $L_2$);
      19. sort $(t_1, t_2) \in A$ using dice(parent($t_1$), parent($t_2$), $\mathcal{M}$);
      20. while $size(A) > 0$ do
         21. $(t_1, t_2) \leftarrow remove(A, 0)$;
         22. add all pairs of isomorphic nodes of $s(t_1)$ and $s(t_2)$ to $\mathcal{M}$;
         23. $A \leftarrow A \setminus \{(t_1, t_2) \in A\}$;
         24. $A \leftarrow A \setminus \{(t_2, t_2) \in A\}$;
AST Mapping with bottom-up and Recovery

Bottom-up

Recovery

```
public class Test {
    private String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Fool";
    }
}
```

```
public class Test {
    public String foo(int i) {
        if (i == 0) return "Bar";
        else if (i == -1) return "Fool";
    }
}
```
Tree edit distance Problems

• Between ordered labeled trees
• Minimum-cost sequence of node edit operations that transform one tree into another

Figure 1: Example trees and edit operations.
add($t_1, a, 1, \text{ReturnStatement}, \epsilon$)
add($t_2, t_1, 0, \text{StringLitteral}, \text{Bar}$)
add($t_3, a, 2, \text{IfStatement}, \epsilon$)
add($t_4, t_3, 0, \text{InfixExpression}, ==$)
add($t_5, t_4, 0, \text{SimpleName}, i$)
add($t_6, t_4, 1, \text{PrefixExpression}, -$)
add($t_7, t_6, 0, \text{NumberLiterral}, 1$)
move($b, t_3, 1$)
updateValue($c, \text{private}$)
pipe and filter architecture

- Framework (abstract modules in grey)

Java (using the Eclipse JDT parser),
JavaScript (using the Mozilla Rhino parser),
R (using the FastR parser [17]),
C (using the Coccinelle parser [26])

- Antlr parsers

GumTree
ChangeDistiller [13]
XYDiff [8]
RTED [27]

XML, JSON,.., Visualization (Web export)
pipe and filter architecture

• Benefit of Antlr support
Runtime Performances

• For Java parser
  • Jenkins 1.509.4 → 1.532.2 where we extracted 1,144 modifications

• For JavaScript parser
  • JQuery 1.8.0 → 1.9.0 where we extracted 650 modifications

• Comparisons
  • A classical text diff tool: lower bound, very fast
  • GumTree: a worst-case complexity of $O(n^2)$
    • the isomorphism test they use is in $O(1)$ thanks to hashcodes
  • RTED algorithm: upper bound, has a cubic worst-case complexity ($n^3$)
Discussion Question?

• Why they did NOT compare GumTree to other tree differencing algorithms (changeDistiller) in running time performance?
Runtime Performances

• Distribution of the running time ratios of the tools
Evaluation

• RQ1)
  • Does *GumTree* produce tree differences that are correct and better than *Unix diff*?

• RQ2)
  • Does *GumTree* maximize the number of mappings and minimize the edit script size compared to *the existing algorithms*?

• RQ3)
  • Does *GumTree* detect move operations better than *ChangeDistiller*?
Evaluation (RQ1)

- Manual Evaluation

- Outputs from *unix diff* and *gumtree* approaches are given to a human evaluator

- The 144 evaluation items were independently evaluated by **three authors of this paper** called the raters. All 3 raters evaluated all the edit scripts of 144 file pairs at the AST and line level (i.e. 288 outputs). This makes a total of $3 \times 2 \times 144 = 864$ ratings.

- Table 1: Agreements of the manual inspection of the 144 transactions by three raters for Question #1 (top) and Question #2 (bottom).

<table>
<thead>
<tr>
<th></th>
<th>Full (3/3)</th>
<th>Majority (2/3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 GT does good job</td>
<td>122</td>
<td>137</td>
</tr>
<tr>
<td>#1 GT does not good job</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>#1 Neutral</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>#2 GT better</td>
<td>28</td>
<td>66</td>
</tr>
<tr>
<td>#2 Diff better</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>#2 Equivalent</td>
<td>45</td>
<td>61</td>
</tr>
</tbody>
</table>

- Question #1: *Does GumTree do a good job?*
  
The possible answers are:
  1. GumTree does a good job: it helps to understand the change.
  2. GumTree does a bad job.

- Question #2: *Is GumTree better than diff?*
  
The possible answers are:
  1. GumTree is better.
  2. *diff* is better.
  3. GumTree is equivalent to *diff*. 
Evaluation (RQ2)

• Measure the performance of tree algorithms with respect to:
  • 1. the number of mappings; 2. the edit script size;
  • ChangeDistiller and RTED

• ChangeDistiller uses a simplified ASTs where the leaf nodes are code statements. They compute the metrics for both simplified ASTs
• CDG (ChangeDistiller granularity) and JDTG (Eclipse JDT granularity)

<table>
<thead>
<tr>
<th></th>
<th>CDG</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GT better</td>
<td>CD better</td>
<td>Equiv.</td>
</tr>
<tr>
<td>Mappings</td>
<td>4007 (31.32%)</td>
<td>542 (4.24%)</td>
<td>8243 (64.44%)</td>
</tr>
<tr>
<td>ES size</td>
<td>4938 (38.6%)</td>
<td>412 (3.22%)</td>
<td>7442 (58.18%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>JDTG</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GT better</td>
<td>RTED better</td>
<td>Equiv.</td>
</tr>
<tr>
<td>Mappings</td>
<td>8378 (65.49%)</td>
<td>203 (1.59%)</td>
<td>4211 (32.92%)</td>
</tr>
<tr>
<td>ES size</td>
<td>10358 (80.97%)</td>
<td>175 (1.37%)</td>
<td>2259 (17.66%)</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GT better</td>
<td>RTED better</td>
<td>Equiv.</td>
</tr>
<tr>
<td>Mappings</td>
<td>2806 (21.94%)</td>
<td>1234 (9.65%)</td>
<td>8752 (68.42%)</td>
</tr>
<tr>
<td>ES size</td>
<td>3020 (23.61%)</td>
<td>2193 (17.14%)</td>
<td>7579 (59.25%)</td>
</tr>
</tbody>
</table>

Table 2: Number of cases where GumTree is better (resp. worse and equivalent) than ChangeDistiller (top, middle) and RTED (bottom) for 2 metrics, number of mappings and edit script size (ES size), at the CDG granularity (top) and JDTG granularity (middle, bottom).
Evaluation (RQ3)

• Analysis of Move Actions
  • GumTree and ChangeDistiller

<table>
<thead>
<tr>
<th></th>
<th>GT only move op</th>
<th>GT other op</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD only move op</td>
<td>77</td>
<td>1</td>
</tr>
<tr>
<td>CD other op</td>
<td>52</td>
<td>12662</td>
</tr>
</tbody>
</table>

18 instances more in CD
49 instances more in GT

Table 3: Comparison of the number of move operations from GumTree and ChangeDistiller for 12,792 file pairs to be compared.

GT produces only move actions, and CD other actions

<CD produces only move actions, and GT other actions
Discussion Question

• Some possible weakness or remaining experiment in RQ1-2-3?
Discussion Question

• Analysis II: Discuss about ideas or thoughts the paper provoked?
  • the new problems you identify?
  • new research directions inspired?
Thank you!!