Flexible Near-miss Code Clone Detection in IDE

Minhaz F. Zibran and Chanchal K. Roy
Department of Computer Science, University of Saskatchewan, Canada
minhaz.zibran@usask.ca, croy@cs.usask.ca

1 Abstract
Although code clone detection recently becomes an active research area, there are only a few tools for clone detection in IDEs such as the Eclipse. We develop an Eclipse plug-in for flexible clone detection using a k-difference suffix-tree algorithm that also deals with near-miss clones.

2 Introduction and Background
- Code Clones: Duplicate or similar code fragments are called code clones.
- Categories:
  - Type-1: identical code fragments except for variations in whitespace and comments,
  - Type-2: where syntactically similar fragments are also considered clones,
  - Type-3: statements added/deleted/modified in copied fragments, and
  - Type-4: semantically similar code fragments.
- Preprocessing of the GST also takes running time for fingerprinting all code snippets using Rabin’s fingerprinting algorithm [3], which runs in linear time.
- Most of the clone detectors (except a few) out there can detect only Type-1 and Type-2 clones [4].
- Most clone detectors are designed for use as individual tools separate from IDE, and many tools are outdated.
- Most of the few tools that offer clone management in IDE, in fact, track clones initially detected by separate clone detector running locally, or remotely on server.
- We develop a clone detector, which (a) applies AST and Suffix Tree based hybrid algorithm for near-miss clone detection; (b) can detect Type-1, Type-2, and Type-3 clones; (c) works locally on Eclipse as a plugin.

3 Our Approach
The Figure shows the major steps in our clone detection approach.
- For each source file AST (Abstract Syntax Tree) is created.
- Using AST code snippets are extracted at the desired granularity (i.e., function, block, etc.).
- Comments are filtered out from source code.
- All code snippets are uniformly formatted.
- Variables and identifiers are consistently renamed.
- Each code snippet is fingerprinted at desired match unit (i.e., line, or token by token) applying Rabin’s fingerprinting algorithm [3], which runs in linear time.
- Generalized Suffix Tree (GST) for the sequence of fingerprints are created using Ukkonen’s linear algorithm [6].
- GST is preprocessed to perform LCA (Lowest Common Ancestor) and LCE (Longest Common Extension) queries in constant time.
- Exploiting the LCA and LCE query capabilities of GST, k-difference hybrid dynamic programming algorithm is applied to determine matches between pairs of snippets.

3.1 Computational Complexity
Suppose, we have \( F \) code fragments, and fragment \( f \) has \( l_f \) lines (or tokens) of code.
- Running time for fingerprinting all code snippets using Rabin’s fingerprinting algorithm is \( O(l_F) \), where \( l_F = \sum_{f=1}^{F} l_f \).
- The construction of the GST takes \( O(l_F) \) time. Type-1 and Type-2 clones are detected in \( O(l_F) \) time as we construct the GST.
- Preprocessing of the GST also takes \( O(l_F) \) time.
- Given a test \( T \) of length \( n \) and a pattern \( P \) of length \( n \), in \( O(kn) \) time and \( O(m + n) \) space, the k-difference hybrid algorithm can find all k-difference matches of \( P \) in \( T \).
- Based on the algorithm our implementation takes \( O(kl_F) \) time and \( O(l_F) \) space to find all occurrence of k-distance near-miss clones (Type-3) of a code fragment in all other fragments in hand.

4 Implementation
- The clone detection algorithm is implemented using Java (JDK 1.6).
- It also allows the user to find clones of a selected portion of code inside any function.
- Detected clones are presented as groups, and selection of an individual clone directs the user to the corresponding location in the source file.

5 Evaluation and Conclusion
To evaluate the core matching algorithms of our implementation, we conducted an empirical study on detecting near-miss clones in Weblab and PostgreSQL [1]. We compared the result of clone detection using our approach with that of NiCad [5]. We found that our algorithm found no false positive, and detected all clones that NiCad detected.

References