SDD: High Performance Code Clone Detection System for Large Scale Source Code

Seunghak Lee
Dept. of Chemistry
POSTECH
Pohang, Korea
boy3@postech.ac.kr

Iryoung Jeong
Dept. of Computer Science Education
Korea University
Seoul, Korea
crex@comedu.korea.ac.kr

ABSTRACT
Code clones in software increase maintenance cost and lower software quality. We have devised a new algorithm to detect duplicated parts of source code in large software. Our algorithm is adequate for large systems and detecting not only the exact but also similar parts of source code. Our simulation of this new algorithm, namely SDD (Similar Data Detection), indicates that it can detect duplicated parts of source code in huge software with high performance.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – Reuse models.

General Terms: Algorithm, Experimentation

Keywords
Duplicated code, code clone, system maintenance, search index

1. INTRODUCTION
Duplicated parts of source code exist for various reasons. Firstly, programmers tend to 'copy and paste' source code to reuse already existing ones because it makes the job easier. Another reason concerns the performance issues of the system. [2] However, it has been known that duplicated parts of source code in a system cause many problems. Most of all, it will increase the system's maintenance cost. For instance, in a case when a new function should be added to a system, all the places related to it have to be modified as well. On top of that it is difficult to assure the quality of systems. When programmers find bugs in systems, they should fix them in all of the duplicated source code. In the case of failing to locate duplicated source code, it becomes more difficult to fix the bugs completely. So, it is important to locate duplicated parts of source code especially in large software.

Code clones can be categorized into two different types: the exact copy of source code and bits and pieces of similar code. Our work covers both of them.

The complexity of existing method [1] to fully detect similar parts of source code grows drastically as the size of the source code gets bigger. So it makes difficult to analyze large systems. However SDD can decrease such a complexity by using an inverted index and an index. The result of SDD implementation indicates that our algorithm works well and we can modulate recall and precision by simply changing parameters. One million lines of source code (JDK com package) was analyzed just in 298 seconds, when SDD was executed on PC with Pentium 4 1.86GHz and 448MB RAM.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

2. SIMILAR CODE DETECTION ALGORITHM
To detect exact and similar parts of source code, we should define what they are. Here we define them as follows. A pair of similar source code has the same chunk within n-neighbor distance respectively as long as they match together. "N-neighbor distance" is a valid distance to link a chain. We can vary n-neighbor distance according to what level of similarity we want to use. Figure 1 shows how the chains are matched together when n-neighbor distance is two. When n-neighbor distance is one, a pair of code would be exactly matched.

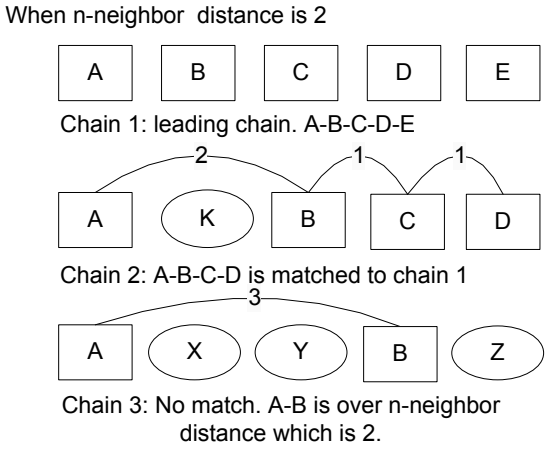

Figure 1: Definition of a similar code fragment

Figure 2 shows the process of extracting code clones in a large system. Our algorithm starts from making an inverted index and an index. An inverted index includes chunks and corresponding positions. On the other hand, an index has information about position and corresponding chunk. Then we extract n-times repeated index. If we want to extract more than three times repeated similar code, n would be three.

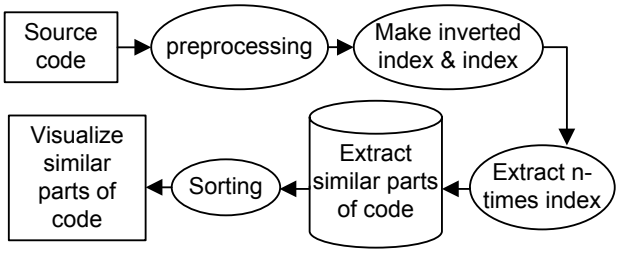

Figure 2: Extract similar code prcoess

By using an inverted index and an index, it is possible to extract similar parts of source code. With tracing adjacent indices sequentially (using an index), positions of adjacent similar code to adjacent indices can be efficiently found from an inverted index. If adjacent indices traced are within n-neighbor distance and any duplicated adjacent indices exist within n-neighbor distance in other places, a leading chain is made. Duplicated chains are also made at the same time. The leading chain and duplicated chains are growing as long as they can be linked. If it is impossible to grow a certain duplicated chain anymore, similarity of the leading chain and the duplicated chain are evaluated and stored, respectively. Evaluated similarity is the ratio of total chunks in the duplicated chain to matched chunks. When the leading chain fails to grow, all duplicated chains to the leading chain are evaluated and stored.

When the whole index is traced, all similar parts of source code are found. In addition, time complexity of our algorithm ('Extract similar code' part in Figure 2) is O(n). The length of duplicated chains is relatively small to the whole source code. Before showing the similar code, the results are sorted by the length of chain and similarity. When it is finished, we visualize the code clones.

3. SIMILAR CODE DETECTION RESULTS

We implemented our algorithm named SDD and analyzed projects with various sizes. In addition, we compared our work to PMD [3] (source code duplication detection tool). It has shown that our work detected similar code clones which PMD could not detect. SDD is able to discriminate similar code clones even slight changes are made to them by modifying, inserting or deleting a few lines or words. N-neighbor distance concept made it possible to detect even those slightest differences of code clones.

Figure 3 shows the result of our simulation. The level of detection of similar code clones can be modulated if we change the n-neighbor distance. In this simulation we set n-neighbor distance to three and chunk size to four words. Code clones less than ten chunks are neglected in results. Elapsed time to analyze java com package (37.65MB) was 67 seconds and post processing time (matching original position and file name) was 231 seconds when SDD was executed on PC with Pentium 4 1.86GHz and 448MB RAM. However, post processing time can be easily reduced if we use a proper data structure for the job. At this time file text searching method is used to do this job.

We believe that SDD is very powerful in detecting similar parts of source code in large systems. Moreover SDD is language independent.

Project Name	Size(MB)	K LOC[1]	# of clone	time(sec)
JDK-1.5 com[2]	37.65	1,098	1019	298
httpd-2.0.54	2.42	84	78	16
lucene-1.4.3	1.25	40	45	8
Phpwiki-1.2.9	0.23	7	5	1
ruby-1.8.2	6.53	245	200	28

Figure 3: Result of simulation[3]

4. FUTURE WORK

By reinforcing preprocessing such as changing identifiers to corresponding positions, parameterized clone detection [4] also can be a possible in our work. It would make it possible to detect the parameter variant parts of source code. When we compare each chunk of index, thesaurus could be helpful. It would extend meanings of words in software.

Our algorithm is easily adaptable to various fields where duplicate detection is useful. For example, detecting the literature plagiarism or similar DNA patterns can be achieved by using our algorithm. Also our algorithm can be useful to recognize spam mails or abusive web contents for their characteristic of duplicated contents.

5. CONCLUSION

We have presented a new algorithm to detect code clones namely SDD. Our simulation shows that SDD can detect exact and similar copies of source code in an acceptable time by using n neighbor distance concept. Using data structure of an inverted index and an index made it possible to detect code clones quickly.

6. REFERENCES

[1] Richard Wettel, "Automated Detection of Code Duplication Clusters" Diploma Thesis, June 2004

[2] T. Kamiya, S. Kusumoto, and K. Inoue, "*CCFinder: A multilinguistic token-based code clone detection system for large scale source code*," IEEE Transactions on Software Engineering, vol. 28, pp. 654--670, July 2002.

[3] PMD project, http://pmd.sourceforge.net/

[4] B. S. Baker. On finding duplication and near-duplication in large software systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), Toronto, pages 86--95, Los Alamitos, CA, July 1995. IEEE Computer Society Press.

[5] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, Clone detection using abstract syntax trees, Int. Conf. Softw. Maintenance, 1998.

[1] LOC : the number of lines, including null and comments

[2] JDK com package

[3] Homepage : http://comedu.korea.ac.kr:8080/sddwiki/FrontPage