Titre: Title:	Detection of Redundant Clone Relations Based on Clone Subsumption
Auteurs: Authors:	Ettore Merlo et Thierry M. Lavoie
Date:	2009
Type:	Rapport / Report
Référence: Citation:	Merlo, Ettore et Lavoie, Thierry M. (2009). Detection of Redundant Clone Relations Based on Clone Subsumption. Rapport technique. EPM-RT-2009-05.

URL de PolyPublie: PolyPublie URL:	http://publications.polymtl.ca/2646/
Version:	Version officielle de l'éditeur / Published version Non révisé par les pairs / Unrefereed
Conditions d'utilisation: Terms of Use:	Autre / Other

Maison d'édition: Publisher:	École Polytechnique de Montréal
URL officiel: Official URL:	http://publications.polymtl.ca/2646/
Mention légale: Legal notice:	Tous droits réservés / All rights reserved

EPM–RT–2009-05

DETECTION OF REDUNDANT CLONE RELATIONS BASED ON CLONE SUBSUMPTION

Ettore Merlo and Thierry M. Lavoie
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Avril 2009

EPM-RT-2009-5

Detection of Redundant Clone Relations Based on Clone Subsumption

Ettore Merlo and Thierry M. Lavoie
Department of Computer and software Engineering
École Polytechnique of Montréal

April 2009

Dépôt légal :
Bibliothèque nationale du Québec, 2009
Bibliothèque nationale du Canada, 2009

EPM-RT-2009-05
Detection of Redundant clone Relations Based on Clone Subsumption
par : Ettore Merlo et Thierry M. Lavoie
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Detection of Redundant Clone Relations Based on Clone Subsumption

Ettore Merlo, Thierry Lavoie

Department of Computer and Software Engineering, École Polytechnique de Montréal,
P.O. Box 6079, Downtown Station, Montreal, Quebec, H3C 3A7, Canada
e-mail: ettore.merlo@polymtl.ca

Abstract

Clone detection has been presented in the literature at different levels of fragment granularity from functions, to syntactic blocks, to variable length strings of source code or tokens. String matching approaches, pref x and suff x trees, metrics, syntactic approaches and others can be used to compare fragments for similarity.

Inclusion relations between source code lines may cause some clone relations to be redundant, when clones code fragments subsume each other. This may occur between nested blocks of source code, for example.

An original method to analyze this kind of redundancy in clone relations is presented. The proposed method is based on eff ciently combining clone subsumption information together with clone similarity relations on code fragments.

The amount of redundancy in clone relations has been evaluated on two open source Java systems, Tomcat and Eclipse. Experimental results are presented. Execution time performance of redundancy analysis is measured and reported. Results are discussed together with further proposed research.

Keywords: clone structural redundancy analysis, clone detection, software metrics, software analysis, open source code analysis.

1 Introduction

The granularity of fragments for clone analysis can be any subset of source code from a few lines to blocks to methods.

When matching of units smaller than methods is performed, some pairs in the computed similarity relation may somehow be structurally redundant because the compared blocks may be subsets of similar fragments.

The objective of this paper is to present and evaluate an original linear algorithm for structural redundancy detection from similarity relation between code fragments of arbitrary granularity.

Structural redundancy detection is computed by constructing equivalence classes of redundant clone blocks.

Blocks in a system may represent classes, methods, statement blocks, and so on. In this paper we consider only relations between blocks that are either method bodies or statement blocks.

Class bodies or other hierarchical blocks are not investigated in this paper, neither are other smaller scale fragments like expressions and their nested sub-expressions, and so on.

An inclusion relation may exist between the source code lines of some of these blocks, depending of the syntactic structure of the source code. For example, a block corresponding to a method body may contain several possibly nested statement blocks. For example, an if statement may be composed of a "then" block and an "else" block. Additionally a "then" block may contain other blocks and so on.

Block subsumption occur when the lines of code of a block are contained in the lines of code of another block. In other words, block subsumption in this paper refers to a set inclusion relation between the lines of clones of two blocks. Syntactic nesting in the AST introduces subsumption relations between nested blocks.

In general, the subsumption partial order generated by the set inclusion relation between ranges of source code lines may correspond to the full power set of subsets of lines of code and it can be quite large and complex.

If we restrict ourselves to syntactic blocks that can be either method bodies or statement blocks, the partial order becomes a forest of trees where each block may have at most one parent (the including block) or none if it's a method body. We call this forest the Block Inclusion Forest (BIF).

The set inclusion relation of source code ranges can be easily extracted from an AST representation of code or from the token sequence representation of it. Construction of the BIF is linear on the size of a system (LOC).

This set inclusion perspective of source code ranges may also benef t other clone detection methods, especially those that are not necessarily AST based, like token-based or string matching ones. It has to be noted that several clone detection approaches such as [9, 16, 21] already make use of some structural information derived from the AST, pre-f xes and suff xes.

When the AST structure is available, structural limits of blocks are easily determined. In token based approaches, often some sort of post or pre-processing is performed to achieve similar objectives.

Often, the results of similarity analysis are used for further purposes such as reporting to the user [11], visualization of clones [23], bug identif cation by token mapping mismatch analysis [25]. Structurally redundant similarity relation pairs increase the volume of input to further processing. In the case of human processing, additional volume may not lead to errors, but could be annoying to developers. When additional volume is fed to further processing, it increases the computational processing time. This is more noticeable when quadratic algorithms are used for further processing by token alignment [23] and token mapping [25].

An experimental evaluation of the execution time performance of the presented algorithm has been performed on two large open source systems written in Java, Tomcat and Eclipse, and compared to the execution of a quadratic naive algorithm. At the same time, f gures concerning the volume of structural similarity redundancy in these systems have been evaluated and reported.

The clone relation used in this paper is based on our previous work [7, 24, 27, 28, 29]. Other approaches for clone analysis have been presented in [6, 9, 14, 16, 19, 20, 21, 26]. Empirical studies and evaluation of clone detection approaches can be found in [3, 5, 10, 15, 22, 32, 33]. Scalability of clone detection approaches has been addressed in [8, 18], while clones and software evolution have been investigated in [4, 13]. Domain specif c clone detection approaches have been presented in [17] for the business f eld and in [12] for the automotive industry. Very interesting and comprehensive surveys about clone detection literature can be found in [30, 31].

Section 2 introduces an example to explain the presented approach and problem. Section 3 describes the proposed algorithm to detect clone relation redundancy in details. Section 4 describes the experiments, set-up, and results; Section 5 discusses results and issues, and Section 6 concludes this paper.

2 Example

Suppose that a system is composed of methods A, B, C, D, and E whose code structure is depicted in Fig. 1, 2, and 3. Suppose also that method A is similar to B and C is similar to D.

```
1  A(Z...) {            1  B(Z...) {
2      if (W...) {      2      if (W...) {
3          X...         3          X...
4      } else {         4      } else {
5          Y...         5          Y...
6      }                6      }
7  }                    7  }
```

Figure 1. Structure of methods A and B

```
1  C(Z...) {            1  D(Z...) {
2      if (W...) {      2      if (W...) {
3          X...         3          X...
4      }                4      }
5  }                    5  }
```

Figure 2. Structure of methods C and D

Blocks in this example are identif ed by the method name and the beginning and ending lines of enclosing curly brackets in the code. $A_{2,4}$ identif es the block enclosed in curly brackets from line 2 to 4 in method A.

Fig. 4 shows the Block Inclusion Forest (BIF) of methods A, B, C, D, and E. Nodes in Fig. 4 are depicted as polygons to emphasize the similarity relation between blocks. Methods are identif ed by the block representing the method body. Thus, method A is identif ed by block A_{1-7} and so on for the other methods. It should be noted that since A_{1-7} and B_{1-7} are similar, corresponding blocks A_{2-4} and B_{2-4} are also similar, as are blocks A_{5-7} and B_{5-7}. Along this perspective, clone relations $(A_{2-4},\ B_{2-4})$ and $(A_{5-7},\ B_{5-7})$ are structurally redundant with respect to the relation $(A_{1-7},\ B_{1-7})$.

Fig. 5 shows the clusters cl_0, cl_1, cl_2, cl_3 of similar blocks obtained when analyzing the whole system under unitary thresholds for identical block analysis. We can observe in this f gure that blocks A_{1-7} and B_{1-7} belong to cluster cl_0; C_{1-5} and D_{1-5} belong to cluster cl_1; A_{2-4}, B_{2-4}, C_{2-4}, D_{2-4}, and E_{1-3} belong to cluster cl_2; and A_{4-6} and B_{4-6} belong to cluster cl_3.

Blocks can be further sub-clustered based on the similarity relation of their parents int he BIF as def ned in Section 3. A_{2-4}, B_{2-4} belonging to cluster cl_2 also belong to sub-cluster $pSim[2,\ 0]$ since their respective parents $A_{1,7}$ and B_{1-7} are both in cluster cl_0. C_{2-4}, D_{2-4} belonging

```
1  E(Z...) {
2      X...
3  }
```

Figure 3. Structure of method E

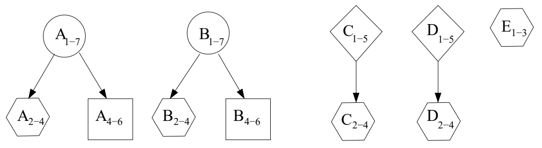

Figure 4. Block Inclusion Forest (BIF)

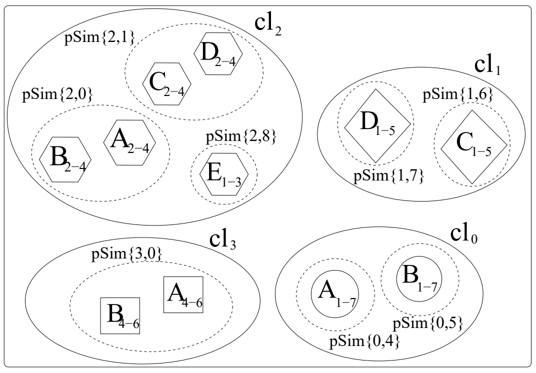

Figure 5. Similarity partitions

to cluster cl_2 also belong to sub-cluster $pSim[2,\ 1]$ since their respective parents C_{1-5} and D_{1-5} are both in cluster cl_1. Following the same reasoning, $A_{4-6},\ B_{4-6}$ belong to sub-cluster $pSim[3,\ 0]$.

A block that represents a method body doesn't have a parent in the BIF and belongs to a sub-cluster containing only the block itself as happens for blocks A_{1-7} from cluster cl_0 in sub-cluster $pSim[0,\ 4]$; B_{1-7} from cl_0 in sub-cluster $pSim[0,\ 5]$; C_{1-5} from cl_1 in sub-cluster $pSim[1,\ 6]$; D_{1-5} from cl_1 in sub-cluster $pSim[1,\ 7]$; and E_{1-3} from cl_2 in $pSim[2,\ 8]$.

Clone relations between blocks in the same sub-cluster are redundant, while clone relations between blocks in the same cluster but different sub-cluster are not redundant.

Clone relation $(A_{2-4},\ B_{2-4})$ is redundant and in fact its elements are in the same cluster cl_2 and also are in the same sub-cluster $pSim[2,\ 0]$. Clone relation $(A_{2-4},\ C_{2-4})$ is not redundant because its elements are in the same cluster cl_2 but are also in different sub-clusters $pSim[2,\ 0]$ and $pSim[2,\ 1]$.

Method bodies cannot produce redundant clone relations since they are the only member of their own sub-cluster. Thus, for example, clone relation $(A_{2-4},\ E_{1-3})$ is not redundant and indeed its elements are in the same cluster cl_2 but also belong to different sub-clusters $pSim[2,\ 0]$ and $pSim[2,\ 8]$

3 Algorithms

Suppose we have the Block Inclusion Forest (BIF) of a system and suppose also that the similarity relation between code blocks is an equivalence relation $sim(b_i, b_j)$ between code blocks b_i and b_j, which produces a partition of all blocks into mutually exclusive clusters cl_k. Equivalence properties of clone detection are often satisf ed by approaches based on AST, metrics, pref x and suff x similarity, and so on.

The idea behind structural redundancy analysis of pairs of blocks is that a similarity relation pair $(b_i,\ b_j)$ of blocks belonging to the same cluster cl_k is redundant if the parents of b_i and b_j in the BIF are similar under the same similarity relation as children, that is, if they, in turn, belong to the same cluster c_p. In other words, similar blocks are structurally redundant if they are also $pSim$ related as follows:

$$\begin{gathered} redundant(b_i,\ b_j) \leftrightarrow \\ sim(BIF.parent(b_i),\ BIF.parent(b_j),\ th) \leftrightarrow \\ (BIF.parent(b_i)\ \neq UNDEF) \wedge \\ (BIF.parent(b_j)\ \neq UNDEF)) \wedge \\ sim(b_i,\ b_j,\ th) \wedge \\ pSim(b_i,\ b_j,\ th) \end{gathered} \tag{1}$$

It should be noted that a block b_t, that doesn't have a parent in the BIF, corresponds to a method body and cannot be redundant, if similar to some other block, since it's a root of a tree in the BIF.

Relation $pSim$ is an equivalence relation on blocks that have parents in the BIF, since its ref exive, symmetric and transitive properties are easily verif ed as follows:

$$\begin{gathered} sim(BIF.parent(a),\ BIF.parent(a)) \\[1em] sim(BIF.parent(a),\ BIF.parent(b)) \rightarrow \\ sim(BIF.parent(b),\ BIF.parent(a)) \\[1em] sim(BIF.parent(a),\ BIF.parent(b)) \wedge \\ sim(BIF.parent(b),\ BIF.parent(c)) \rightarrow \\ sim(BIF.parent(a),\ BIF.parent(c)) \end{gathered} \tag{2}$$

A block b_t that doesn't have a parent in the BIF belongs to an equivalence class $[b_t] = \{b_t\}$ containing only the block itself.

Equivalence redundant classes under the $pSim$ relation can be computed using the algorithm presented in Fig. 6 for Structural Redundancy Analysis (SRA), where $clId$ function associates blocks with the cluster they belong to, in the cluster partition generated by the clone relation.

Relation $pSim$ is computed as a bi-dimensional associative table, rather than as a full bi-dimensional matrix, whose generic element $psim[k,\ m]$ represents the set of blocks

```
1  $pSim \leftarrow computeParentSimilarity($
       $clusters, clId,\ BIF)$

2      $seed\ =\ \mid clusters\mid$
3      $forall\ c_k \in\ clusters$
4          $forall\ b_i \in c_k$
5              $p_i\ = BIF.parent(b_i)$
6              $if\ (p_i \neq\ UNDEF)$
7                  $pSim[k,\ clId(p_i)].add(b_i)$
8              $else$
9                  $pSim[k,\ seed].add(b_i)$
10                 $seed\ = seed +\ 1$
11     $return\ pSim$
```

Figure 6. Structural redundancy analysis (SRA)

b_i that are all similar to one another and that have parents which are similar too. Refer to Fig. 5 for a graphical representation of similarity partitions induced by $pSim$ relation. Index k is associated with cluster c_k of similar blocks. The union of all $psim[k,\ m]$ that have the same k is equal to cluster c_k. Index m identif es the cluster of similar parents of some member b_i of $psim[k,\ m]$, so that either the parent of b_i exists and it belongs to cluster c_m or it doesn't exist because b_i is a method and m represents a progressive identif er that doesn't correspond to any similarity cluster. In this case, the progressive identif er is generated through a $seed$ as shown in lines $2,\ 9,\ 10$ in Fig. 6. Line 3 traverses all clusters. Line 4 iterates over the blocks b_i in cluster c_k. Line 5 computes p_i parent of b_i in the BIF. Line 6 tests if p_i exists and then adds b_i to the proper element of $pSim$ at index $[k,\ clId(p_i)]$, where $clId$ returns the identif er of the cluster to which p_i belongs. At least, p_i belongs to a cluster containing p_i itself. If line 6 fails, that is if b_i is a method or a function and therefore it doesn't have a parent, the index of $pSim$ is $[k,\ seed]$. The initial value of $seed$ is out of the range of cluster identif ers (see line 3) and it is incremented for each additional use (line 10).

Lines 5 to 10 can be executed in constant time if elements of $pSim$ are represented as lists and if the operation add can be implemented so that it has a constant time execution complexity. This condition is easily satisf ed in conventional list implementations. Lines 5 to 10 are executed as many times as the number of all blocks belonging to all clusters, but since clusters c_k represent a partition of blocks, this number is equal to the total number of blocks in the system under investigation. We can conclude that algorithm SRA is linear on the number of blocks composing a system.

The following properties hold for elements in the equivalence classes of structural redundancy $pSim$ computed by SRA:

$$\begin{gathered}\forall b_i\ b_j \in pSim[k,m] \rightarrow redundant(b_i,\ b_j)\\[1ex] b_i \in pSim[k,m]\ \wedge\ b_j \in pSim[k,n] \wedge\\ (m \neq n) \rightarrow \neg\ (redundant(b_i,\ b_j))\end{gathered} \tag{3}$$

Additionally, we may def ne as *def nitely redundant* those blocks belonging only to redundant pairs. This occurs when a $pSim$ class is equal to the whole cluster to be sub-partitioned. Similarly, we may def ne as *def nitely non-redundant* those blocks that participate only in non-redundant pairs. This occurs when blocks are alone in a class and there are at least two $pSim$ classes from the sub-partitioning of a cluster. *Possibly redundant* blocks participate in both redundant and non-redundant pairs, which occurs when the sub-partitioning of a cluster produces more than one $pSim$ class with at least two elements.

4 Experiments and results

Experiments have been performed on two open-source Java systems, Tomcat and Eclipse, and have been executed on an Intel Core 2 Duo, 3.0 GHz clock, 3 GB RAM, under Linux Fedora 8. Code has been compiled with g++ 4.1.2. Tomcat [2] is an implementation of the Java Servlet and the Java Server Pages technologies and is widely used to power different kinds of web-based systems. Eclipse [1] is a complete IDE to develop Java applications. The size, number of methods, number of blocks and average nesting level of blocks in the BIF for Tomcat and Eclipse are reported in Table 1. In this table and in all reported f gures and experiments, data refer to blocks larger than 6 LOC, which is a threshold that has been chosen as a lower bound of significance for block sizes. Choice of 6 LOCs comes from the literature on clones and from previous authors' experience and is considered a reasonable size threshold for clone detection. Other size thresholds can be used if required.

The distribution of nesting levels of blocks in the BIF for the investigated systems is shown in Fig. 7 and Fig. 8. Methods are considered as being at nesting level 0. Nesting levels of statement blocks in the BIF start at 1 and increase as nesting does.

System	Tomcat	Eclipse
Version	5.5	3.3
LOC	130K	1.3M
Methods	5047	60326
Blocks	5538	32113
BIF Average Nesting	0.89	0.62

Table 1. System features

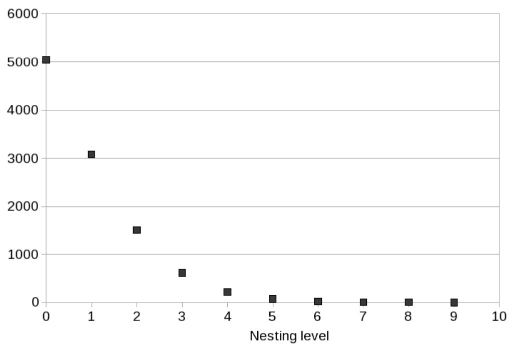

Figure 7. Tomcat BIF nesting level distribution

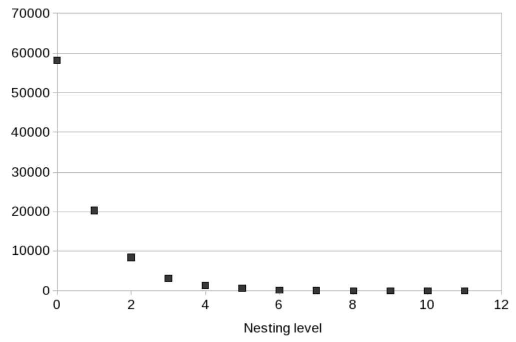

Figure 8. Eclipse BIF nesting level distribution

The number of blocks at higher BIF nesting levels decreases as block nesting increases. It is interesting to compute the distribution of nesting levels because method clones produce as many redundant relations as the number of their nested blocks. A rather f at structured system has less structurally redundant clones than a deeply nested system for the same cloning ratio at method level.

Syntactic analysis of investigated systems has been performed using Eclipse to extract the Block Inclusion Forest (BIF) and block metrics from the Abstract Syntax Tree (AST). Block similarity has been computed using $CLAN$ [10]. The algorithm for similarity clustering used in $CLAN$ is shown in Fig.9. The following metrics have been computed for each block in the investigated systems: number of statements, number of branches (IF, CASE, etc.), number of loops (FOR, WHILE, DO, etc.), number of calls, number of parameters (zero for nested blocks, possibly non-zero for methods).

```
1  $clustersType \leftarrow computeClusters($
       $blocks,\ metrics)$

2      $clusters.clear()$
3      $forall\ b \in blocks$
4          $i = 1$
5          $key = ()$
6          $while\ (i \leq\ metrics[b].size())$
8              $key.append(int(metrics[b][i]))$
9              $i = i + 1$
10         $kSet[b] = key$
11         $clusters[key] = clusters[key] \cup \{b\}$
12     $return\ clusters$
```

Figure 9. Similarity Clustering

Overall method and block information have been reported in Table 1 together with average nesting levels in the BIF. Table 2 presents the cardinality of the clusters computed by algorithm SRA. The minimum cluster cardinality is 2 for all steps and has not been reported.

The distribution of cluster cardinality is reported in Fig. 10 for Tomcat and in Fig. 11 for Eclipse.

Table. 4 presents the redundant clone block information f gures. Redundant clusters can be safely ignored, because they are totally composed of *def nitely redundant* blocks. It is interesting to remark that the percentage of *def nitely redundant* blocks is quite small for both systems, but is higher for Tomcat than for Eclipse. *Possibly redundant* blocks are in a larger number for Eclipse than for Tomcat and the number of *def nitely non-redundant* is about the same for both systems. The percentage of redundant number of clone pairs is between 10 and 13% in the investigated systems and the total number of relevant clone pairs is between 86 and 89% for those systems.

Tomcat		Eclipse	
avg	max	avg	max
3.71	79	5.36	342

Table 2. Cluster cardinalities

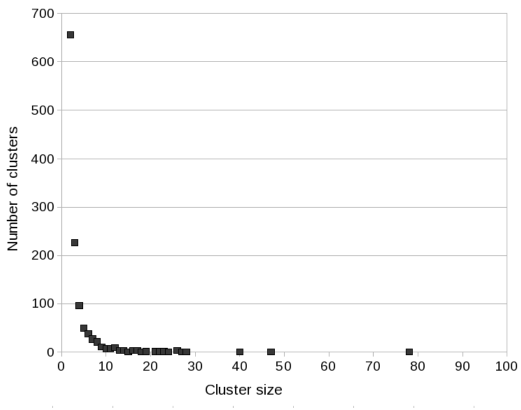

Figure 10. Tomcat cluster cardinality distribution

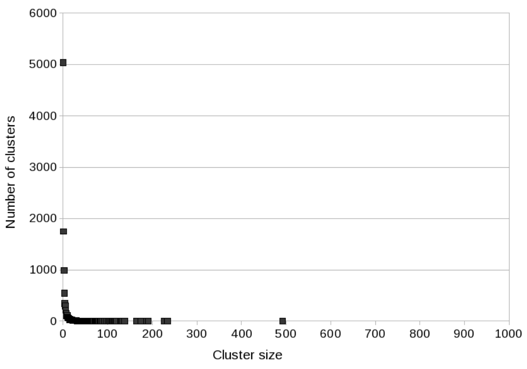

Figure 11. Eclipse cluster cardinality distribution

Sub-cluster cardinalities represent the size of structurally redundant sets of blocks. Cardinality distribution of sub-clusters corresponding to $pSim$ partition is depicted in Fig. 12 for Tomcat and in Fig. 13 for Eclipse.

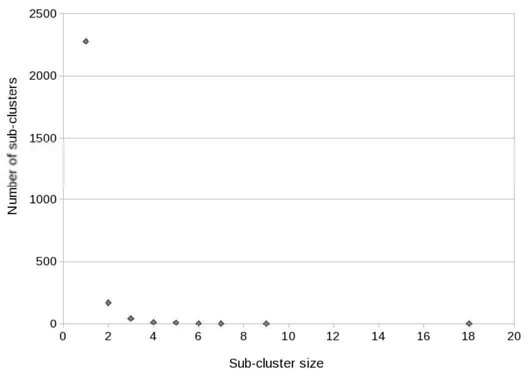

Figure 12. Tomcat sub-cluster cardinality distribution

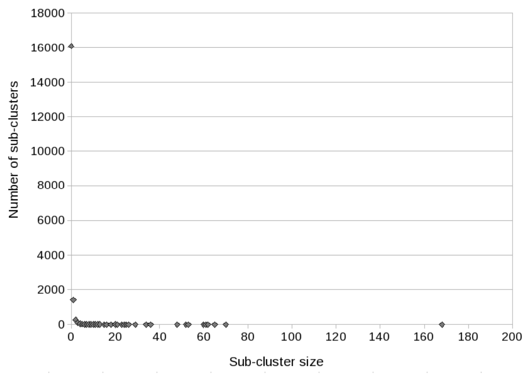

Figure 13. Eclipse sub-cluster cardinality distribution

It is interesting to remark in these f gures that the overall distribution shape reminds that of a Pareto distribution.

Table 3 show CPU execution time in seconds for SRA. Tomcat execution times are smaller because its total number of blocks is smaller than that of Eclipse.

5 Discussion

Results show that structural redundancy in clone relation can be quickly evaluated using the SRA algorithm on a common desktop computer.

From an application point of view, the results seem

System	Time (s)
Tomcat	0.201
Eclipse	1.91

Table 3. Eclipse and Tomcat execution times

	Tomcat	Eclipse
Number of clusters	1192	10363
Relevant clusters	1093	9775
Redundant clusters	99	588
Number of redundant blocks	355 (6.33%)	2310 (3.5%)
Number of possibly redundant blocks	809 (14.43%)	15568 (23.61%)
Number of def nitely non-redundant blocks	4445 (79.24%)	48044 (72.89%)
Number of pairs	21456	791681
Relevant pairs	19121 (89.1%)	687280 (86.8%)
Redundant Pairs	2335 (10.9%)	104401 (13.2%)

Table 4. Clusters redundancy analysis

interesting for software engineering purposes such as refactoring, evolution and bug detection analysis.

The presented results also depend on several specif c factors used in the experiments and discussed in the following.

The investigated systems are written in Java. Furthermore, only two specif c systems with specif c structural block inclusion and cloning features have been investigated. Results using other object oriented languages or procedural languages may be different and should be investigated. Investigation should also be carried out on more Java systems for different nesting and cloning characteristics.

The Java inner class programming construct presents an anomaly of classif cation between blocks and functions. Methods bodies from inner classes are actually methods that are nested in other method bodies from other classes. For the presented experiments and results, methods from inner classes have been labeled as compound statement blocks rather than methods at root level in the BIF. This little lack of precision could be easily corrected if required.

Clone relations have been based on software metrics and on the specif c metrics mentioned in Section 4. Other approaches, based for example on string matching, pref x or suff x trees, or graph matching, may produce slightly different clusters of similar fragments. In addition, only blocks bigger than 6 LOC have been considered for experiments. Eliminating small blocks from clone analysis is a common strategy to avoid reporting trivial relations between small blocks. The specif c choice of 6 LOCs is consistent with the literature on clones and with the authors' previous experience and is considered a reasonable size threshold for clone detection. Other size thresholds can be used if desired or the threshold f ltering could be questioned and removed if necessary.

Metrics based block analysis is eff cient because metrics for clone analysis are compositional and a single pass through the AST is suff cient to annotate it with the metrics value for each sub-tree. It should be noted that the choice of metrics has been based on the authors' previous experience, but the metrics presented in [9, 19, 23] could be used almost interchangeably.

Metrics based approaches suffer from the typical distortion that similarity of metrics doesn't necessarily mean similarity of code. Consider, for example, two blocks, the f rst one composed of an if statement containing a for statement and the second composed of a for statement containing an if statement. Although the metrics count may be the same, the two blocks should not be considered similar, at least not for a unitary threshold indicating a perfect match. Conversely, similarity of code necessarily means similarity of metrics. Think of two code blocks: the f rst being a for statement including an if statement and the second being an if statement including a for. It's easy to imagine a situation where the metrics are the same for the two blocks, while the two fragments are not similar. In metrics based approaches, false negatives may be absent, while false positives may exist.

We are interested here in discussing whether the presented redundancy detection approach changes or not the intrinsic matching distorsion mentioned above. This distortion may have an impact in structural redundancy clone analysis because false positives may exist in nested blocks as well as in functions at top nesting levels. Since false negatives are absent, when a clone relation is not detected between children or parents in the BIF, no similarity or error is possible. Errors may only occur when some similarity relation exists among children or parents.

When similar children don't have similar parents in the BIF, children are correctly reported as similar. False positive children are still reported as false positive. No new errors are introduced by the structural redundancy analysis. When similar parents don't have similar children, parents may be reported as similar, recursively depending on their parents' similarity, and so on. False positive parents may still be reported as false positive. There is no new error generated and no change in the global error rate.

Falsely positive children that have falsely similar parents are not reported after structural redundancy analysis, because they are erroneously considered redundant. However, this reduces the number of false positives to parents only and it may indeed reduce the global error rate. Falsely positive children that have truly similar parents are not reported since they are considered redundant, which is def nitely not an error since parents are indeed similar. Truly positive children that have falsely similar parents are not reported as clones since they are erroneously considered redundant. This is the only case in which the redundancy analysis introduces a new error. Truly positive children that have truly similar parents are not reported as clones, which is perfectly correct.

Overall, structural redundancy analysis reduces the error rate of metrics based clone detection when falsely positive children have falsely similar parents, and it increases the error rate when truly positive children have falsely similar parents.

The global error after structural redundancy analysis is the balance between the two types of errors. Unfortunately, this balance has not been evaluated in this paper. Nevertheless, we may argue that if the false positives are uniformly distributed over the fragments, the two errors may somehow compensate each other and therefore the overall impact may not be very signif cant. The precise values of uncompensated errors should be investigated in future experiments.

Experimental evaluation of *def nitely redundant*, *def-nitely non-redundant* and *possibly redundant* fragments has been measured and shows that between 3 and 6% of blocks are *def nitely redundant*, between 14 and 23% of blocks are *possibly redundant*, while between 10 and 13% of clone pairs are redundant. These f gures are encouraging showing that some computational effort can be reduced by taking into account redundancy information.

Some source code inclusions relations can be originated by sub-sequences of statements in a compound statement. The presented approach doesn't yet address inclusion relations induced by sub-sequences and this issue is left to further research. Sub-sequences are nevertheless included in a compound statement, so, if this information is represented in the BIF, the algorithm would compute the redundancy analysis also for sub-sequences.

Some further processing of clone blocks information may be done directly from the clusters and redundancy sub-partitions. For example, visualization of clone blocks can be done by selecting a def nitely non-redundant clone block representative of a cluster and by visualizing the differences with respect to the selected block of all other members of the same cluster. This is the visualization approach used in CLAN.

If bug detection caused by inconsistent image identif er replacement (as described in [25] is sought, identif er mapping can be performed with respect to a def nitely non-redundant representative of a cluster and by analyzing the consistency of mapping in all other members of the same cluster.

Nevertheless, some further processing of clone information may require the output of all pairs of clone blocks. The number of pairs of clone blocks that can be constructed from clusters is quadratic on the cardinality of clusters. Although clusters and redundancy information can be computed in linear time, computation of all pairs of clone blocks shows nevertheless a quadratic component in complexity. Redundancy identif cation may reduce by factors the quadratic component, but still computation of all clone block pairs remains quadratic in general.

6. Conclusions

Clone relations may be affected by structural redundancy when block fragments at different levels of nesting in the Block Inclusion Forest (BIF) are considered for similarity.

A linear algorithm for structural redundancy detection of similarity information has been proposed and experimentally evaluated on Tomcat and Eclipse in this paper.

The experimental evaluation of *def nitely redundant*, *def-initely non-redundant* and *possibly redundant* analyses has been performed.

The detected redundancy of clone pairs appears to be about 10% for Tomcat and 13% for Eclipse.

The proposed SRA algorithm computes redundant sub-partitions of clusters of similar blocks in linear memory and execution time complexity on the total number of analyzed blocks and is also fast in practice: execution CPU times have been about 0.2 s for Tomcat and about 1.9 s for Eclipse.

Further research is required on more numerous and different system, possibly written in languages other than Java.

7. Acknowledgements

This research has been funded by the Natural Sciences and Engineering Research Council of Canada under the Discovery Grants Program. The authors wish to thank Melissa Mongeau for her contribution to Eclipse based metrics extraction.

References

[1] Eclipse. http://www.eclipse.org.

[2] Tomcat. http://tomcat.apache.org.

[3] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by accident: An empirical study of source code cloning across software systems. In *International Symposium on Empirical Software Engineering*, 2005.

[4] G. Antoniol, U. Villano, E. Merlo, and M. D. Penta. Analyzing clone evolution in the linux kernel. *Information and Software Technology*, pages 755–765, 2002.

[5] L. Aversano, L. Cerulo, and M. D. Penta. How clones are maintained: An empirical study. In *European Conference on Software Maintenance and Reengineering*, 2007.

[6] B. Baker. Finding clones with dup: Analysis of an experiment. *IEEE Transactions on Software Engineering*, 2007.

[7] M. Balazinska, E. Merlo, M. Dagenais, B. Lagu, and K. Kontogiannis. Advanced clone-analysis as a basis for object-oriented system refactoring. In *Proc. Working Conference on Reverse Engineering (WCRE)*, pages 98–107. IEEE Computer Society Press, 2000.

[8] H. Basit, S. Pugliesi, W. Smyth, A. Turpin, and S. Jarzabek. Eff cient token based clone detection with f exible tokenization. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2007.

[9] I. Baxter, A. Yahin, l. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, pages 368–377, 1998.

[10] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, 2007.

[11] S. Bouktif, G. Antoniol, M. Neteler, and E. Merlo. A novel approach to optimize clone refactoring activity. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1037–1043. ACM Press, 2006.

[12] F. Deissenboeck, B. Hummel, E. Juergens, B. Schaetz, S. Wagner, S. Teuchert, and J. F. Girard. Clone detection in automotive model-based development. In *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society Press, 2008.

[13] E. Duala-Ekoko and M. Robillard. Tracking code clones in evolving software. In *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society Press, 2007.

[14] S. Ducasse, O. Nierstrasz, and M. Rieger. On the effectiveness of clone detection by string matching. *International Journal on Software Maintenance and Evolution: Research and Practice*, 2006.

[15] R. Falke, P.Frenzel, and R. Koschke. Empirical evaluation of clone detection using syntax suff x trees. *Empirical Software Engineering Journal*, 13(6):601–643, 2008.

[16] N. Gode and R. Koschke. Incremental clone detection. In *European Conference on Software Maintenance and Reengineering*, 2009.

[17] J. Guo and Y. Zou. Detecting clones in business applications. In *Proceedings of the Working Conference on Reverse Engineering*, 2008.

[18] Z. Jiang and A. Hassan. A framework for studying clones in large software systems. In *Workshop on Source Code Analysis and Manipulation*, 2007.

[19] J. H. Johnson. Identifying redundancy in source code using f ngerprints. In *CASCON*, pages 171–183, October 1993.

[20] T. Kamiya. Variation analysis of context-sharing identif ers with code clone. In *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*. IEEE Computer Society Press, 2008.

[21] T. Kamiya, S. Kusumoto, and K. Inoue. Ccf nder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.

[22] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2005.

[23] K. Kontogiannis, R. De Mori, R. Bernstein, M. Galler, and E. Merlo. Pattern matching for clone and concept detection. *Journal of Automated Software Engineering*, 3:77–108, March 1996.

[24] B. Lagüe, D. Proulx, E. Merlo, J. Mayrand, and J. Hudepohl. Assessing the benef ts of incorporating function clone detection in a development process. In *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, pages 314–321, 1997.

[25] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, pages 1–17, 2006.

[26] A. Marcus and J. I. Maletic. Identif cation of high-level concept clones in source code. In *ASE '01: Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, page 107, Washington, DC, USA, 2001. IEEE Computer Society.

[27] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, pages 244–253, Monterey, CA, Nov 1996.

[28] E. Merlo. Detection of plagiarism in university projects using metrics-based spectral similarity. In R. Koschke, E. Merlo, and A. Walenstein, editors, *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. IBFI.

[29] E. Merlo, G. Antoniol, M. D. Penta, and F. Rollo. Linear complexity object-oriented similarity for clone detection and software evolution analysis. In *Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press*, pages 412–416. IEEE Computer Society Press, 2004.

[30] U. of Alabama at Birmingham. Clone literature. "http://students.cis.uab.edu/tairasr/clones/literature".

[31] C. Roy and J. Cordy. A survey on software clone detection research. Technical Report Technical Report 2007-541, School of Computing, Queen's University, November 2007.

[32] C. Roy and J. Cordy. Scenario-based comparison of clone detection techniques. In *International Conference on Program Comprehension*, pages 153–162. IEEE Computer Society Press, 2008.

[33] C. K. Roy and J. R. Cordy. An empirical study of function clones in open source software. In *Proceedings of the Working Conference on Reverse Engineering*, 2008.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873

École Polytechnique de Montréal

École affiliée à l'Université de Montréal

Campus de l'Université de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec)
Canada H3C 3A7

www.polymtl.ca

ÉCOLE
POLYTECHNIQUE
MONTRÉAL