On Detection of Gapped Code Clones using Gap Locations

Yasushi Ueda[†], Toshihiro Kamiya[‡], Shinji Kusumoto[†] and Katsuro Inoue[†]

[†]Graduate School of Information Science
and Technology, Osaka University,
Toyonaka, Osaka 560-8531, Japan
Phone:+81-6-6850-6571, Fax:+81-6-6850-6574
{y-ueda, kusumoto, inoue}@ist.osaka-u.ac.jp

[‡]PRESTO, Japan Science and Technology Corp.
Current Address:
Graduate School of Information Science
and Technology, Osaka University,
Toyonaka, Osaka 560-8531, Japan
Phone:+81-6-6850-6571, Fax:+81-6-6850-6574
kamiya@ist.osaka-u.ac.jp

Abstract

It is generally said that code clone is one of the factors to make software maintenance difficult. A code clone is a code portion in source files that is identical or similar to another. Clones are introduced because of various reasons such as reusing code by 'copy-and-paste' and so on. Since developers usually modify the copied-and-pasted code portions, there are some gaps between the original code portion and it. Here, we call such code portions include some gaps Gapped code clone. Up to the present, several code clone detection methods, which give consideration to such gap, have been proposed. However, it needs a lot of cost to detect all the gapped code clones. This paper proposes a new method to visualize the gapped code clones just as if they were actually detected, based on the detection results of conventional code clone. Using the proposed method, the developer can specify the target clones efficiently. Moreover, we implement the proposed method into the maintenance support environment and conduct the experimental evaluation.

Keywords: Software Maintenance, Gapped Code Clone, Code Visualization

1 Introduction

Recently, as software system becomes larger and more complex, the cost for debugging and maintaining a program has been increasing. It is generally said that code clone is one of the factors to make software maintenance difficult [8].

A code clone is a code portion in source files that is identical or similar to another. Clones are introduced because of various reasons such as reusing code by 'copy-and-paste', mental macro, or intentionally repeating a code portion for performance enhancement, etc [6].

If a code portion includes some faults, we have to correct all the code clones corresponded to the code portion. Also, in order to improve the maintainability of the program, it is often required that a set of code clones is integrated to one module. In these cases, it is necessary to develop a systematic method to detect code clones automatically. Up to the present, several code clone detection methods have been proposed [1][2][3][6][7][11][13][14][17][15].

We have also proposed a clone detection technique, which consists of transformation of input source text and token-by-token comparison. Then, based on the proposed code clone detection technique, we developed a tool named CCFinder [12], which extracts code clones in C/C++, Java, or COBOL source files. Since CCFinder chops input code into a token sequence and transforms it based on transformation rules (ex. normalizing the user-defined identifiers), it can detect clone code portions that have different in syntax but might have similar meanings.

We have applied CCFinder to several commercial software and encountered a practical problem. In the case of 'copy-and-paste' reuse, the developers usually do not reuse the code portion as it was but partially modify and then reuse it. Moreover, in the modification, they do not only replace the user-defined identifiers in the code portion but also modify them. For example, some additional statements would be inserted into it. Then, some gaps are exist between the original code portion and the copied-and-pasted code portion. Here, we call such code clone as "gapped code clone".

In such case, the developers can subjectively identify the code clones even if they include some gaps among them. On the other hand, CCFinder detects the clone as several short code clones separately. Or, since the minimum length of a code clone must be set in CCFinder beforehand, if the code portion is too short, CCFinder does not identify it as a code clone. Conversely, if we set a small value to the minimum length, then a lot of code clones are detected and the information is practically useless.

In this paper, we propose a new gapped clone detection method using CCFinder. The proposed method does not detect all the gapped code clones but just show all the candidates of gapped code clones based on the conventional output from CCFinder. Then, choosing the interesting gapped code clone among them, the user can conduct the clone analysis efficiently. Also, we have implemented the proposed method into our maintenance support environment Gemini [19]. Then, we have applied it to the data collected from actual program development in Osaka University and confirmed the usefulness of the proposed method.

Section 2 introduces the several types of code clone and describes the needs of gapped code clone detection. Section 3 proposes a new gapped code clone detection method and implementation. Then, in Section 4, we apply the proposed method to the program development and evaluate the usefulness of it. Finally, Section 5 concludes the paper and presents suggestions for future work.

2 Preliminaries

2.1 Classification of code clones

Here, we classify code clone corresponded to a code portion C into three types:

Exact code clone E is a code portion that is the same as C except for the difference about blank, new line and comments.

Renamed code clone R is a code portion that is the same as C except for the difference about the corresponded names of user-defined identifier (name of variables, constant, class, method and so on). Also, the reserved words and the sentence structures are the same between R and C.

Gapped code clone G: G is a code portion that is partly similar to C. That is, G includes some different code portion (we call this portion as **gap**) from C. Assume that a code portion C is copied and pasted. Then, each of the following three kinds of codes is called gap: (1) Newly added code: They are inserted to the pasted code portion C, (2) Deleted code: They are deleted from the pasted code portion C, (3) Modified code: They are modified in the pasted code portion C. Later, we call G simply "gapped-clone".

Figure 1 shows the example of Exact code clone E, Renamed code clone R and Gapped code clone G.

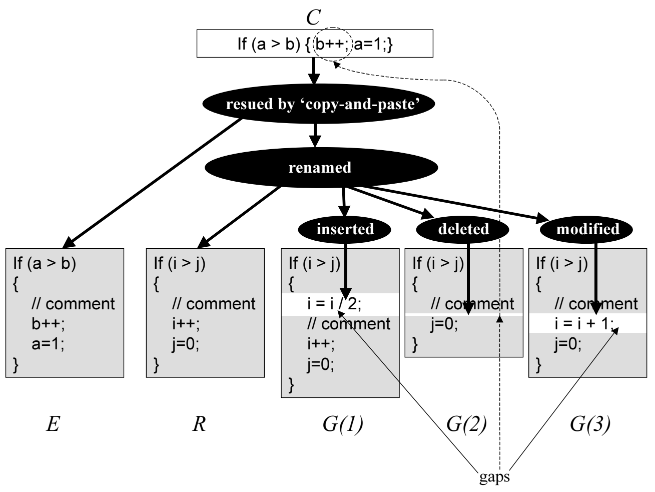

C
If (a > b) {b++; a=1;}
resued by 'copy-and-paste'
renamed
inserted
deleted
modified
If (a > b) { // comment b++; a=1; }
If (i > j) { // comment i++; j=0; }
If (i > j) { i = i / 2; // comment i++; j=0; }
If (i > j) { // comment j=0; }
If (i > j) { // comment i = i + 1; j=0; }
E
R
G(1)
G(2)
G(3)
gaps

Figure 1. 'Copy-and-paste' reuse

2.2 What type of code clones does CCFinder detect?

CCFinder detects code clones from programs and outputs the locations of the clone pairs on the programs. The process consists of four steps:

Step1: Lexical analysis: Each line of source files is divided into tokens corresponding to a lexical rule of the programming language. The tokens of all source files are concatenated into a single token sequence, so that finding clones in multiple files is performed in the same way as single file analysis.

Step2: Transformation: The token sequence is transformed, i.e., tokens are added, removed, or changed based on the transformation rules that aim at regularization of identifiers and identification of structures. Then, each identifier related to types, variables, and constants is replaced with a special token. This replacement makes code portions with different variable names clone pairs.

Step3: Match Detection: From all the sub-strings on the transformed token sequence, equivalent pairs are detected as clone pairs. Here, the user can specify the minimum length of code clone. So, code clones that are shorter than the minimum length are omitted.

Step4: Formatting: Each location of clone pair is converted into line numbers on the original source files.

As the result, CCfinder can detect exact code clone and renamed code clone. Later, we call both exact code clone and renamed code clone as **NG-clone** (Non-Gapped code clone).

```
    1. static void foo() throws RESyntaxException
    2. {
    3.    String a[] = new String [] {"123,400", "abc"};
A1  4.    org.apache.regexp.RE pat =
A1  5.              new org.apache.regexp.RE("[0-9,]+");
A1  6.    int sum = 0;
    7.    for (int i = 0; i < a.length; ++i)
B1  8.    {
B1  9.       if (pat.match(a[i])){
B1 10.          sum += Sample.parseNumber(pat.getParen(0));}
   11.    }
C1 12.    System.out.println("sum = " + sum);
   13. }
   14. static void goo(String [] a) throws RESyntaxException
   15. {
A2 16.    RE  exp = new RE("[0-9,]+");
A2 17.    int sum = 0;
   18.    int i   = 0;
   19.    while (i < a.length)
B2 20.    {
B2 21.       if (exp.match(a[i]))
B2 22.          sum += parseNumber(exp.getParen(0));
   23.       i++;
   24.    }
C2 25.    System.out.println("sum = " + sum);
   26. }
      :
      :
```

Figure 2. Example code clones detected by CCFinder

In Figure 2, there are two methods written in Java and the labels on the left side of the line number denote the NG-clones. Here, the minimum length of the code clone is set at five (tokens). In Figure 2, `A1` (lines 4-6) and `A2` (lines 16 and 17), `B1` (lines 8-10) and `B2` (lines 20-22), and `C1` (line 12) and `C2` (line 25) are NG-clones, respectively. Each of the clone length is 7, 18, 6 tokens.

As you can see, there are several differences between the clones A1 and A2 or B1 and B2 as follows.

Difference1: Name spaces
(e.g. "`org.apache.regexp.RE`" and "`RE`").

Difference2: Variable names (e.g. "`pat`" and "`exp`").

Difference3: Indentations or line feeds.

Difference4: Brace notations.

By the Transformation in CCFinder's code clone detection process, the differences are omitted. It aims to identify practically meaningless differences as the same code. However, the arguments or `for` and `while` statements in the two methods are still identified as the different code portions although they may be trivial differences.

2.3 Needs of gapped-clone detection

As described in Section 2.2, the two methods in Figure 2 are quite similar each other.

Generally, in the case that the minimum length of the code clone is set as five tokens, too many code clones would be detected although only three clone pairs are detected in Section 2.2.

On the other hand, if the minimum length is set as 15 for the code of Figure 2, only `B1` and `B2` is detected as code clone and other clones are not detected. It is quite difficult to identify that the two methods are similar based on only the information of `B`. Moreover, if the minimum length is set as 20, CCFinder cannot detect code clones from the two methods.

However, if we permit a code clone to include gaps that are shorter than 10 tokens, the code portions labeled with `A`, `B` and `C` are jointed and they are identified as a single code clone. In that case, even supposing that the minimum length is set as a little large number such as 15 or 20, it can be detected because the length becomes longer. Clearly, it is easier to check the similarity of the two methods based on the large one than based on only the information of `B`. Therefore, the detection of gapped-clones is meaningful in investigating the similarity of program.

3 Focusing on gapped-clones using gap locations

When a list of NG-clones (Non-Gapped clones) exist, you can solve the gapped-clones detection as a kind of combination problem, to decide what combinations of NG-clones can be considered as gapped-clones. If there are many overlapping or overcrowded NG-clones, one NG-clone may have many other NG-clones to be combined into a gapped-clone. As a whole, identification of gapped-clones from such entangled NG-clones makes a combination explosion, which needs long time for computation. Practically, in order to investigate similarity of methods or code blocks as we have done in Section 2.2 (shown as Figure 2), you can use each entanglement to find similar code portions, instead of wasting time on untangling it into gapped-clones.

Our approach is to detect entanglements of NG-clones as disjoint subsets of a set of all NG-clones, before computation of NG-clones. These entanglements, which can be computed in relatively shorter time, show the locations where gapped-clones possibly exist. A tool enables the users to see such entanglements in a scatter plot and to pick up interactively one of them to find gapped-clones in it.

3.1 Focusing process

Figure 3 shows the entire process of our approach to support interactive focusing on gapped-clones. The process consists of four steps:

Step1: NG-Clone detection.

Step2: Gap identification.

Step3: Visuallization.

Step4: Analysis of source code.

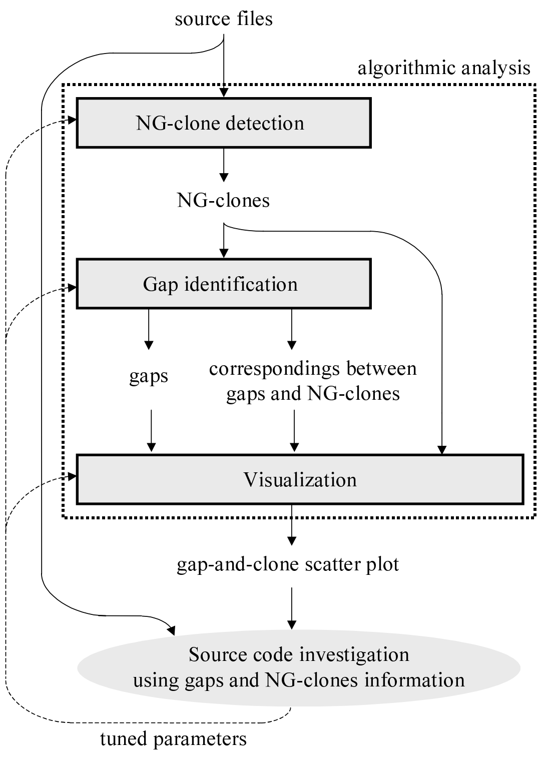

source files
algorithmic analysis
NG-clone detection
NG-clones
Gap identification
gaps
correspondings between gaps and NG-clones
Visualization
gap-and-clone scatter plot
Source code investigation using gaps and NG-clones information
tuned parameters

Figure 3. Gapped-clone focusing process

In Step1, we detect NG-clones from input source code. Next in Step2, the information of gap locations is generated from NG-clones detected in Step1. Then in Step3, we visualize the locations of gaps and NG-clones. Finally in Step4, a user analyses the source code using the visualized results.

As a sample input for explanation of the algorithm, we use the following two sequences.

code sequence X: “`ABCDCDEFBCDG`”,

code sequence Y: “`ABCEFBCDEBCD`”

Here, symbols “`A`”,“`B`”,“`C`”,`...` are code portions in a certain unit such as character, token, line, statement, function, etc. The algorithm explained in this section can be adapted to any presupposed NG-clone detection algorithms regardless of the unit that they use. An implemented tool (used in the later experiment) uses token as the unit. Also, though the sequences X and Y are different as a whole, when X equals completely to Y, the algorithm is the same. We consider the detection to be performed among multiple files (programs) or within a single file (program).

Step1 (NG-clone detection)

Detect NG-clones from input source code. Next, sort the NG-clones as the preparation to identify gap locations efficiently.

Table 1 shows the sorted list of NG-clones in the sample. This table contains all NG-clones regardless of their length, e.g., length of $\{$`c6`$\}$ is 5, $\{$`c1`, `c2`, `c3`, `c5`, `c7`$\}$ are 3, $\{$`c4`$\}$ is 2.

Even such small strings (Sequences X and Y: their lengths are 12) have seven NG-clones. Easily guessed from this sample, large-scale software will have tremendous number of short NG-clones. Moreover, almost of them are short coincidental ones because of the existence of simple statements such as substitution and such information is practically useless. So, practical clone detection tools often do not detect short clones whose length is smaller than a threshold.

In the below explanation, we assume that a minimum length of NG-clones (hereafter, threshold1) are specified. Threshold1 will also become a minimum length of identical sub-sequences in gapped-clones, whose locations are finally identified in this process. Conversely, it is needed for a user to specify threshold1 much smaller than the minimum size of the gapped-clones that he/she expects to detect. The tool that we have implemented has an option to specify threshold1.

The reason to sort the detected NG-clones in this step is to identify gap locations effectively in Step2 that uses a kind of divide-and-conquer strategy and an optimization. Each code portion of clones is enclosed within an single file, that is, does not exist over two or more files. Therefore, a set of all clone pairs can be divided into subsets, in which all clone pairs have two code portions found in the same pair of source files. Sorting of clone pairs is done for each of such subset of clone pairs.

Here, without loosing generality, suppose that all NG-clone pairs in the sample are included in a single such subset, that is, all code portions of NG-clone pairs in the code sequence X exist in a source file X and all code portions of NG-clone pairs in the code sequence Y exist in a source file Y (file X $\neq$ file Y). Positions of NG-clones in file X are the primal key of sorting (a clone pair which appears previously within a file appears previously also within a sorted list), and positions of NG-clones in file Y are the second key of sorting. In the case that file X = file Y, one of two code portions of a NG-clone pair which appears previously within the file is the primal key and the other code portion is the second key. Table 1 shows a list of clone pairs after the sorting.

Step2 (Identification of gap location)

Generate gap locations from sorted NG-clones.

The detailed algorithm of this step is shown as a pseudo code in Figure 4. The data saved to each gap

Table 1. Detected result of NG-clone

NG-clone ID	code sequence X	code sequence Y	matched string
c1	1 - 3	1 - 3	“ABC”
c2	2 - 4	6 - 8	“BCD”
c3	2 - 4	10 - 12	“BCD”
c4	5 - 6	11 - 12	“CD”
c5	5 - 7	7 - 9	“CDE”
c6	7 - 11	4 - 8	“EFBCD”
c7	9 - 11	10 - 12	“BCD”

Positions of code portions are shown by indexes of starting and ending symbols. For example “6 - 8” means that the portion starts at 6th symbol and ends at the 8th symbol.

are: location of the gap on source code and two NG-clones at both ends of the gap. These data are used in Step3.

Table 2 shows all the generated gap data in the sample. If you illustrate these gaps (and NG-clones) in a scatter plot, it will looks like Figure 5(a), in which gaps are named `g1`, ..., `g7`.

The `threshold2` shown in Figure 4 represents the upper limit of the length of each gap that is permitted to be included in gapped-clones.

The optimization in Figure 4 makes use of the fact that NG-clones are stored as a sorted list (`sortedNgCloneDB`). In the inner `for` loop, once a clone pair with a distance of more than `threshold2` from a clone `ci` is found, there is no chance to find any other clone within a distance `threshold2` in the rest of the list. If we can suppose that `threshold2` is enough smaller than length of input source code and that there is no significant deviation in density of NG-clones in scatter plot, then the number of NG-clones within a certain distance from a NG-clone is considered up to a certain constant. Therefore, the overall time complexity of this algorithm is $O(n)$ (n: number of NG-clones) because each execution of the inner loop takes a constant time.

Table 2. Gap data

gap ID	code sequence X	code sequence Y	length
g1	4	4 - 6	3
(g2)	4	4 - 10	7
g3	4 - 6	-	3
(g4)	4- 8	4 - 9	6
g5	-	9 - 10	2
(g6)	5 - 8	9	4
g7	8	-	1

```
// for each NG-clone
for (i = 0; i < ngCloneTotalCount; i++)
{
  NgClone ci = sortedNgCloneDB.get(i);

  // search connection targets
  for (j = i; j < ngCloneTotalCount; j++)
  {
    NgClone cj = sortedNgCloneDB.get(j);

    // if NG-clone ci and cj are near
    dx = distance(ci.codeInX, cj.codeInX);
    dy = distance(ci.codeInY, cj.codeInY);
    if (((0 <= dx) && (dx < threshold2)) &&
        ((0 <= dy) && (dy < threshold2)))
    {
      // create new gap information
      Gap newGap = new Gap(ci, cj);
      gapDB.add(newGap);
    }
    else
    {
      // optimization
      if (distance(c.codeInX, d.codeInY)
                                  >= threshold2)
        break; // goto next i’s loop
    }
  }
}

// Function "distance(x, y)" computes the
// difference from end position of code portion x
// to start position of code portion y.
// The result may be zero or minus when x and y
// are overlapping or when y appears ahead of x.
```

Figure 4. Gap identification algorithm

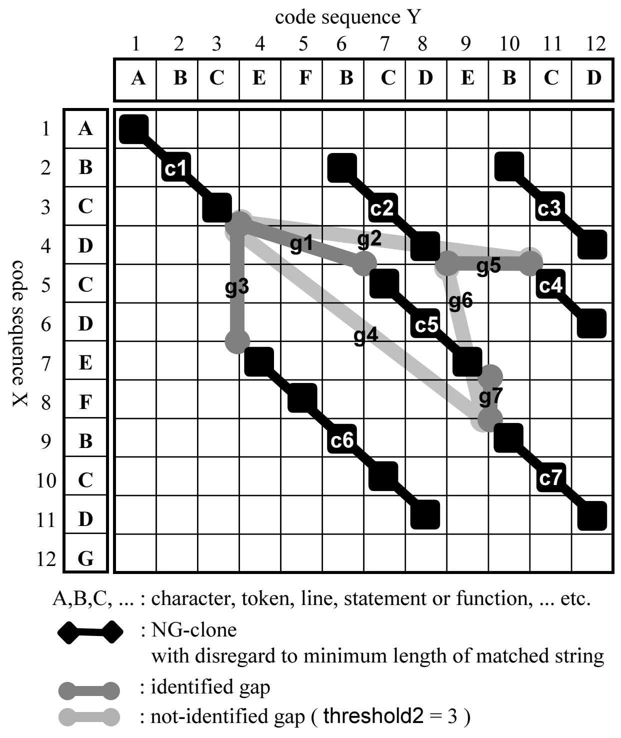
We think that some discussions are needed about how to decide value of `threshold2`. However, we use a constant as `threshold2` in our experiments of the later section for the present. As another choice, we could decide the value based on NG-clones around each gap, since too large gap in comparison of NG-clones may make a very “riddled” gapped-clone including small NG-clones and large gaps.

In the example, assume that value of `threshold2` is set to 3, which makes that `g2`, `g4` and `g6` are not detected.

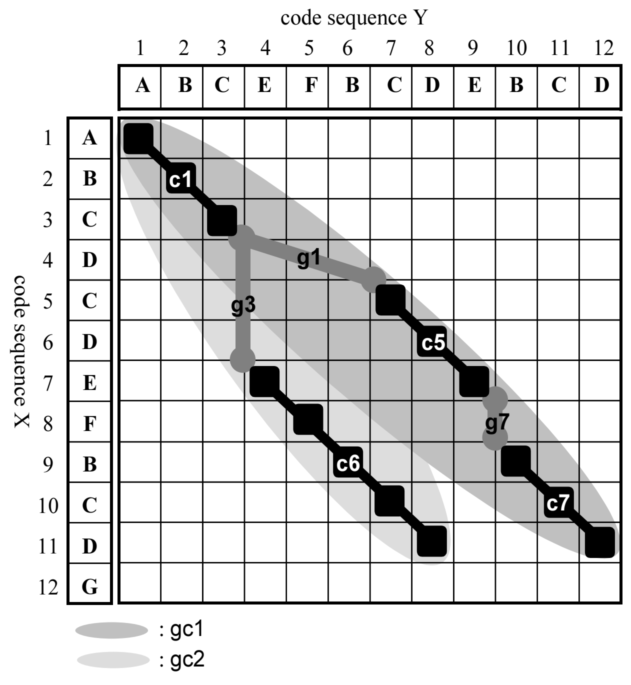
Step3 (Visualizing the location of gaps and NG-clones)

Step3-1 (gap-and-clone scatter plot)
Draw a scatter plot of gaps NG-clones

To visualize gapped-clones in a pseudo way, draw de-



(a) before filtering



(b) after filtering

Figure 5. Gap-and-clone scatter plot

tected NG-clones and gaps into a scatter plot (hereafter, *gap-and-clone scatter plot*). Figure 5(a) shows the gap-and-clone scatter plot of the sample.

In a gap-and-clone scatter plot, both axes are index of symbols and a NG-clone or a gap is represented as a line segment. In the figure, the vertical axis holds indexes of tokens in file X and the horizontal axis holds indexes of tokens in file Y. Each NG-clone is a black line segment which is penetrating black squares that are identical tokens within the clone's two code portions on file X and Y. Each gap is a gray line with circles at both ends, at which it is touching code clones.

By picking up NG-clones and their neighbor gaps by turns, you can make a gapped-clone. Table 3 shows such paths (i.e. gapped-clones) of NG-clones and gaps in the example. Gaps `g2`, `g4` and `g6` are not detected in this example by `threshold2`, so that gapped-clones including them are not detected.

Step3-2 (Filtering)
Remove NG-clones and gaps that do not contribute to make a gapped-clone.

Although this step is an optional, but it improves sharply visibility of gap-and-clone scatter plots.

Figure 5(a) shows all possible NG-clones (`c1`, ..., `c7`) and gaps (`g1`, ..., `g7`) detected from code sequence X and Y, when the thresholds for size are not used. When you analyze a large-scale source code, such gap-and-clone scatter plot has too many small gaps and clones to be understood. You can specify appropriate `threshold1` and `threshold2` to control the number of NG-clones and gaps, but these thresholds are not to distinguish long gapped-clones from short ones. For example, if you specify `threshold1` to be 3, then all NG-clones and gaps except `c4` , `g2` and `g5` will appear. If you specify a little larger value, say, 5, then only one NG-clone `c6` and no gaps remain.

To select gapped-clones indirectly by the possible size of them, another parameter is introduced, which is the minimum size of each entanglement of NG-clones and

Table 3. Gapped-clone path list

ID	path	code sequence X	code sequence Y
`gc1`	`c1 g1 c5 g7 c7`	"`ABC-CDE-BCD`"	"`ABC---CDEBCD`"
`gc2`	`c1 g3 c6`	"`ABC---EFBCD`"	"`ABCEFBCD`"
`gc3`	`c2 g5 c4`	"`BCDCD`"	"`BCD--CD`"

A gap is shown as one or more hyphens. The number of them represent the size of the gap, that is, distance of the NG-clones of both ends of the gap.

gaps. This size means the upper limit of gapped-clone included by the entanglement.

The calculation of the size of entanglement is: First, divide set of NG-clones and gaps into subsets, in which a NG-clone and a gap are touched directly or indirectly in a recursive way. Second, for each subset `s` of NG-clones and gaps, let us `sStartX` be the position of code portion of the NG-clone that appears at the foremost of X and `sStartY` be one of Y. Let us `sEndX` be the position of code portion of the NG-clone that appears at the rearmost of X and `sEndY` be one of Y. Then define `sSize`, which is the size of `s`:

$$\texttt{sSize} = max(\ \texttt{sSizeX}\ ,\ \texttt{sSizeY}\ ),$$
$$\texttt{sSizeX} = \texttt{sEndX} - \texttt{sStartX},$$
$$\texttt{sSizeY} = \texttt{sEndY} - \texttt{sStartY}$$

The third parameter `threshold3` is used to select the entanglements of NG-clones and gaps by size `sSize`, that is, the possible size of the largest gapped-clone size in the entanglement. In the example, if you specify `threshold3` to be 8, the remaining NG-clones and gaps are like in Figure 5(b).

Step4 (Analysis of source code)
Investigate source files with gap-and-clone scatter plot, changing parameters.

A GUI tool of gap-and-clone scatter plot provides location information of NG-clones and gaps specified by a user. The user can refer to a gap-and-clone scatter plot and locations of NG-clones, pick up the range including gapped-clones that he/she gets interested, and investigate the corresponding source files.

The user also modify parameters to change how short NG-clones or gaps to be considered as candidate of gapped-clones and how long gapped-clones should be. The following parameters are used in the above interactive investigation:

Threshold1 Minimum size of NG-clone in NG-clone detection

Threshold2 Maximum size of gap in identification of gap location

Thershold3 Minimum size of entanglement of NG-clones and gaps

These thresholds decide trade-off between computation complexity and rate of gapped-clone overlooked, and effect of each threshold depends on the thresholds above it. For example, the smaller `threshold1` makes the long time to detect NG-clones. But if you want to find gapped-clones that have code portions containing many modifications and each unmodified part is short one, then you have to set small value to `threshold1`. On the other hand, when you want to find gapped-clones with a few gaps, which may be easily merged into a single routine, you can specify not-so-small value to `threshold1`. Anyway, either `threshold1` or `threshold2` greatly affects computation time. Small `threshold1` or large `threshold2` makes the computation time a square order because small `threshold1` makes $O(l^2)$ clone pairs detected from size-l source code and large `threshold2` makes $O(n^2)$ gaps detected from n clone pairs.

3.2 Advantage

The advantage of the method proposed in this paper is that it supports its user to find gapped-clones in interactive and visual way with a gap-and-clone scatter plot which needs $O(n \log n)$ computation time, without computation of exact locations of gapped-clones that takes $O(n^2)$ time.

3.3 Implementation

We have implemented the proposed algorithm by extending a GUI maintenance support tool Gemini [19], which has multiple views to display a scatter plot of code clones, metrics of clone classes, and source files including the code clones specified by a user. Gemini uses the tool CCFinder internally to detect NG-clones. Figure 6 shows snapshots of Gemini.

On the view of gap-and-clone scatter plot (Figure 6(a)) implemented in Gemini, a user can execute zooming, selection of a NG-clones within a mouse-dragged rectangle, and its corresponding entanglement of NG-clones and gaps (Figure 6(b)).

4 Application to the programming exercise

4.1 Overview

We have applied our approach to source programs developed in a certain programming exercise of Osaka University.

In the exercise, each student writes a compiler in C language, which translates a program written in the subset of Pascal language into a certain assembly language.

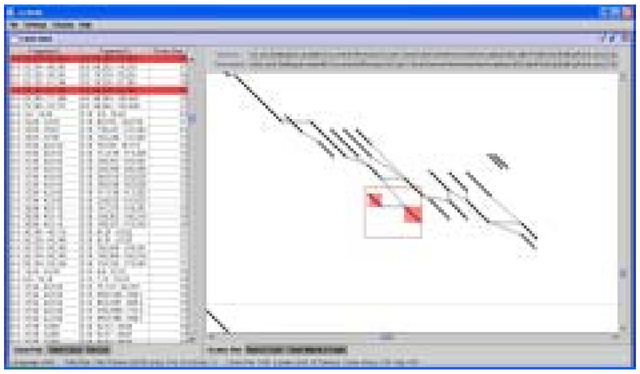
The exercise consists of three steps (sub-exercises):

Step1(Ex.1): Making a syntax checker($Parser$).

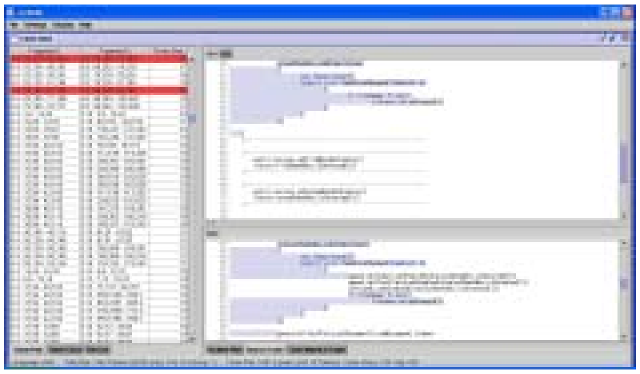
Step2(Ex.2): Making a semantic checker($Checker$).

Step3(Ex.3): Making a compiler(SPC).

In addition, it was required that $Checker$ and SPC are developed by reusing the code of the previous program. That is, $Checker$ is developed by reusing $Parser$ and SPC is developed by reusing $Checker$. So, each student would have frequently conducted ‘copy-and-paste’



(a) gap-and-clone scatter plot



(b) corresponding code

Figure 6. Snapshots of Gemini

programming and then slightly modified the pasted code portions. Thus, a lot of gapped code clones would be contained in the programs.

We collected the programs ($Parser$, $Checker$ and SPC) from 69 students. Totally, the size of all the programs is about 360,000 lines.

4.2 Analysis

In order to evaluate the usefulness of the proposed approach, we analyze the following items:

(1) Usefulness of gap-and-clone scatter plot: In order to confirm the usefulness of the gap-and-clone scatter plot, we repeatedly change the values of minimum size of code clone and entanglement of NG-clones and gaps in the filtering. Then, we checked the how many gapped-code clones are detected which include several short NG-clones.

(2) Type of gapped-clone found in gap-and-clone scatter plot: We examine the actual code portions that are appeared as gapped-clones through the gap-and-clone scatter plot. Also, we confirm whether the gapped code clones such as one in Figure 1 exist or not.

We analyzed the programs collected from 69 students. Among them, we show the distinctive results for the programs collected from a student S who conducted program reuse skillfully in the experiment [19].

4.2.1 Usefulness of gap-and-clone scatter plot

Figure 7 shows three scatter plots of a student S's three programs $Parser$ (2267 tokens), $Checker$ (4394 tokens), SPC (5738 tokens); each scatter plot contains clones detected from three programs. Scatter plot (a) contains NG-clones with 10 or more tokens (hereafter, such scatter plot with NG-clones and without gaps is called *NG-clone scatter plot*). The enormous black dots in this plot mean that there are so enormous short NG-clones that a 'fine-grained' analysis may be possible, if you were to endure investigating all the clones. Then, if you decided to observe only big NG-clones, say, with 30 or more tokens, you will get Scatter plot (b), where you can do rapid analysis by concentrating on small number of long clones but may miss many instances of copy-paste-modify cloning.

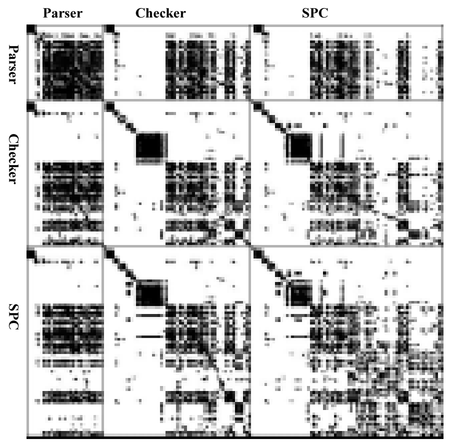
The gap-and-clone scatter plot can provide fine-grained and rapid analysis, with less computation complexity than gapped-clone detection. Figure 7 (c) is a gap-and-clone scatter plot of S's three programs with parameters `threshold1` $= 10$, `threshold2` $= 10$, and `threshold3` $= 30$, that is, the NG-clones with 10 or more tokens and the gaps less than 10 tokens that possibly constitutes gapped-clones with 30 or more tokens.

The trade-off in NG-clone scatter plot and trade-up by gap-and-clone scatter plot can be explained quantitatively. Figure 8 shows histograms of NG-clones in NG-clone scatter plot and in gap-and-clone one. The left histogram means that most NG-clones in Figure 7(a) have length from 10 to 30 tokens and the small number of NG-clones with 30 or more tokens also appears in Figure 7(b). The right histogram means that about two thousand of such NG-clones with length from 10 to 30 tokens may contribute to make gapped-clones with 30 or more tokens.

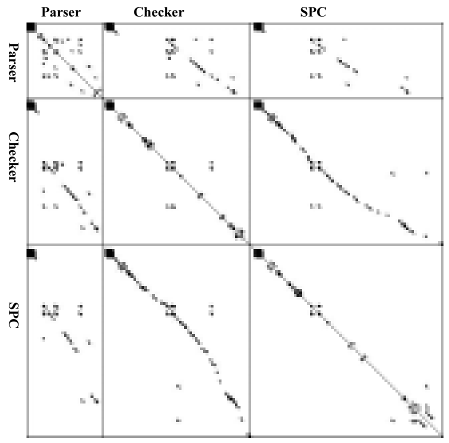
The above observations are not accidental ones, all programs other than S's produce the similar results.

4.2.2 What type of gapped-clone was found ?

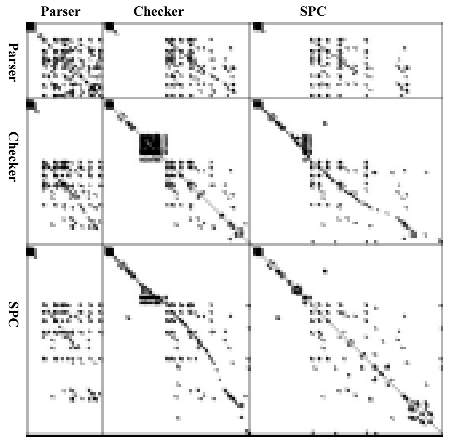
Figure 9 contains three versions of a function "`void sentence()`" in $Parser$, $Checker$, and SPC of S, and a gap-and-clone scatter plot of these source texts, in which `threshold1` $= 10$, `threshold2` $= 10$,



(a) NG-clone scatter plot (threshold1 = 10 token)

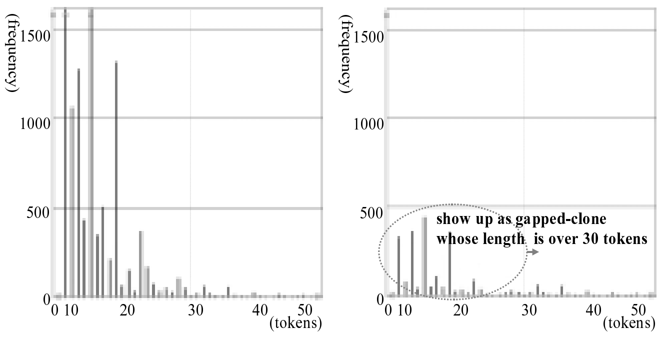


(b) NG-clone scatter plot (threshold1 = 30 token)



(c) Gap-and-clone scatter plot (threshold3 = 30 token)

Figure 7. Comparison between gap-and-clone scatter plot and NG-clone scatter plot



NG-clones in Fig. 7(a) and (b)

NG-clones in Fig. 7(c)

Figure 8. Histogram of NG-clone size

`threshold3` $= 20$. Each mark (A, B, C, etc) shows correspondings between a line segment on the plot and code portions with gray backgrounds.

Mark E on the plot contains three NG-clones E_1, E_2, and E_3 and gaps among them. These gaps correspond to the added code portions between two versions, $Checker$ and SPC, which expands functionalities of function `sentence()`. That is precisely what we have expected to observe, a gapped-clone created by modification. This gapped-clone E does not appear in a NG-clone scatter plot of NG-clones with 20 or more tokens, since each of the three NG-clones in E has length less than 20 tokens.

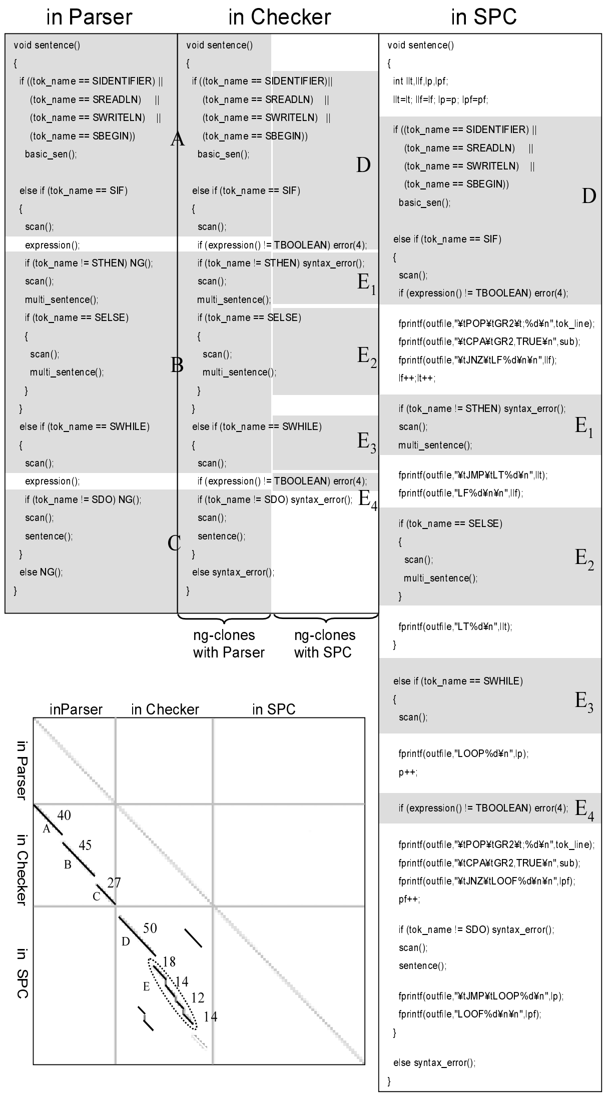
In this experiment, a NG-clone detection tool CCFinder does not report a code portion which begins at the middle of a function and ends at another function as a clone, but it divides such a portion by the function boundary and reports two smaller code portions. These code portions are desired to remain isolated, but currently they look as if a gapped-clone on gap-and-clone scatter plot. This problem will be resolved by modifying CCFinder to report such boundaries and adding a rule to Gemini; a gap does not bridge across any boundary.

5 Conclusions

In this paper, we have proposed the method to show the gapped-clone based on the information of the gap location identified by using the detection results of NG-clone. Then, we have implemented the method into our maintenance support environment Gemini. Finally, in order to evaluate whether the proposed method can detect the gapped clone effectively, we have applied it to the data collected from the programs developed in the series of three exercises. As the results, we have successfully found the gapped-clones, which are composed of several short clones each of which is too short to appear individually. The gapped clones are surely ones that we wanted to identify as described in sub-section 2.3.

However, we have just shown the gapped code clones on the gap-and-clone scatter plot. Currently, we have no mechanisms to evaluate the characteristic of each of the gapped code clones quantitatively. It is necessary to examine the method to extract efficiently the each gapped code clone based on the location information of gap and NG-clone.

Several studies also have proposed gapped-clones detection methods. However, there are some differences among the definitions of clones. So, we are going to examine the difference and compare other methods to our approach.



in Parser

```
void sentence()
{
  if ((tok_name == SIDENTIFIER) ||
      (tok_name == SREADLN)    ||
      (tok_name == SWRITELN)   ||
      (tok_name == SBEGIN))
    basic_sen();

  else if (tok_name == SIF)
  {
    scan();
    expression();
    if (tok_name != STHEN) NG();
    scan();
    multi_sentence();
    if (tok_name == SELSE)
    {
      scan();
      multi_sentence();
    }
  }
  else if (tok_name == SWHILE)
  {
    scan();
    expression();
    if (tok_name != SDO) NG();
    scan();
    sentence();
  }
  else NG();
}
```

in Checker

```
void sentence()
{
  if ((tok_name == SIDENTIFIER)||
      (tok_name == SREADLN)   ||
      (tok_name == SWRITELN)  ||
      (tok_name == SBEGIN))
    basic_sen();

  else if (tok_name == SIF)
  {
    scan();
    if (expression() != TBOOLEAN) error(4);
    if (tok_name != STHEN) syntax_error();
    scan();
    multi_sentence();
    if (tok_name == SELSE)
    {
      scan();
      multi_sentence();
    }
  }
  else if (tok_name == SWHILE)
  {
    scan();
    if (expression() != TBOOLEAN) error(4);
    if (tok_name != SDO) syntax_error();
    scan();
    sentence();
  }
  else syntax_error();
}
```

in SPC

```
void sentence()
{
  int llt,llf,lp,lpf;
  llt=lt; llf=lf; lp=p; lpf=pf;

  if ((tok_name == SIDENTIFIER) ||
      (tok_name == SREADLN)    ||
      (tok_name == SWRITELN)   ||
      (tok_name == SBEGIN))
    basic_sen();

  else if (tok_name == SIF)
  {
    scan();
    if (expression() != TBOOLEAN) error(4);

    fprintf(outfile,"¥tPOP¥tGR2¥t;%d¥n",tok_line);
    fprintf(outfile,"¥tCPA¥tGR2,TRUE¥n",sub);
    fprintf(outfile,"¥tJNZ¥tLF%d¥n¥n",llf);
    lf++;lt++;

    if (tok_name != STHEN) syntax_error();
    scan();
    multi_sentence();

    fprintf(outfile,"¥tJMP¥tLT%d¥n",llt);
    fprintf(outfile,"LF%d¥n¥n",llf);

    if (tok_name == SELSE)
    {
      scan();
      multi_sentence();
    }

    fprintf(outfile,"LT%d¥n",llt);
  }

  else if (tok_name == SWHILE)
  {
    scan();

    fprintf(outfile,"LOOP%d¥n",lp);
    p++;

    if (expression() != TBOOLEAN) error(4);

    fprintf(outfile,"¥tPOP¥tGR2¥t;%d¥n",tok_line);
    fprintf(outfile,"¥tCPA¥tGR2,TRUE¥n",sub);
    fprintf(outfile,"¥tJNZ¥tLOOF%d¥n¥n",lpf);
    pf++;

    if (tok_name != SDO) syntax_error();
    scan();
    sentence();

    fprintf(outfile,"¥tJMP¥tLOOP%d¥n",lp);
    fprintf(outfile,"LOOF%d¥n¥n",lpf);
  }

  else syntax_error();
}
```

Figure 9. Gapped-clones in source code history of S's "void sentence()"

References

[1] B.S. Baker, "A Program for Identifying Duplicated Code", *Computing Science and Statistics*, 1992, 24:49-57.

[2] B.S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems", *Proceedings the 2nd Working Conference on Reverse Engineering*, 1995, 86-95.

[3] B.S. Baker, "Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance", *SIAM Journal on Computing*, 1997, 26(5):1343-1362.

[4] M. Balazinska , E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis, "Advanced Clone-Analysis to Support Object-Oriented System Refactoring", *Proceedings the 7th Working Conference on Reverse Engineering*, 2000, 98-107.

[5] M. Balazinska , E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis, "Measuring Clone Based Reengineering Opportunities", *Proceedings 6th IEEE International Symposium on Software Metrics*, 1999, 292-303.

[6] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees", *Proceedings IEEE International Conference on Software Maintenance-1998*, 1998, 368-377.

[7] S. Ducasse , M. Rieger, and S. Demeyer. "A Language Independent Approach for Detecting Duplicated Code", *Proceedings IEEE International Conference on Software Maintenance-1999*, 1999, 109-118.

[8] M. Fowler, *Refactoring: improving the design of existing code*, Addison-Wesley, 1999.

[9] D. Gusfield, *Algorithms on Strings, Trees, And Sequences*, Cambridge University Press, 1997.

[10] J. Helfman, "Dotplot Patterns: a Literal Look at Pattern Languages", *TAPOS*, 1995, 2(1):31-41.

[11] J.H. Johnson, "Identifying Redundancy in Source Code using Fingerprints", *Proceedings CASCON'93*, 1993, 171-183.

[12] 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*, 2002, 28(7):654-670.

[13] R. Komondoor, and S. Horwitz, "Using slicing to identify duplication in source code", *Proceedings 8th International Symposium on Static Analysis*, 2001.

[14] J. Krinke, "Identifying Similar Code with Program Dependence Graphs", *Proceedings 8th Working Conference on Reverse Engineering*, 2001, 562-584.

[15] J. Mayland, C. Leblanc, and E. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", *Proceedings IEEE International Conference on Software Maintenance-1996*, 1996, 244-253.

[16] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto, "Software Quality Analysis by Code Clones in Industrial Legacy Software", *Proceedings 8th IEEE International Symposium on Software Metrics*, 2002, 87-94.

[17] M. Rieger, and S. Ducasse, "Visual Detection of Duplicated Code", *Proceedings ECOOP'98 Workshop on Experiences in Object-Oriented Re-Engineering*, 1998, 75-76.

[18] T.M. Pigoski, "Maintenance", *Encyclopedia of Software Engineering*, 1994, 1:619-636.

[19] Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue, "Gemini: Maintenance Support Environment Based on Code Clone Analysis", *Proceedings 8th IEEE International Symposium on Software Metrics*, 2002, 67-76.

[20] S.W.L. Yip, and T. Lam, "A Software Maintenance Survey", *Procceedings 1st Asia-Pacific Software Engineering Conference*, 1994, 70-79.