International Journal of Computational Intelligence Systems, Vol. 6, No. 6 (November, 2013), 1143-1150

Synthesis Algorithm of Homology Detection Based on AHP

Baojiang Cui*, Wenhao Fan, Ce Bian
School of Computer Science, Beijing University of Posts and Telecommunications
Beijing, China
E-mail: cuibj@bupt.edu.cn

Tao Guo, Yongle Hao
China Information Technology Security Evaluation Center
Beijing, China
E-mail: guotao@itsec.gov.cn

Jianxin Wang
School of Information, Being Forestry University
Beijing, China
E-mail: wangjx@bjfu.edu.cn

Received 16 March 2013

Accepted 29 March 2013

Abstract

Traditional software homology detection techniques based on text, token, abstract syntax tree in many cases get the real similarity inaccurately when they work alone. In this paper, a synthesis algorithm based on Analytic Hierarchy Process (AHP) is proposed, which combines text, token, syntax tree comparison algorithms synthetically according to their contributions to performance factors such as omission ratio and fall-out ratio. Numerous results indicate that the synthesis algorithm reflects software homologous similarity more accurately.

Keywords: software homology, Analytic Hierarchy Process, synthesis algorithm.

* School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, 100876, Beijing, China

1. Introduction

Software design and development is a knowledgeable high-technology industry which is growing fast and updating the society. With the rapid development of software industry, especially the popularities of open source software, software's copy right faces heavier and heavier thread because of its properties such as high portability and various forms. The growing number of software plagiarism seriously damages the original author's legitimate rights and interests. Homology detection is obliged to fight against intellectual property crime.

Currently most software source code homology detecting techniques are based on text, token and abstract syntax tree (AST). In early days, Baker[1] and Johnson[2] used text-based comparison algorithm based on string and fingerprints matching. It is the first homology detection method on text level. The current typical text-based detection methods are line and block comparison ones which support single-line and multi-line text comparison to detect homologous similarity. The characteristics of text-based comparison algorithm are simple to implement, low fall-out ratio, high omission ratio and weak anti-interference. It can only detect simply complete plagiarism[3]. The existing homology detection tools based on text are UltraCompare, WinDiff, WinMerge, etc.

Token-based comparison algorithm first deletes the useless characters in the source code, does lexical analysis on source code, and then replaces source code terms with special token flags. Thus the source code is changed into token sequence. Finally the algorithm detects homologous similarity by comparing the hash value of the token sequences. Token-based comparison algorithm has much stronger anti-interference capability compared with text-based algorithm, but it doesn't take structure information of source code into account. Furthermore, token-based method can only make key field replace plagiarism detectable[4] and it has a high fall-out ratio. Token-based comparison algorithm is widely used in famous homology detection tools such as CP-Miner[5], CCFinder[6], Winnowing[7], JPlag[8] and so on.

Syntax tree-based algorithm first preprocesses the code to generate the syntax tree which contains the syntax structure information of the code[9]. Then it calculates and compares the hash values of target and sample syntax tree to judge homology. Algorithm based on syntax tree completes source code analysis on syntax level and takes syntax structure of source code as judge basis. So it has characteristics of good anti-interference capability, can detect lots of plagiarism modes while it has a particular error detection because of its Hash-based comparison[10].

According to analysis above, each of these existing homology detection algorithms has both advantages and disadvantages to detect homology of software while none of them can get the optimal performance. To achieve better detection result, a synthesis homology detection algorithm based on Analytic Hierarchy Process (AHP) is proposed in this paper. This algorithm balances the contribution of homology detection algorithms based on text, token and abstract syntax tree to performance factors such as omission ratio and fall-out ratio, and reasonably reflects the real similarity more accurately.

This synthesis homology detection algorithm based on AHP will be presented in the following sections: Section 2 compares the contribution of text-based, token-based and AST-based homology detection algorithms to performance factors such as omission ratio, fall-out ratio and performance factors to synthesis similarity. Section 3 describes the implementation process of AHP synthesis algorithm. Experiments in section 4 prove that when code size is large and plagiarism modes are complex, the result of the algorithm based on AHP is much closer to the real one than any result of text-based, token-based, AST-based algorithms, which shows that AHP-based synthesis algorithm is more suitable to practical environment

2. Contribution Comparison and Analysis

Code similarity of software reflects their homology qualitatively and quantitatively. There are many evaluation factors to evaluate algorithms' detection result. Considering their correlation and overlap, factors are summed up as omission ratio and fall-out ratio. Algorithms based on text, token, AST have different performances on different performance factors. Synthesis homology detection algorithm based on AHP proposed here optimizes performance factors of synthesis similarity, and also reflects the real similarity of source code more accurately.

Contributions of the three existing homology detection algorithms to the performance factors and contributions of performance factors to synthesis similarity are analyzed in this section.

2.1. *Contribution analysis to synthesis similarity*

Software code similarity is defined as the proportion of the homologous part in all code. Errors are unavoidable in homology detection due to the inherent limitation of the algorithms and varied plagiarism modes. The performance factors considered here are omission ratio (O) which shows missed detection and fall-out ratio (F) which shows error detection.

Assuming one software source code detection result is as following:

Table1.Code detection result example

detection result / artificial result	similar code	dissimilar code
similar code	a	c
dissimilar code	b	d

Detection similarity: $S_{det} = \frac{a+b}{a+b+c+d}$

Theoretical similarity: $S_{theo} = \frac{a+c}{a+b+c+d}$

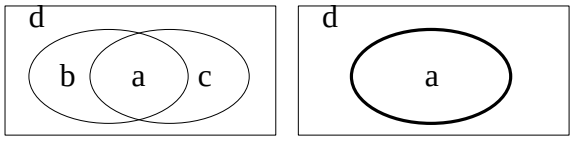
Omission ratio: $O = \frac{c}{a+c}$

Fall-out ratio: $F = \frac{b}{a+b}$

The assumption here is that ignoring external factors, there is an ideal homology detection algorithm, whose detection result has no missed detection and false positive detection, namely, $b = 0$, $c = 0$, then,

$$Sim_{det} = \frac{a+b}{a+b+c+d} = Sim_{theo} = \frac{a+c}{a+b+c+d}$$

This following figure can present this process more intuitively.



(a) in practical case (b) in ideal case

Fig. 1.Homology detection result

So we can conclude that for any given positive number ε, there are positive numbers δ_1, δ_2, such that if $0 < |O| < \delta_1, 0 < |F| < \delta_2$, then $|Sim_{det} - Sim_{theo}| < \varepsilon$, then the limit for $\mathrm{Sim_{det}}$ will be Sim_{theo}.

$$Sim_{theo} = lim_{O \to 0, F \to 0} Sim_{det} \quad (1)$$

Defective detection algorithm produces flawed performance factors, leading to the deviation between detected and theoretical similarity. It is necessary to synthesize text, token, AST-based homology detection algorithms in a certain strategy which can balance and optimize the three performance factors for the purpose to draw detected similarity near theoretical one.

How much does synthesis similarity depend on these performance factors? It is in proportion to the probability of negative performance factors appearing in detected code. Considering of homology detection application fields of this paper, omission ratio is the most important factor. Manual confirmation is indispensable after detection so that fall-out ratio is relatively less significant than omission ratio. Statistical analysis of experiments demonstrates the importance ratio of omission ratio to fall-out ratio is 4:1.

2.2. *Contribution analysis to performance factors*

Factors we mentioned here include omission ratio and fall-out ratio. Omission ratio and fall-out ratio measure the synthesis similarity negatively. The higher the omission ratio and fall-out ratio is, the lower the accuracy of synthesis similarity is. When comparing the contributions of text, token, AST-based detection algorithms to omission ratio and fall-out ratio, this paper has considered the effect of detection algorithms on the synthesis similarity positively.

2.2.1*Contributions to omission ratio*

When it comes to omission ratio, it is considered that the better detection algorithm is, the more plagiarism modes it can detect. Common plagiarism modes can be summarized as: complete copy; identifier rename; data type change; code segment sequence change; parameter sequence change, and so on. Table 2 shows the statuses of text, token and AST-based homology detection algorithms of common plagiarism modes.

For common plagiarism modes, text-based detection algorithm can detect only one mode, token-based one can detect four modes and AST-based one can detect eight modes. So the importance ratio of text, token,

AST-based algorithms is 1:4:8 contributing to omission ratio.

Table 2. Contribution analysis of text, token and AST homology detection algorithms to omission ratio

Plagiarism modes \ Algorithm	text	token	AST
Complete copy	detectable	detectable	detectable
Rename identifier	missing	detectable	detectable
Change data type	missing	detectable	detectable
Redefine data type	missing	detectable	detectable
Change code segments sequence	missing	missing	detectable
Change parameters sequence	missing	missing	detectable
Multi-line code reconstruction	missing	missing	detectable
Equivalent structure replacement	missing	missing	detectable
Add unrelated statements or variables	missing	missing	missing

2.2.2*Contributions to fall-out ratio*

For fall-out ratio, algorithm quality lies in the misjudgment of particular statements which are not homologous. Table 3 shows the statistical data of text, token and AST Homology detection algorithms to common false detection modes.

Table 3. Contribution analysis of text, token and AST homology detection algorithms to fall-out ratio

Misjudgment modes \ Algorithm	text	token	AST
Short expression error	correct	error	correct
Short statements block error	correct	error	correct
Hash value confliction error	correct	correct	error

For common misjudgment modes, there is little false detection using text-based detection algorithm. Token-based algorithm detects one mode without mistake and AST-based one does two. So the importance ratio of text, token, AST-based algorithms is 3:1:2 contributing to fall-out ratio.

3. Analytic Hierarchy Process Synthesis Algorithm

Analytic Hierarchy Process (AHP) is a mathematical modeling method. The basic idea is that in the premise of achieving goals the contributions of detection algorithms to performance factors will be compared quantitatively and delivered layer by layer. Relative importance weight from the bottom to the top layer is confirmed[11].

The synthesis homology detection algorithm proposed in this paper is based on text, token and AST, and considers the performance factors such as omission ratio and fall-out ratio. This synthesis algorithm also confirms the contributions of the three algorithms to synthesis similarity according to their respective advantages, reflecting the reasonable real one more accurately.

The formula of synthesis similarity is as below,

$$Sim = w_{txt} Sim_{txt} + w_{token} Sim_{token} + w_{AST} Sim_{AST} \quad (2)$$

In Eq. (2), $Sim, Sim_{txt}, Sim_{token}, Sim_{AST}$ stand for similarities of synthesis, text, token and AST-based similarities. $w_{txt}, w_{Token}, w_{AST}$ stands for weight of text, token and AST-based detection algorithm.

Model and weights of the three detection algorithms in synthesis homology detection are introduced in the rest of this section.

3.1. *Structure of AHP synthesis homology detection hierarchy analysis model*

The structure of AHP hierarchy analysis model is establishing the evaluation factor system to analyze objects systematically. It includes goal layer, rule layer and solution layer. Goal layer is the achieving goal of the analyzed object, rule layer is the middle part to achieve the goal and solution layer is the execution solution chosen objectively.

In previous analysis, text, token and AST-based detection algorithms have respective advantages in performance factors of synthesis similarity. The

performance factors reflect the accuracy of detection similarity directly. In the synthesis homology detection model, the goal layer is the synthesis similarity. The rule layer is performance factors evaluating synthesis similarity. The solution layer is text, token and AST-based detection algorithms. Fig. 2 is the AHP synthesis homology detection model.

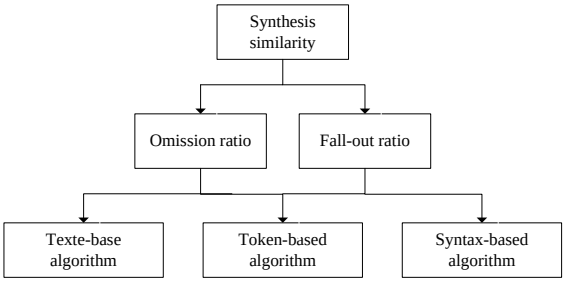


Fig. 2.AHP synthesis homology detection model

3.2. *Judgment matrix*

The Judgment matrix is the importance matrix whose element value stands for the relative contribution that the same layer factors make to the prior layer factors. The importance degrees of relative contribution are measured by divisions from one to nine as is shown in Table 4.

The judgment matrix $A=(a_{ij})_{m\times n}$ has the following properties,

$a_{ij}>0$,

$a_{ij}=1/a_{ji}$ when $i\neq j$,

$a_{ij}=1$ when $i=j$.

In above description, a_{ij} is the importance ratio between i and j.

Table 5. The contribution judgment matrix of text, token and AST-based detection algorithms to omission ratio

B1	C1	C2	C3
C1	1	1/4	1/8
C2	4	1	1/5
C3	8	5	1

Table 6. The contribution judgment matrix of text, token and AST-based detection algorithms to fall-out ratio

B2	C1	C2	C3
C1	1	3	2
C2	1/3	1	1/2
C3	1/2	2	1

Table 7. The contribution judgment matrix of omission ratio, fall-out ratio to synthesis similarity

A	B1	B2
B1	1	4
B2	1/4	1

According to the argument of the second part in this paper, the importance degree method is used to define that the degree is m-n+1 when m>n and it is 1/(n-m+1) when m<n if the ratio of the two factors is m:n. The contribution judgment matrixes of text, token and AST-based detection algorithms to the performance factors omission ratio, fall-out ratio and the contribution judgment matrix of omission ratio, fall-out ratio to synthesis similarity are shown in Table5-7.

3.3. *Hierarchy weight*

Hierarchy weight is divided into single layer weight and synthesis layer weight. Single layer weight is defined as

Table 4. Importance degree meaning

Importance degrees	meaning
1	The first factor is the same important with the other.
3	The first factor is a little more important than the other.
5	The first factor is much more important than the other.
7	The first factor is intensively more important.
9	The first factor is extremely more important.
2,4,6,8	The median of degrees above.

weight of one layer factors to the prior layer ones. The synthesis layer weight is the weight of solution layer factors to the goal layer. The single layer weight is necessary to calculate the whole layer weight.

The procedure of single weight calculation is as follows,

- Normalize each column of the judgment matrix.

$$\bar{a}_{ij}=\frac{a_{ij}}{\sum_{k=1}^{n} a_{kj}}(i=1,2,\cdots,n)$$

- Sum up each row of the normalized matrix.

$$\overline{W_i}=\sum_{j=1}^{n} \bar{a}_{ij}(j=1,2,\cdots,n)$$

- Normalize the vector $\overline{W}=[\overline{W_1},\overline{W_2},\cdots,\overline{W_n}]$.

$$W_i=\frac{\overline{W_i}}{\sum_{i=1}^{n}\overline{W_i}}(i=1,2,\cdots,n)$$

Hierarchy single layer weight vector is as bellow：

Table 8. Hierarchy single layer weight vector

B1	single weight	B2	single weight	A	single weight
C1	0.07	C1	0.54	B1	0.80
C2	0.21	C2	0.16	B2	0.20
C3	0.72	C3	0.30		

$$C.I.=\frac{\lambda_{max}-n}{n-1} \quad (4)$$

In the above equation λ_{max} stands for the maximum eigenvalue of judgment matrix and n stands for the order of the judgment matrix.

- Confirm the relative average random index *R.I.*

Inquire random index Table 9 of the judgment matrix to get *R.I.*

- Calculate consistency ratio *C.R.*

$$C.R.=\frac{C.I.}{R.I.} \quad (5)$$

The judgment matrix is acceptable when *C.R*<0.1. It is not when *C.R.*>0.1, in that case it is necessary to modify the judgment matrix.

Consistency ratios of the four judgment matrixes above are 0.0922, 0.0089, 0, which are acceptable.

3.5. *Synthesis layer weight*

The procedure of the synthesis layer weight calculation is as follows,

The synthesis layer weights of m factors in the (k-1)*th* layer to the goal layer are $w_i^{(k-1)}$.

The single weights of n factors in the k*th* layer to the j*th* factor of the k-1*th* layer are $p_j^{(k)}=(p_{1j}^{(k)},p_{2j}^{(k)},\cdots,p_{nj}^{(k)})$.

Table 9.Average consistency random index R.I. table the result of 1000 time positive reciprocal matrix calculating

matrix order	1	2	3	4	5	6	7	8
R.I.	0	0	0.52	0.89	1.12	1.26	1.36	1.41
matrix order	9	10	11	12	13	14	15	
R.I.	1.46	1.49	1.52	1.54	1.56	1.58	1.59	

3.4. *Consistency inspection*

Consistency inspection can use mathematical method to confirm the rationality of judgment matrixes in the following procedures.

- Calculate the maximum eigenvalue of judgment matrixes.

$$\lambda_{max}=\frac{1}{n}\sum_{i=1}^{n}\frac{(AW)_i}{W_i} \quad (3)$$

- Calculate the consistency index *C.I.*

The synthesis weights of factors in the k*th* layer to the goal layer are calculated using Eq. (6).

$$w_i^{(k)}=\sum_{j=1}^{n} p_{ij}^{(k)} w_j^{(k-1)}(i=1,2,\cdots,n) \quad (6)$$

The synthesis layer weight vector is as bellow:

Table 10. Hierarchy synthesis layer weight vector

A	C1	C2	C3
synthesis weight	0.16	0.20	0.64

They are the weights of text, token and AST-based detection algorithms to synthesis similarity. The formula for synthesis similarity calculation is shown in Eq. (7).

$$Sim=0.16Sim_{txt}+0.2Sim_{Token}+0.64Sim_{AST} \quad (7)$$

4. Experiment Analysis

The synthesis similarity calculation model proposed in this paper considers different quality of text, token and AST-based detection algorithms, and reasonably reflects the real similarity of expert calculation more accurately in the practical application environment of multifarious plagiarism means.

4.1. *Simulative plagiarism experiment on different plagiarism method*

Simulative plagiarism of open source software emule-0.42 is taken for example here. In experiments, common plagiarized methods including variable name change, function name change, type redefine, parameter sequence change, code sequence change and synthesis plagiarism method are designed to detect open source software emule-0.42. The result of AHP synthesis detection algorithm and text, token and AST-based ones are shown in Table 11. The formula of synthesis error rate is shown as Eq. (8).

$$e_{syn}=\frac{|synthesis\ similarity\text{-}reasonable\ real\ similarity|}{reasonable\ real\ similarity} \quad (8)$$

From this result we can see that algorithm based on text, token, AST have respective advantages to different single plagiarism method which support the rationality of contribution analysis in section 2. However, for synthesis plagiarism method, synthesis similarity is closer to the reasonable one compared with the other three, which will get further validation in the next experiment.

Table 11. Experiment on different plagiarism method

plagiarism mode \ similarity	text similarity	token similarity	syntax similarity	synthesis similarity	reasonable similarity	synthesis error rate
variable name change	30.79%	62.72%	55.28%	52.85%	56.11%	5.80%
function name change	57.47%	68.78%	61.37%	62.23%	60.97%	2.06%
type redefine	58.94%	70.95%	53.95%	58.15%	58.03%	0.21%
parameter sequence change	68.05%	79.89%	69.38%	71.27%	68.46%	4.10%
code sequence change	41.30%	53.41%	40.53%	43.23%	41.13%	5.11%
synthesis method	54.51%	64.89%	54.51%	56.59%	55.22%	2.47%

In the experiments, there are tens of thousands lines of source code with simulative plagiarisms such as copy-paste, parameter name change, function name change, type redefinition and multi-line code sequence change. Several miss detection code segments are added for verification of text, token and AST-based algorithms. Experts in the homology field calculate line number of artificial simulation plagiarized code. The reasonable real similarity is the ratio of it to the whole number.

4.2. *Simulative plagiarism experiment on different software*

In experiments, synthesis plagiarized methods including copy-paste, parameter name change, function name change, type redefinition and code sequence change are designed to detect a large number of open source software such assnort-2.9.0.3, mysql++3.1.0, ucos2.86, junit4.7. The result of AHP synthesis detection algorithm and text, token and AST-based ones are shown in Table 12.

Plentiful experiments illustrate that compared with any of text, token and AST-based algorithms, synthesis homology detection algorithm based on AHP gains a more accurate similarity and gets a lower error rate when there are a great quantity of detected code and various plagiarism means.

Table 12.Experiment on different software source code

Similarity / Detected code	text similarity	token similarity	syntax similarity	synthesis similarity	reasonable similarity	synthesis error rate
snort-2.9.0.3	53.87%	72.62%	60.64%	61.95%	62.32%	0.59%
mysql++-3.1.0	31.67%	52.20%	43.06%	43.07%	42.51%	1.31%
ucos-2.86	45.11%	51.57%	47.50%	47.93%	48.47%	1.11%
junit-4.7	33.24%	52.34%	44.02%	43.96%	43.35%	1.41%
ossec-hids-2.5.1	44.50%	62.74%	60.22%	58.21%	59.02%	1.37%
libprelude-1.0.0	52.36%	70.28%	58.98%	60.18%	60.67%	0.81%

5. Conclusion

This paper calculates contribution weights of the three algorithms on synthesis similarity by contribution analysis based on AHP synthesis homology detection model in which performance factors omission ratio and fall-out ratio are taken into account. The synthesis similarity calculation model balances different performance quality of text, token and AST-based detection algorithms on performance factors omission ratio and fall-out ratio. This model reasonably reflects the real similarity more accurately by means of artificial simulative plagiarism and experts' calculation in the homology field.

It is verified by artificial simulation of various plagiarisms and homology detection that homology detection algorithm based on AHP approaches the reasonable similarity more closely compared to text, token and AST-based detection algorithms. The more complex the detected codes and plagiarisms are, the more accurately the homology detection algorithm based on AHP gets the detection result.

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 61170268, No. 61272493 and No. 61100047).

References

1. Brenda S. Baker, A Program for Identifying Duplicated Code, in Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.
2. J Howard Johnson, Identifying Redundancy in Source Code Using Fingerprints, in Proceedings of the 1993 Conference of the Centre for Advanced Studies Conference(CASCON'93), pp. 171-183, Toronto, Canada, October 1993.
3. S. Ducasse, M. Rieger and S. Demeyer, A Language Independent Approach for Detecting Duplicated Code, in Proceedings of the 15th International Conference on Software Maintenance, ICSM 1999, pp. 109-118.
4. K. Roy and J.R. Cordy, A Survey on Software Clone Detection Research, Queen's School of Computing TR 2007-541, 115 pp. 64-68, 2007.
5. Z. Li, S. Lu, S. Myagmar and Y. Zhou, CP-Miner: A tool for finding copy-paste and related bugs in operating system code, in OSDI, pp. 289–302. 2004
6. Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, (2002, July), CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 28(7):654–670.
7. S. Schleimer, D. S. Wilkerson and A. Aiken, Winnowing: local algorithms for document fingerprinting, in Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03), pp. 7685, San Diego, California, June 2003.
8. Lutz Prechelt, Guido Malpohl, and Michael Philippsen, Finding plagiarisms among a set of programs with JPlag, in Journal of Universal Computer Science, 8(11):10161038, November 2002.
9. I. Baxter, A. Yahin, L. Moura and M. Anna, Clone Detection Using Abstract Syntax Trees, in ICSM, pp. 368-377, 1998.
10. S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, Comparison and Evaluation of Clone Detection Tools, IEEE TSE,33(9):577-591, 2007.
11. Thomas L. Saaty, Decision making with the analytic hierarchy process, International Journal of Services Sciences(Volume 1, Number 1 / 2008) ,pp.83-89.