Application of Modified Memetic Algorithm to Uncover Authorship Styles in Software Forensics

Y. Manas Kumar, L.Yamuna, S.R.Y .Himatej

***Abstract:** Our paper sincerely advocates a memetic algorithm to uncover authorship styles. For software forensics experts our proposed mechanism will greatly reduce the time, effort whenever a malicious job is done to break into a software system. We have considered three factors, namely the variable naming convention, usage of comment styles. We have considered three factors, namely the variable naming convention, usage of comment styles, usage of data structures. We observe that these 3 factors can greatly help to uncover authorship style of a pro-grammer thus saving us from further damage in this technologically dependant society.*

***Keywords:** Software forensics, Memetic, Authorship, nearness value, genetic.*

I. INTRODUCTION

Adversaries in software industry exists in forms which are quite difficult to analyses software forensics is counted up as one of them .the most popular software threats are viruses, logic bombs, Trojans worms which leave the functionality of a software as a regret after their attack is completed damage detection happens only after the adversaries finish their task intrusion detection engineers face this uphill challenge of reporting the damage done to the software researchers have found that 70% of adversaries who crack the software leave behind some code in software forensics field this leftover traces of code are analyzed to get an insight of the nature of the programmer. Multiple factors effect a person's programming style, so to establish an authorship style is not easy as said. In this paper we will look into the factors through which we can obtain styles of coders. The main issue is lack of robust formal methods (or) tools to meet this challenge ,no method of discouraging anonymity in software system is full proof given contexts which change with respect time .Talking of remnants of an attack viruses generally deposit their code in source files, object code executable code .In software forensics domain, this code acts as evidence which is used to verify source of the attack .this corresponds to how legal officials work with handwriting analysis to identify suspects who may be involved in such crimes.

Revised Manuscript Received on January 25, 2019.

Y. Manas Kumar, Assistant Prof. Department of Computer Science, Pragati Engineerining College(A), Kakinada City India.

L.Yamuna, Assistant Prof. Department of Computer Science, Pragati Engineerining College(A), Kakinada City India.

S.R.Y .Himatej, B.Tech IV Year, Department Of Computer Science, Pragati Engineerining College(A), Kakinada City India.

in programming languages few languages heavily use Data types control structures in innovative way which gives scope for authors to develop their unique styles while developing software. While development reuse debugging of code usage of certain stylistic elements help a lot. Also, the gravity of this challenge increases when two (or) more authors collaborate to develop a single module of a software. In such cases the authorship styles are mixed and tracing individual authorship is impossible. to start the process of identification samples of

code are kept for observation if samples of code amount significantly then statistical methods can be easily applied to find authorship of the code but our immediate issue is that code after compilation (or) optimization may not look exactly same as the original source code .so, reverse engineering such codes may induce considerable amount of ambiguity which effects analysis at further stages. Other feature which has to be seriously looked into is choice of data structures and algorithm. beginners would not prefer advanced data structure with which they are comfortable .in same sense choice of algorithms also rests with authors based on their competence level complex algorithm, even if they are time saving is not picked up by large number of coders for obvious reasons, the next feature in contention for authorship analysis is usage of error handling methods. it has been found by practitioners that seldom such codes occur in case they occur, it is due to developmental policies enforced by team managers which is part of routine error checking individual authors mostly neglect exception handling routines in their code. The next feature is choice of system calls while providing support to code .this scenario mostly occurs in UNIX like environment. For instance index, strchr methods are part of two different versions of UNIX each author gets habituated only to usage of the same method while coding the last feature but definitely the most importance one we consider in this paper is nature of errors made by authors. some authors make same kind of errors consistently for example off by one error in loops for arrays or while referencing dereferencing pointers after proper comparison of such error this can be used as a metric in uncovering authorship style. Memetic algorithms have derived strength from computational intelligence.

They inherit some principles of evolutional computing mechanisms but in a stricter sense they cannot be considered as evolutionary technique. They are motivated by integration of genetic evolution with memetic evolution.

The term meme is used to refer to a piece of discrete cultural data which can be easily subjected to process of selection and variation of the evolutionary techniques.The main philosophy of memetic algorithms is individual improvements plus population co-operation .the strategy is to make good use of all available source of knowledge. It performs a global search to find good area in search space by taking help from a local search heuristic repeating this process until the objective function is met is the main crux of the algorithm. The main issue is to maintain balance between the local & global mechanisms of search so that the premature convergence of the system does not happen thereby saving substantial amount of computational resources. Apart from these issues the residing challenges is about learning should occur and when to stop the learning process .also issues like learning on which individual solutions should be used can take considerable effort while using memetic algorithms one of the application of memetic algorithms is pattern recognition. Pattern recognition techniques try to uncover patterns of input from data sources. We have used the ability of memetic algorithms to uncover authorship styles based on available patterns while developing a framework which will infer whether (or) not the styles of specific authors are matching to an extent. If exact match doesn't happen this framework indicates what percentage of authorship style is established. This helps us in concluding the most probable author of a module (or) entire software application.

Architectural principle of proposed Framework

We populate a database with authorship styles of known sources In software Forensics they may be left over code of suspects who may be an adversary for the software system .Then we develop a pattern of styles of coding which acts as the training data this training data is given as input to the framework to learn following operating principles of the memetic algorithm. The other input parameter which can be considered is extent to which the input author style is matched from the database content .this value is represented as nearness value. If nearness value is less than 20% then we say authorship style is uncovered to a known author whose style of coding resides in the database .The figure below represents the scheme of our approach.

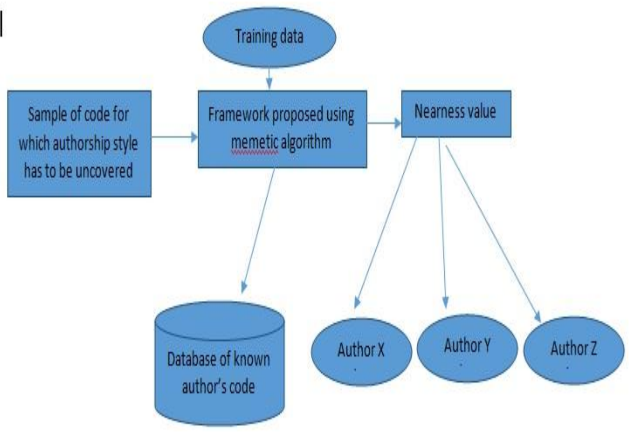
Training data
Sample of code for which authorship style has to be uncovered
Framework proposed using memetic algorithm
Nearness value
Database of known author's code
Author X
Author Y
Author Z

Fig. 1.: Proposed Framework using memetic algorithm

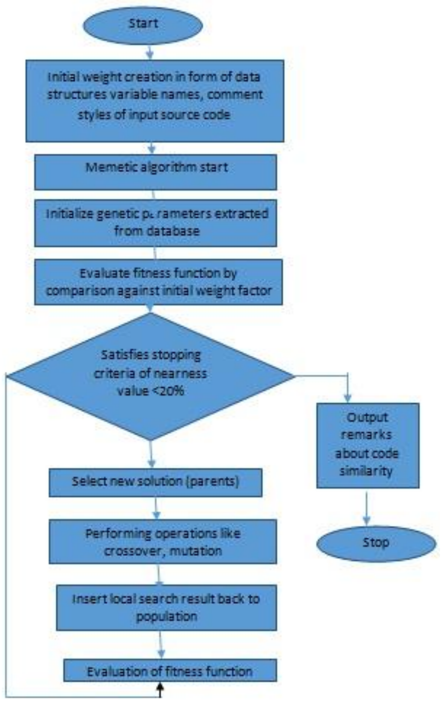
Start
Initial weight creation in form of data structures variable names, comment styles of input source code
Memetic algorithm start
Initialize genetic parameters extracted from database
Evaluate fitness function by comparison against initial weight factor
Satisfies stopping criteria of nearness value <20%
Output remarks about code similarity
Stop
Select new solution (parents)
Performing operations like crossover, mutation
Insert local search result back to population
Evaluation of fitness function

Fig. 2. Flowchart of proposed algorithm

2.1 Description of algorithm proposed in detail

Step 1: Starting the process by keeping available the source code for which authorship has to be established.

Step 2: Initial weight creation for optimization of objective function For this, data structures which are simple are given low-value like arrays=1 , linked lists=2 , trees=3, graphs=4 , matrices=5 etc

The sample code present in database is also coded with same vales for corresponding data structures. Similarly weights are assigned for length of variable names. Say, one character=1 unit weight two characters=2 units and so on. For comment style, single length comments are assigned 1 unit weight and multi-line comments are assigned 2 units weight.

Step 3: Now the memetic algorithm initiates by initializing the abovementioned values of each feature for available source code samples in database.

Step 4: fitness function is evaluated by comparison of the value obtained by adding weight values of each feature.

For instance, if samples code c1 is as follows:

```
{
…
int a [10];     //this is array of maximize
int b;
…
…
}
```

The weight values for above code are 1 unit (a [10]), 1 unit (//This is array of max size10), 2 units (two characters for variable name bb)

Fitness Function for c1=1+1+2=4 units

Step 5: From database we get similar values of weights we compute the difference between the weight values obtained in step 4, step 5.If nearness value is less than 20% between the two source codes we say both codes are similar and stop the algorithm.

If it does not happen we proceed to following steps.

Step 6: We select the top two source codes from database who have nearness value in range of 20-40% with respect to the sample-code.

Step 7: performing crossover, mutation over this two codes, say c2 ,c3.

code	C2	C3
Weight of data structure	2	2
Weight of comment style	2	1
Weight of variable name	3	2

code	C_{new}
Weight of data structure	2
Weight of comment style	2
Weight of variable name	2

We perform crossover function over weight of comment style of c2 with corresponding entry of c3 and weight of variable name of c2 with entry of c3 as they are the only weight entries which are different to get a new source code weight values like shown below.

C new is mutated version of c2 c3 after crossover operation finishes

Step 8: This above entry is added to the database and n fitness function evaluation is carried out again.

Step 9: Go back to step 5.

II. EVALUATION

We observe many aspects of authorship attribution in our evaluation: (1) the degree to which our techniques recover author style in program binaries, (2) the trade-offs involved in imprecise classification (i.e., tolerating some false positives), and (3) whether and how much stylistic clustering of one set of programs can be improved by using information derived from another, unrelated set. Our evaluation shows that: The binary code features we introduce effectively capture programmer style. Our classifier achieves accuracies of 81% for ten distinct authors (10% accuracy is expected for labels selected by random chance) and 51% when discriminating among almost 200 authors (0.5% for random chance). These results show that a strong author style signal survives the compilation process. – The authorship classifier offers practical attribution with good accuracy, if a few false positives can be tolerated. The correct author is among the top five 95% of the time for a data set of 20 authors, and 81% of the time when 100 authors are represented. – Stylistic knowledge derived from supervised authorship classification can be transferred to authorship clustering, improving cluster quality. The cluster assignments improve by 20% when clustering uses a stylistic metric.

3.1 Methodology

For our case we have obtained source code from the popular code repository, github. We have considered 10 authors, programmers as we can call them. We populated our database with code of these authors which were committed in the repository nearly 3 years back. The training data is applied to our proposed mechanism, so that a nearness value maybe achieved to uncover the authorship style. Subsequently we have taken a snippet of code which belongs to one of the 10 authors to validate our algorithm. The code snippet we take is the most recent one which was committed just 2 weeks ago. We have trained our proposed scheme as per comment style, type of data structure used, variable naming style. The nearness value gets greatly affected by application of our proposed scheme. The below results are tabulated which represent how authorship can be indicated by nearness degree.

Nearness Value(in %)
72
65
28
21
17
82
88
12
4
2

Other classification algorithm when trained with code which was recently developed. So, when we populate our database with codes, we must be careful about the history of the code. Code which is too old may end up in resulting a false positive thereby misleading the software forensics expert. To face this threat our mechanism has to be applied an appreciable number of times with respect to time intervals of say 2 years and the results of nearness value have to be carefully analyzed.

The next part of our experiment evaluation was how each of the 3 factors we used affected the nearness value when applied over a cluster of authors, the following results were obtained.

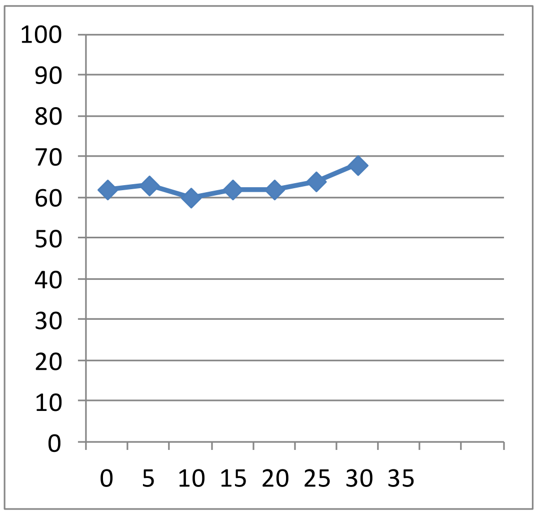

Fig. 3.Plot of no. of authors and nearness degree with variable naming convention

In the proposed mechanism with variable name convention we can clearly observe that the nearness value ranges from 60% to 70% over range of 30 authors. This leads us to the inference that variable naming convention does not help in uncovering authorship style .Using this factor alone is not much useful for the software forensics expert .So next we employ the usage of comment styles for same parameters and obtain the following result.

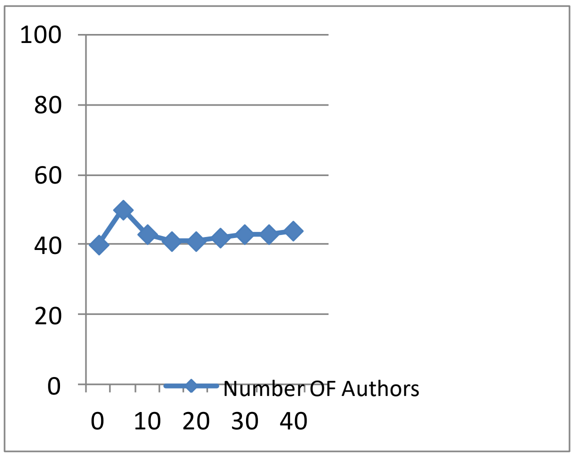

Fig. 4. Plot of no. of authors and nearness degree with the usage of comment styles

We observe that the nearness value more or less remains the same even though number of authors increase up to 40 .we can state easily that this factor alone is also failing to uncover a traceable author uniquely. This leads us to applying the proposed mechanism on similar number of authors using factors of data structures. We obtain following graphical result. It is evident that as number of authors increase there is a steep increase in the nearness value but we can also observe that after number of authors reaches more than 60,the nearness value tends to remain stable indicating the fact that most developers tend to use same data structure.

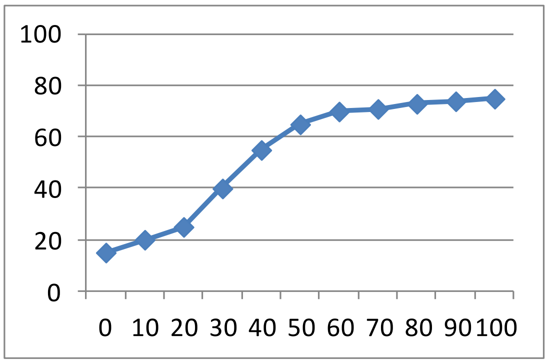

Fig. 5. Plot of no. of authors and nearness degree with the usage of similar data structures

To handle specific type of problems. our discussion is limited to identification of similar styles of programming .Hence, we can advocate a statement like none of the discussed three factors are strong enough alone to uncover a unique authorship style .so, combining the three factors gives us a better solution in hand. when all the three factors are used to train our memetic algorithm the chances to isolate limited number of authorship styles are greatly increased.

The database contain the history of codes should be maintained carefully by the database administrator if the code is tampered, the usage of our said mechanism cannot conclude successfully.

3.2 Challenges faced by proposed framework:

The first and most difficult issue to be faced is the amount of code compared .A small sized sample cannot give effective result. This challenge is aggravated when an author has reused code for obvious reasons to tackle this challenge our learning mechanism of the framework does not have any provision. The next challenges is if collaborative work has been done then a unique authorship style will be developed and that style cannot match to single (or) even multiple authors. In such cases our framework becomes ineffective Nevertheless the nearness value may predict a known authorship style another pertinent issue faced is availability of similar functional codes. If we try to match codes which are of different functionality say a boot record code with that of screen editor code even if author is same it is highly unlikely to find the correspondence. Hence choosing correct sample of code and populating the database remains a crucial task for the success of our proposed framework.

III. CONCLUSION

The framework we have advocated can be used to support many legal cases involving patent claims as well as trade secret. The utility of our proposed framework lies along with other rigorous approaches employed in software forensics which include mathematical tools, reverse engineering tools.

Studies must occur to validate the accuracy of the mechanism involved and a fair degree of tolerance for false negatives should also be in place .Exploring the true potential of memetic algorithm will be possible only when metrics for code comparison are robust which is in turn possible only when strict language development constructs are incorporated while developing applications. Further work in this framework is necessitated due to changing coding styles of authors in modern world.

REFERENCES

1. Abbass H (2001) A memetic Pareto evolutionary approach to artificial neural networks. Lecture Notes in Computer Science 2256: 1–12.
2. Aggarwal C, Orlin J, Tai R (1997) Optimized crossover for the independent set problem. Operations Research 45: 226–234.
3. Aguilar J, Colmenares A (1998) Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. Pattern Analysis and Applications 1: 52–61.
4. Beasley J, Chu P (1996) A genetic algorithm for the set covering problem. European Journal of Operational Research 94:393–404
5. Beasley J, Chu P (1998) A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4: 63–86
6. Becker B, Drechsler R (1994) Ofdd based minimization of fixed polarity Reed-Muller expressions using hybrid genetic algorithms. Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processor, pp 106–110
7. Berger J, Salois M, Begin R (1998) A hybrid genetic algorithm for the vehicle routing problem with time windows. Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, pp 114–127
8. Berretta R, Cotta C, Moscato P (2001) Forma analysis and new heuristic ideas for the number partitioning problem. Proceedings of the 4th MIC — Metaheuristic International Conference, pp 337–341
9. Frederick Mosteller and David L. Wallace. Applied Bayesian and Classical Inference: The Case 0/ the Pcdcmlist PapfTs. Springer Series in Statistics. Springer-Verlag, 1D64.
10. Herbert Solomon. Confidencc Intervals in Legal Settings1 pages 455-473. John Wiley & Sons, 1986.
11. Eugene H. Spafford. The Internet worm program: an analysis. Computer Communication RC1Jiew, 190), January 1989. Also issued as Purdue CS technical report TR•CSD-82:3.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication