Exploring Paraphrasing Techniques on Formal Language for Generating Semantics Preserving Source Code Transformations

Aviel J. Stein*‡ Levi Kapllani*‡ Spiros Mancoridis* Rachel Greenstadt†
*College of Computing and Informatics
Drexel University, Philadelphia, Pennsylvania 19144
Email: ajs568@drexel.edu lk489@drexel.edu
† Tandon School of Engineering
New York University, New York, New York 10003
Email: greenstadt@nyu.edu

Abstract— **Automatically identifying and generating equivalent semantic content to a word, phrase, or sentence is an important part of natural language processing (NLP). The research done so far in paraphrases in NLP has been focused exclusively on textual data, but has significant potential if it is applied to formal languages like source code. In this paper, we present a novel technique for generating source code transformations via the use of paraphrases. We explore how to extract and validate source code paraphrases. The transformations can be used for stylometry tasks and processes like refactoring. A machine learning method of identifying valid transformations has the advantage of avoiding the generation of transformations by hand and is more likely to have more valid transformations. Our data set is comprised by 27,300 C++ source code files, consisting of 273 topics each with 10 parallel files. This generates approximately 152,000 paraphrases. Of these paraphrases, 11% yield valid code transformations. We then train a random forest classifier that can identify valid transformations with 83% accuracy. In this paper we also discuss some of the observed relationships between linked paraphrase transformations. We depict the relationships that emerge between alternative equivalent code transformations in a graph formalism.**

I. Introduction and Overview

In everyday life, people use paraphrasing to convey information they have heard or read using different words from those used by the original writer or speaker. We paraphrase for better clarity or just to respect the authorship of the writer or speaker. In Section II we discuss how this process has been explored in NLP and has been crucial to many applications such as information extraction, information retrieval, query and pattern expansion, summarization, question answering, machine translation, semantic parsing, etc. [1] Additionally, paraphrases are more effective at retaining semantic correctness and imitating author style than other machine learning techniques including generative adversarial networks (GANS).

In Section III, we discuss some of the differences between formal language and natural language. Natural languages have more redundancy and ambiguity, but share attributes such as grammar, syntax, and words. Section IV covers how machine learning approaches such as Random Forests, Deep Learning, GANS attribution, and Imitation have also been applied to computer source code effectively. However, there has not been any research in source code paraphrasing, despite its potential. In this work, we explore source code paraphrasing by defining and generating source code paraphrases and exploring the potential advances source code paraphrasing would open. We loosely define semantically equivalent code paraphrases as those that result in programs that pass exactly the same test cases as the original code. Source code paraphrasing can be used to simplify code, generate new ways to solve problems, create pseudo-code, translate between programming languages, and support authorship attribution. The transformations we generate in this work could be used for tasks such as refactoring, optimization, as well as authorship attribution, obfuscation, and imitation in training under adversarial conditions. We posit that these transformations can be used in such tasks as GANS for authorship obfuscation, code refactoring, and guideline enforcing.

In Section V, we consider texts to be parallel if they are both solutions to a problem as validated by a test suite. We use the Levenshtein edit distance [2] to measure the likelihood that two lines of code are paraphrases. Once we have found the candidate lines of source code, we verify that semantics were preserved for the task by comparing the outputs of a test suite that is assumed to exist for the original (non-paraphrased code).

In Section VI, our method employs the use of parallel corpora of source code that solve the same problem. The Google Code Jam challenge problems, which is widely used in NLP for source code stylometry, also fits well for this inquiry. This paraphrase method results in the generation of approximately 4,500 candidate phrases and yields approximately 11% semantic-preserving code transformations. The time required to generate these transformations depends on the number of semantically valid phrases.

In Section VII, we evaluate our method by training a random forest classifier to distinguish between valid and invalid lines

‡The authors contributed equally to this work.

of code. We discuss the impact this work may have on NLP and Formal Language Processing (FLP) in Section VIII. There are various ways to judge the quality and use of our generated paraphrases and we recommend some. We describe future work and its benefit of this method. Section IX concludes the paper by presenting the important findings, namely that paraphrase techniques work with formal languages to produce source code transformations and that models can be made to evaluate transformations automatically.

II. Natural Language Processing

The analysis of text is useful in the context of the Internet, where data drenched forums abound. Many of the tasks trained and tested on in this space are of news and social media data. Language models are built in order to extract useful information from these resources. [3] Language models are systematic approaches in the development of useful features for machine learning algorithms. [4] The model can build relationships between words and phrases through an analogy method. For example, a model can identify pairs which may be similar. To determine the extent that two pairs overlap in their meaning, we present the following analogy. Clothing:shirt and dish:bowl is: "clothing is to shirt as dish is to bowl," and ask how valid it is. [5] Machine learning algorithms can use the features generated by these models to make predictions. Dolan et al present a study in which human judges with 448 sentence pairs generated by these algorithms believed approximately 26% were correct, 34% were partially correct and 40% were not related. [6] Stylometry explores the extent to which style (writing, music, art, etc) can be measured for attribution, imitation, obfuscation and other purposes. For example, you can build a language model for distinguishing authors. This task can be applied to many authors with high accuracy and has even been used to imitate and obfuscate their styles. Moreover, previous work used a sequence-to-sequence language model, inspired by machine translation, that was able to retain some of the semantics of the original text. [7] The generative adversarial network tries to imitate the style of another author while retaining the original semantics. While these methods can fool a machine learning model, they do not guarantee semantic equivalence so they could be easily identified by a human judge.

III. Natural versus Formal Languages

Natural language has evolved over time and continues to do so. Even though there are tools for communication, confusion and ambiguity sometimes is impossible to avoid. Formal languages are precise tools that we have created to solve these limitations. Formal languages have been developed and used by scientists who want to formulate specific rules about what they are trying to communicate. They have grammar, vocabulary, and syntax that are less ambiguous and redundant. There is still some debate about the specific qualities of formal languages. In 2018, the International Journal of Computer Mathematical Sciences published an article discussing the difference between formal languages and natural languages stating that "formal languages don't have semantics. So in formal languages, importance is given to syntax only, not to the meaning of the sentence".[8] However, all source code is written in formal languages and the NLP methods that utilize the semantics of language can be used on them. What is important to take away from this section is that formal languages still have a well-defined structure, are defined using grammar, vocabulary in the form of tokens, and semantics as defined by the mapping of well-formed syntactic structures to executable machine instructions. In contrast to NLP, we will refer to methods applied to formal languages as formal language processing as FLP.

IV. Formal Language Processing

The overlap between NLP tasks and code analysis is increasing. Stylometry techniques generate similar results for both source code and textual data. Language-agnostic methods can apply fairly simple language analysis such as term frequency inverse document frequency (TF-IDF) to build a language model with the ability to perform accurate authorship. [9] Even binary code can contain traces of style. With source code, the method collects features from abstract syntax trees (ASTs), layout, and lexical features, and achieves a 94% accuracy when testing the style of 1,600 different authors [10]. Generating paraphrases can be used to change the features extracted, potentially obfuscating or imitating in an adversarial context. However, semantic preserving GANS have been able to reduce the attribution accuracy of traditional stylometry methods significantly. [7] They relied on 5 types of code transformation: Control, Declaration, API, Template, and Miscellaneous, using ASTs, control flow graphs, use-define chains, and declaration reference mappings for generating files. Their obfuscation reduced attribution accuracy to 1%. Code transformations are crucial for GANS because they rely on generating new data to try to misdirect the detective network and improve it. [11] The generation of paraphrases has the potential to be used to change the features extracted. Given the success of Code Stylometry we are hopeful that paraphrasing can have an impact on privacy and code improvement as well.

V. FLP Paraphrases

We explore how readily paraphrasing techniques can be applied to formal languages. Some applications may be other FLP tasks or code analysis but this area has yet to be explored. We build our technique by analogy to NLP approaches. These approaches vary by the type of corpus and goals for the results. To find semantic preserving code transformations, we use methods appropriate for a monolingual parallel corpora.

A survey of the most frequently used methods for generating phrasal and sentential paraphrases by NLP researchers in the past two decades categorizes approaches by corpus types. [10] Four techniques used on monolingual parallel corpora are presented, two of which rely on distributional similarity while the other two focus on more direct non-distributional methods. [12] [13] [1] [14] The last approach uses statistical machine translation tools to generate novel paraphrases using different

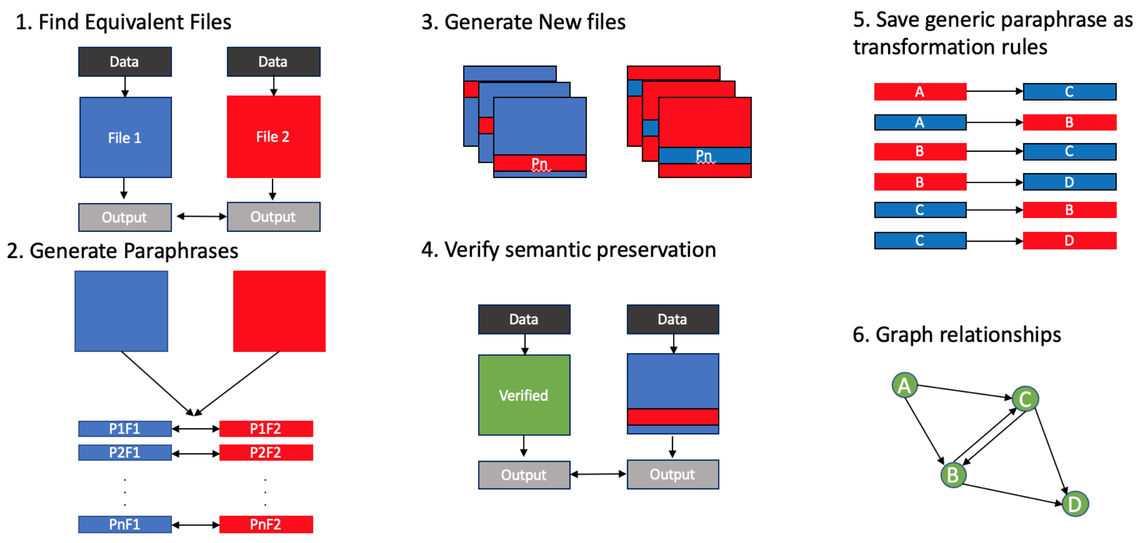

Fig. 1. Our method of identifying, validating, and comparing paraphrases.

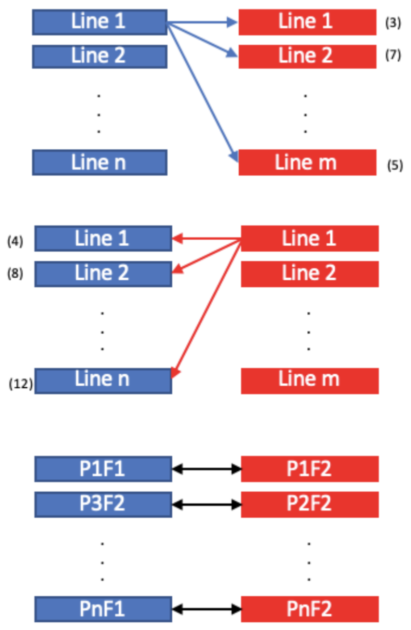

Fig. 2. Computing line edit distance between two files to identify paraphrases.

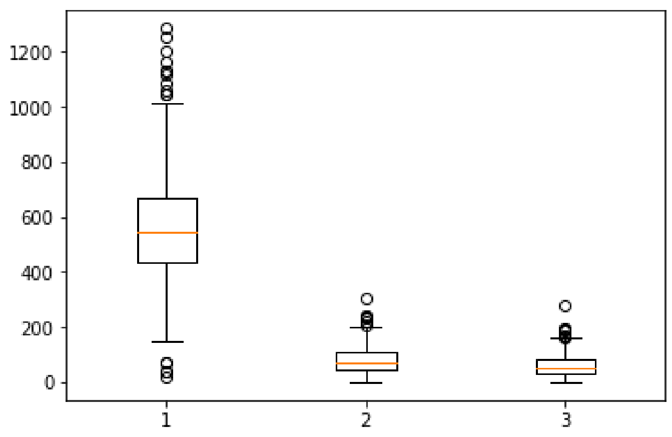

Fig. 3. Box plots of number of invalid (1), compileable (2), and valid (3) paraphrases.

news articles from the web, which is the approach we built our method on. Quirk et al. use the edit distance (Levenshtein distance) to compare pairwise sentences and determine new paraphrases. They use word alignment algorithms to improve this process and measure results.

If the dataset has several versions of the same text written in the same language, the corpus can be called parallel monolingual. A strict real-world example of a parallel corpus is famous books that have several translations in another language. The Illiad, for example, has many English translations. Each of them contain slight variations depending on the author that was involved in the translation, their interpretation, and the primary ideas they want to emphasize, but they are all written in English and derived from the same story. These texts are thereby parallel and can be used to extract English paraphrases. By the same token, we use source code solutions that solve the same problem, but are written by different programmers as our monolingual parallel corpora.

Next, to generate the paraphrases we could decompose each source code file to varying levels of granularity including source lines, statements, functions, classes, etc. We decided to begin at the source code line level because it is the first time this has been done and we decided to begin at the most elementary level of abstraction. Following other works, we decided to compute the Levenshtein edit distance between each pair of lines in two distinct files. [1] This distance is defined as the sum of all mappings, deletions, substitutions,

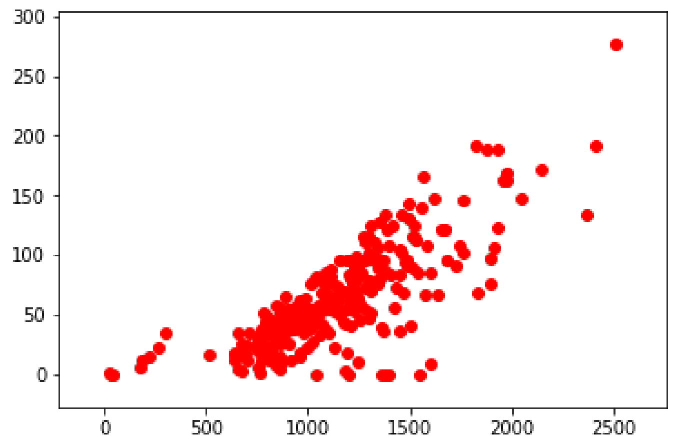

Fig. 4. Time required to extract valid paraphrases based on the number of candidate pairs, where the x-axis is time in seconds and the y-axis is the number of transformations extracted.

and insertions needed to ensure an equivalence between two lines. The cost of mapping a character is 0 and 1 for all other operations. We pair each line of the first script with the line from the second script which grants the smallest Levenshtein distance (see Figure 2). Similarly, we pair each line of the second script. Occasionally some threshold of distance is used as the filter by which to permit pairs. We decided to only consider lines pairs as potential paraphrases if they are mutually closest to each other and yet syntactically distinct. To validate the paraphrases, we must show that the phrases still obey the syntactic rules and do not cause run-time issues. To be useful as code transformations they also must retain the original behavior. To accomplish this, we can modify the solutions to use the new paraphrase instead. Variable names are a problem that arises. We must keep the original variable and function names unchanged. If the solutions are determined by their ability to pass a test suite then we can use the same test suite to determine whether it still constitutes an acceptable solution. If the paraphrased code produces the same output, we can assume that the phrase transformation is a viable code transformation. Additionally, we may find that many phrases can be compared and connected.

VI. Results

The Google Code Jam dataset has been valuable to code analysis research and has been used to develop and benchmark results under a controlled setting.[15] In this work, we show how to identify and validate semantically equivalent paraphrases between pairs of C++ files. Each of the solutions of the tasks in our data set is written in C++ and consists of approximately 50 to 200 lines. Our data set consists of 27,300 files consisting of 273 topics each with 10 parallel files i.e., those that solved the same challenge question. The computational complexity for computing the edit distance is $O(n^2)$. Using the initial C++ scripts and substituting some lines with the new paraphrases generated, creates new solutions to the problems solved by the initial two C++ scripts. Since the solution size per problem is 10 files, there were 90 permutations of size two, each producing paraphrases. To ensure this, we tokenize all paraphrase pairs into keywords, variables, numerical values, etc. using a set of regular expressions. For each syntactically unique paraphrase pair, a new pair is created by swapping all tokens of the two paraphrases, except variable and function names. We identified 152,658 candidate paraphrase pairs and filtered out syntactically identical pairs. Google Code Jam provides a website to practice with a test set to check if you solved the problem that was used to validate the pairs. Of the remainder 63,264 failed to compile, meet run-time requirement (5 seconds), or pass the test suite and 16,096 passed all of these filters.

To formalize the phraphrases into a graph form we generalize a paraphrase to have generic variables names such as a, b c. . . aa, ab, ac. . . zz, aaa, etc. We used each phrase as a node and the edge represents seeing a transformation from one to the other (see Figure 5). Many transformations occur frequently, but a large majority only appear once or twice. The expectation of a distribution closer to a normal distribution suggests that a larger data-set set should be used but also that there are both common practices and idiosyncrasies represented (see Figure 6). We were also curious about how dense the connected networks were (see Figure 7). The more nodes, the more chance for highly connected graph. But they remained relatively constant in their density. While this graph looks barren, many of the graphs are quite sparse. Figure 7 shows that most of the graphs have fewer than 10 nodes. The number of paraphrases and size of graphs are heavily skewed. Future studies consisting of a larger sample size may yield a more normal distribution.

VII. Evaluation

To speed up the paraphrase extraction process or extract paraphrases without a test suite we trained a Random Forest model on the verified transformations. Limited to just the Levenshtein Distance, we get 20% valid paraphrases, which is competitive with state-of-the-art NLP technique's semantic retention. We generalized the phrases by relabeling all the variable names to be a, b, c, . . . z, aa, ab, zz, aaa, aab, etc to avoid reputation and make it adaptable to new code. We were then able to use only unique transformations in our dataset. Additionally, we removed all instances in the overlap between positive and negative samples, which ended up being 3% of the data. Some of the overlap could be due to the context in, or time sensitivity of, the task. The dataset contained 7,300 positive instances and 28,374 negative ones which reduced the data-set to 44% size of original with similar distribution of positive to negative observations. We trained and tested a random forest model with 10-fold train test split and having 100 estimators and no max depth. From 10 trials of this experiment resulted in an average 83.7% and standard deviation of .03% we should expect 36% accuracy which surpasses NLP instances. The ROC curve in Figure 10

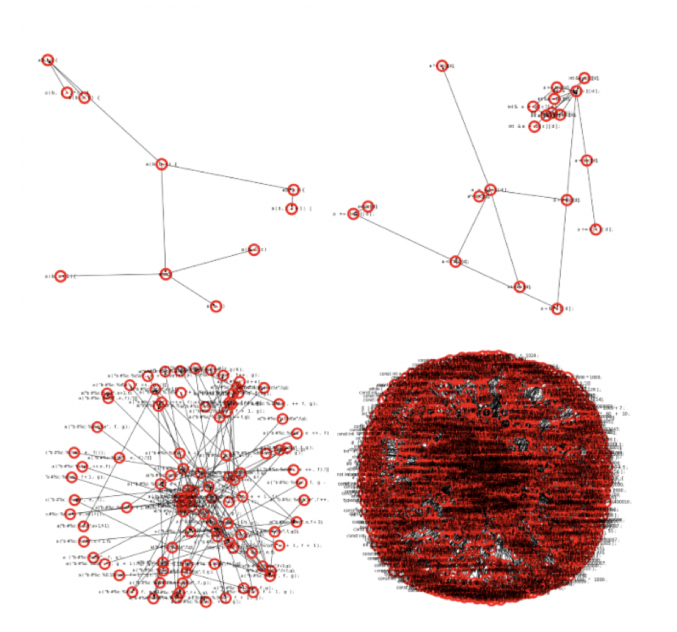

Fig. 5. Paraphrases graphed to show related phrases and how you can get from one to another. Here we are showing graphs with 10, 20, 101, and 1,435 nodes respectively.

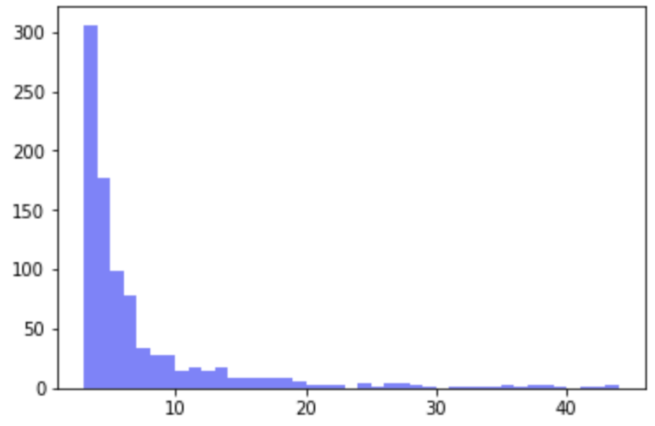

Fig. 6. The x-axis is the number of times a paraphrase is seen; the y-axis shows the number of phrases fall into this category (cardinally).

also shows a fairly balanced between true positive and false positives.

VIII. Discussion

These code transformations have the potential to improve code in various ways such as formatting, refactoring, and run-time execution optimization based on evaluation of the transformation. Formatting, for example, is as simple as "we like lines of code in form A instead of form B" so when we see phrase B, replace with phrase A. We can extend this idea to refactoring if we consider A to be better than B by some refactoring or optimization metric. These transformations can also be evaluated for stylometric purposes. For example, is either a common or uncommon phrase being used? Perhaps the author would like to use one or the other because they perceive it to be the best. Or it is also possible that an author always uses one way, but in this particular instance, he or she does not want people to know that they wrote this code.

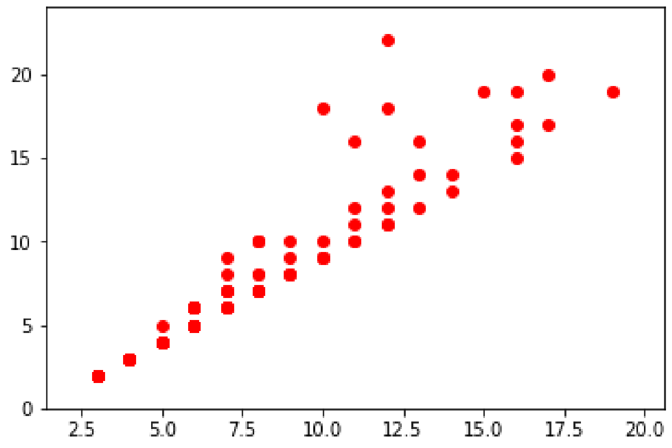

Fig. 7. The density of the graphs increases as the number of nodes increases

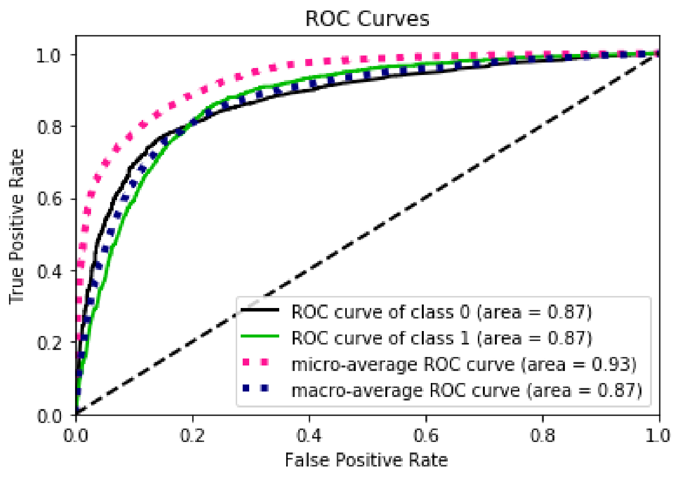

Fig. 8. ROC Curve

String	Close Parentheses
TernaryOpen	Curly Brace
Float Literal	Comparator
Open Parentheses	Whitespace
Close Bracket	Comment
Assignment Operator	String Literal
Operation	Annotation
Bit Operation	Open Bracket
Preprocessor Statement	Close Curly Brace
Keyword	Assignment
Integer literal	Punctuation

TABLE I
C++ Token Types

In this case, a recommender system could suggest ways that other people use or alternatively just uncommon methods of writing to obfuscate the code. The benefit of this method is that it is amenable to automation and does so quickly (at least faster and cheaper than source code transformations generated manually by a human).

This work opens many avenues of possible interest. The path directly forward suggests work on applying this method to high level paraphrases such as statements, functions, etc. The transformations can be used to impact FLP stylometry and is particularly well-suited for GANS. Both the refactoring and optimizing techniques use transformations that are commonly found in source code developed by humans rather than code that is generated automatically. Finally, they may help illuminate complicated or obfuscated code in the question/answering way that NLP uses these paraphrases.

IX. Conclusion

This work demonstrates that paraphrase techniques from Natural Language Processing are also suitable for formal languages and that the paraphrases identified can be used as code transformations with similar accuracy as NLP techniques. The Random forest model we trained can accurately discern valid source code transformations from invalid ones with accuracy much higher than comparable methods in NLP and without the need of a test suite and thus suitable for "production quality" code. To generate these automatically is desirable for applications such as Stylometry and other systematic approaches to changing source code.

Acknowledgment

The authors would like to thank the researches of PSAL, Anne Cocos, and Matt Burlick. This research was partially funded by the Auerbach Berger Chair in Cybersecurity held by Spiros Mancoridis, the Army Research Laboratory, and DARPA.

References

[1] C. Quirk, C. Brockett, and W. Dolan, "Monolingual machine translation for paraphrase generation," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 142–149.

[2] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[5] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 746–751.

[6] B. Dolan, C. Quirk, and C. Brockett, "Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 350.

[7] R. Shetty, B. Schiele, and M. Fritz, "A4nt: author attribute anonymity by adversarial training of neural machine translation," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1633–1650.

[8] S. N. Deore and M. N. Sawant, "Formal and natural languages: The difference analysis," vol. 7, no. 5, 2018, pp. 135–138.

[9] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-scale and language-oblivious code authorship identification," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 101–114.

[10] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing programmers via code stylometry," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 255–270.

[11] E. Quiring, A. Maier, and K. Rieck, "Misleading authorship attribution of source code using adversarial learning," *arXiv preprint arXiv:1905.12386*, 2019.

[12] R. Barzilay and K. R. McKeown, "Extracting paraphrases from a parallel corpus," in *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*, 2001, pp. 50–57.

[13] A. Ibrahim, B. Katz, and J. Lin, "Extracting structural paraphrases from aligned monolingual corpora," in *Proceedings of the second international workshop on Paraphrasing-Volume 16*. Association for Computational Linguistics, 2003, pp. 57–64.

[14] B. Pang, K. Knight, and D. Marcu, "Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003, pp. 102–109.

[15] "Code jam," accessed: 2019-05-01. [Online]. Available: https://codingcompetitions.withgoogle.com/codejam