Improving Performance of Automatic Duplicate Bug Reports Detection using Longest Common Sequence

Introducing New Textual Features for Textual Similarity Detection

Behzad Soleimani Neysiani
Department of Software Engineering,
Faculty of Computer & Electrical Engineering,
University of Kashan,
Kashan, Esfahan, Iran,
B.Soleimani@grad.kashanu.ac.ir
Tel Number: +98-913-960-6471

Seyed Morteza Babamir
Department of Software Engineering,
Faculty of Computer & Electrical Engineering,
University of Kashan,
Kashan, Esfahan, Iran,
Babamir@kashanu.ac.ir

***Abstract*— automatic duplicate bug reports detection is a famous problem in mining software repositories since 2004 for software triage systems e.g. Bugzilla. Textual features are the most important type of features in similarity and duplicate detection e.g. BM25F which indicate the common term frequency in two reports. Sometimes a common sequence can show more similarity in two texts, thus new features based on longest common sequence (LCS) of two bug reports proposed in this paper as new textual features for text similarity detection. Android, Eclipse, Mozilla, and Open Office dataset are used for evaluation of proposed features and the experimental results show LCS-based features are important and the accuracy, precision and recall of classifier prediction models improved 4.5, 2.5 and 2.5 percent respectively on average after using LCS and get up to 96, 98 and 97 percent respectively on average using different classifiers.**

Keywords— Triage System, Bug Reports, Duplicate, Automatic, Detection, Text Mining, Natural Language Processing, Information Retrieval, Longest Common Sequence

I. INTRODUCTION

Triage systems are very important software repositories in popular open source software like Open Office, Mozilla Firefox browser, Android operating system and Eclipse integrated development environment for organizing user bug reports. Automatic duplicate bug reports detection are very important to reduce the triagers' effort [1-3]. There exist about 10 to 60 percent duplicate bug reports which are time-consuming to find duplicates and handling the reports thereof [4].

Every bug report contains different fields or data like identical, categorical, textual, temporal, and attachment file. Duplicate detection needs similarity metrics between bug reports which can find more similar bug reports that can be duplicated. These similarity metrics usually named features which are calculated by comparison of an aspect in two bug reports. These features can be categorized through as followings [5]:

1. Identical and temporal features: these features are calculated by subtraction of the values of the same field of two reports. These features indicate the temporal relation of two reports which the lesser the value, the higher the probability of similarity of two reports.
2. Categorical features: these features computed by comparing the categorical fields of two bug reports where the equality of two nominal value of Eq. (1), (2) or (3), or the difference of two ordinal or interval values of Eq. (4) or (5) are calculated in two bug report d and q.

$$feature_1(d,q) = \begin{cases} 1 & if\ d.Product = q.Product \\ 0 & otherwise \end{cases} \quad (1)$$

$$feature_2(d,q) = \begin{cases} 1 & if\ d.Company = q.Company \\ 0 & otherwise \end{cases} \quad (2)$$

$$feature_3(d,q) = \begin{cases} 1 & if\ d.Type = q.Type \\ 0 & otherwise \end{cases} \quad (3)$$

$$feature_4(d,q) = \frac{1}{1 + |d.Priority - q.Priority|} \quad (4)$$

$$feature_5(d,q) = \frac{1}{1 + |d.Version - q.Version|} \quad (5)$$

3. Textual features: these features are extracted through Natural Language Processing and Information Retrieval techniques from textual fields. These fields need some pre-processing like the removal of frequent and less important words named stop words, tokenizing sentence to extract words and stemming words to find the pure form of each word. This provides the possibility to count the same words in two bug reports, thus, a textual similarity. There exists a concept named *n-gram* indicating that it is better to compare *n-sequence-word* of two reports together, e.g. 2-gram considers and find 2 sequential words of a text in another. Here, every time n is increased in *n-gram* checking, similarity increases more between two texts because they do not reveal only some vocabulary similarity but some

978-1-7281-0872-8/19/$31.00 ©2019 IEEE

conceptual similarity. Term Frequency (TF) and Inverse Document Frequency (IDF) equations are renowned Information Retrieval techniques for textual similarity calculation [6]. An occurrence of a term *t* in document *d* which can be a textual field of a bug report in triage systems is checked through Eq. (6). Parameter *K* is the number of textual fields in document *d* and *f* is an index through textual fields of a bug report with each text field of w_f importance weight and *length* is the number of characters in term *t* and $average\_length_f$ is the average length of all words in this field. The importance of a term t in the whole bug reports as a document *D* which contains many *d* documents and each document contains many *t* terms are involved in Eq. (7). BM25F is an aggregation value as a weighted average of TF-IDF approaches for all common terms in both texts *d* and *q* and K1 is a constant for preventing division by zero in Eq. (8).

$$TF_D(d,t) = \sum_{f=1}^{K} \frac{w_f \times occurrences(d[f],t)}{1 - b_f + \frac{b_f \times length}{average\_length_f}} \quad (6)$$

$$IDF(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \quad (7)$$

$$BM25F_{ext}(d,q) = \sum_{t \in d \cap q} IDF(t) \times \frac{TF_D(d,t)}{K_1 + TF_D(d,t)} \quad (8)$$

4. Contextual features: these features compare textual fields of a bug report with a word list with a special content e.g. security or performance of the software. The result obtained from these textual features indicates how much does this report related to special context, thus, the conceptual category for bug reports. Contextual features of two bug reports can be compared as a vector by cosine similarity equation.

New features can be very important for duplicate detection and every new feature can improve the performance of similarity detection especially text-based features. So, this paper focuses on using the longest common sequence (*LCS*) as a new feature for textual feature extraction. This article is organized as follows: The literature review is presented in section II where explains the state of the arts in duplicate bug report detection approaches and their features; The methodology of this research for duplicate bug report detection and LCS features extraction described in section III; The experimental results of the proposed method in bug reports represented in section IV and the article is concluded in section V.

II. Literature Review

There are many related works which used different type of features described in section I. *Table I.* shows the kind of features used in the state of the art in duplicate bug report detection over recent decade, which numbered columns refer to feature categories in section I, in other words, 1, 2, 3 and 4 are equal to identical and temporal, categorical, textual, and contextual features. Every research uses some features of these categories and some of them introduce new features. Some researchers focus on feature extraction and some of them focus on the methodology of information retrieval and duplicate detection process by classifiers or heuristic approaches. It is important to consider that the length of LCS used the first time for similarity detection before [7], but this research introduces two other new features based on LCS approach and also, consider the different type of features together for performance improvement.

TABLE I. Related works that used different types of features.

Reference	Year	1	2	3	4
Čubranić and Murphy [8]	2004	*			
Hiew [9]	2006		*		
Runeson, et al. [10]	2007		*		
Jalbert and Weimer [11]	2008	*	*		
Bettenburg, et al. [12]	2008	*	*		
Wang, et al. [13]	2008		*		
Nagwani and Singh [14]	2009		*		
Sureka and Jalote [15]	2010		*		
Sun, et al. [16]	2010		*		
Sun, et al. [6]	2011	*	*		
Nguyen, et al. [17]	2012		*	*	
Banerjee, et al. [7]	2012		*		
Alipour, et al. [18]	2013	*	*	*	
Lazar, et al. [19]	2014	*	*		*
Wang, et al. [20]	2014		*		
Aggarwal, et al. [21]	2015	*	*	*	
Sharma and Sharma [22]	2015	*	*	*	
Zou, et al. [23]	2016		*	*	
Yang, et al. [24]	2016	*	*		
Rakha, et al. [2]	2017		*		
Budhiraja, et al. [25]	2018		*	*	
Su and Joshi [26]	2018	*			
Bommaraju, et al. [27]	2018			*	
Hindle and Onuczko [28]	2018			*	

III. Proposed Method

The methodology of duplicate bug report detection is shown in *Fig. 1*. The first step is preprocessing bug report and made it ready for feature extraction, especially textual fields of the bug report like title and description. In this step, null values should be considered which can be predicted for numerical values or eliminate the report with textual null fields. Also, stop words (less important words) are removed from textual fields of bug reports, then the text will be tokenized –separate terms- and finally it can be stemmed –the stem of each term recognized-. Then features extraction phase will calculate the different type of features. There is a specific topic as n-gram in textual features which refer every textual feature extracted for *n* connected

sequential terms e.g. bi-gram use 2 connected terms. After feature extraction, a predictor or classifier model should be made for duplicate detection. Many classifiers like Decision Tree, Naive Bayes, Neural Networks e.g. Perceptron, Regression methods e.g. Linear or Logistic, even ensemble approaches can be used for duplicate detection. Then the performance (accuracy, precision, recall) of this made model should be evaluated which cross-validation process can be used for this purpose.

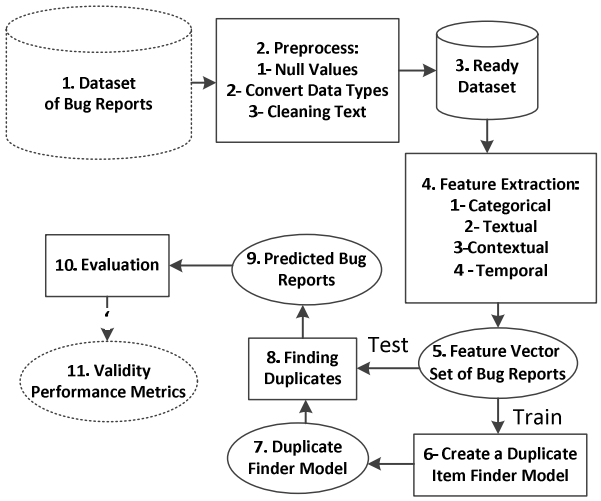

Fig. 1. The methodology of duplicate bug report detection

In this research dynamic LCS algorithm [29] used to extract a most similar substring of the description of two bug reports as described in Fig. 2. This algorithm gets two string X and Y indexed from 1 to m and n respectively. Then made a zero $m \times n$ matrix to count the longest similar sequence of two strings e.g. if X be 'hello my dear John. How are you?' and Y be 'hi dear Johnathan. are you there?', then the LCS will be 'h dear John. are you?' which these characters are a common sequence in both texts and the length of LCS will be 21. Also, other features can be extracted from this LCS text like the number of words in LCS and average length of words in LCS. So, 3 LCS features can be extracted as textual features for duplicate bug report detection. The time complexity of LCS algorithm is O ($m \times n$) but there are other version of the LCS algorithm with lower time complexity too [30]. Also, it is important that the terms of sentences have no typo, so, it is necessary to find the typos in sentences, and then correct them. There are many typos in bug reports [31], especially interconnected terms which can be modified before LCS calculation [32].

```
Algorithm: Longest Common Sequence Length
Input: X[1..m] and Y[1..n] as String
Output: The Longest Common Sequence Length
Let C = Array[0..m , 0..n]
for i := 0..m
    C[i,0] = 0
for j := 0..n
    C[0,j] = 0
for i := 1..m
    for j := 1..n
        if X[i] = Y[j]
            C[i,j] := C[i-1,j-1] + 1
        else
            C[i,j] := max(C[i,j-1], C[i-1,j])
return C[m,n]
```

Fig. 2. Longest Common Sequence Feature Calculation Algorithm

IV. Evaluation Method

The experimental evaluation method based on the methodology of Fig. 1 implemented by Java programming language for feature extraction based on sate of the arts features [33] and the LCS algorithm implemented in Java to extract new LCS-based features. Then Rapid Miner tool used for duplicates bug report detection model creation and evaluating the performance of new features versus state of the art features. Android, Mozilla, Eclipse and Open Office bug report dataset [34] are used for the evaluation process. Every dataset clustered based on merge id of every bug report in classes of bug reports which duplicate bug reports will be in the same cluster or group and then pairs of bug reports selected from these clusters. If both bug reports were in the cluster, then the selected pair will be labeled as duplicate otherwise as non-duplicate. These features extracted from bug reports based on the state of the arts:

- Bug Id difference as an Identical feature
- Product, Company, Type, Priority and Version as Categorical features
- BM25F of uni-gram and bi-gram terms of title + description of bug reports as textual features
- Cosine Similarity of General, Java, Networking and Cryptography word lists as contextual features

Even though there are some bug reports with null value for the title or description, but there is no bug report with null value for both title and description [31], so the title and description fields considered together as a text field for some textual features like LCS which don't need them separately. Then LCS-based features extracted from bug reports and appended to the state of the arts features and both feature collections used in 10-fold cross-validation process. The duplicate detector model predict the duplication or non-duplication of a pair of bug reports which the result based on the actual state can be true or false and four

modes of Table II will be occur which the first row show the duplication state in the real world based on bug report clusters (the actual value of duplication state) and first column show the predicted result of duplicate detector model (the predicate value).

TABLE II. MODES OF DUPLICATE DETECTOR

Actual → / Predict ↓	Dup (D)	Non-Dup (ND)	Total
True	True Dup (TD)	False Dup (FD)	True Prediction (TP=TD+TND)
False	False Non-Dup (FND)	True Non-Dup (TND)	False Prediction (FP=FD+FND)
Total	Duplicates (D=TD+FND)	Non-Duplicates (ND=FD+TND)	Total (TT = TP+FP=D+ND)

Then some experiments designed based on variables of Table II, which contains 80 experiments based on independent variables. Also, the definition of dependent variables based on Table II is described in Table III and the control variables of experiments specified based on the dataset, especially the number of pairs which less researchers mention that in their papers and is very important for comparing duplicate detector's performance. This research considers all possible duplicate pairs of bug report clusters and also, consider 90 times non-duplicate pairs randomly, in other words 10 percent of pairs are duplicate and these are the most pairs based on the state of the arts.

TABLE III. VARIABLES OF EXPERIMENTS

Variable Type	Variable Name	Variable States (Values)
Independent	Dataset	Android, Mozilla, Eclipse, Open Office
	Stemming	No, Yes
	Classifier	Naïve Bayes (NB), Decision Tree (DT), Linear Regression (LR), Perceptron Neural Network (P), Bayesian Boosting (BB) by Decision Tree
	Features	State of the Art Features (SotAFs), Longest Common Sequence Features (LCSFs)
Control	Number of Bug Reports	**Dataset** — **Bug Reports**: **Android** 37,626; **Eclipse** 45,234; **Mozilla** 75,648; **Open Office** 31,135
	Number of Bug Pairs	**Dataset** — **Pairs**: **Android** 209,260; **Eclipse** 55,361; **Mozilla** 142,971; **Open Office** 66,151
Dependent	Accuracy	$\text{Accuracy} = \frac{True\ (T)}{Total\ (TT)}$
	Precision	$\text{Precision} = \frac{True\ Duplicates\ (TD)}{Duplicates\ (D)}$
	Recall	$\text{Recall} = \frac{TD}{TD + FND}$

The average accuracy, precision and recall rate in all experiments shown in Fig. 3 in percent, which demonstrate 4.95, 2.59 and 2.72 percent relative improvement respectively, for LCS Features (LCSFs) versus State of the Arts Features (SotAFs) and indicate that LCS features improve the performance of duplicate detection. Also, the standard error (standard deviation from average divide by number of samples) is shown in all charts.

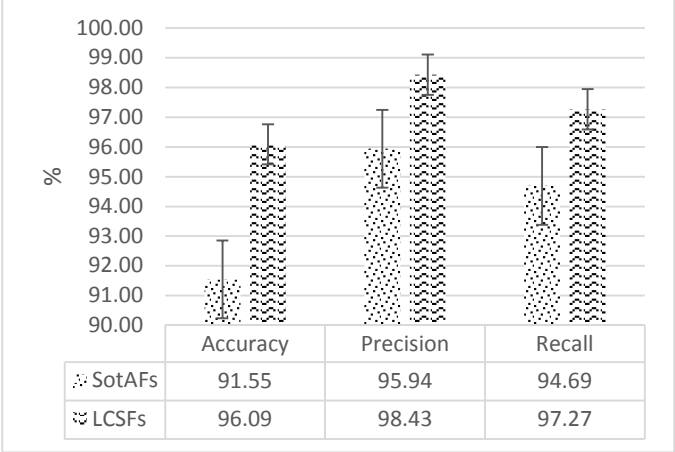

Fig. 3. Performance of LCS features versus state of the arts features

Fig. 4 show the accuracy of LCS features versus the SotAFs for various datasets which indicate 6.4, 3.9, 3.5 and 6.2 percent relative improvement for Android, Eclipse, Mozilla and Open Office dataset respectively and 5 percent improvement averagely.

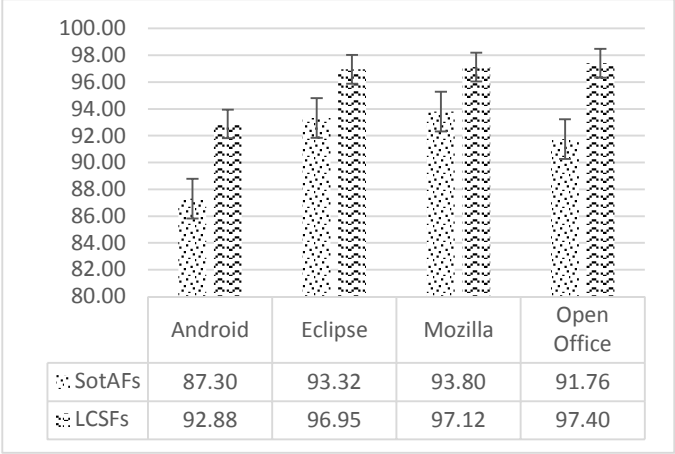

Fig. 4. Accuracy of LCS versus state of the arts features in various datasets

Fig. 5 show the accuracy of LCS features versus the SotAFs for various duplicate detector (classifiers) which indicate 1.1, 1.1, 8.8, 12.0 and 2.6 percent relative improvement for Bayesian Boosting, Decision Tree, Linear Regression, Naïve Bayes and Perceptron classifiers respectively and 5 percent improvement averagely. There is a question in this figure that why Android accuracy is lower than others? It seems the Android dataset have some unlabeled bug reports as duplicate or the context of bug reports are very similar but different. It should be checked accurately in future researches.

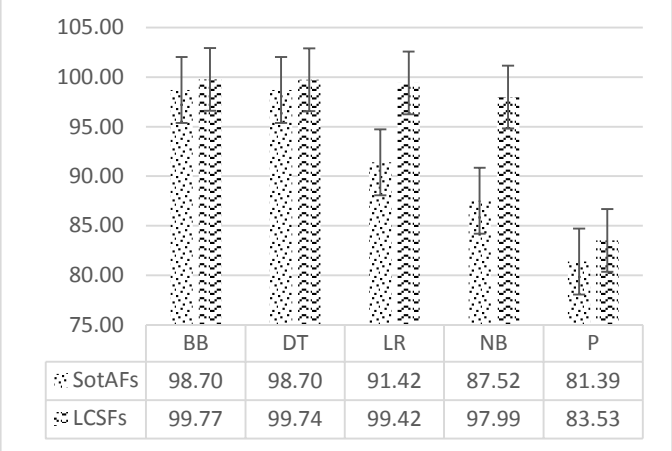

Fig. 5. Accuracy of LCS versus state of the arts features in various classifiers

Fig. 6 show the accuracy of LCS features versus the SotAFs for various stemming condition -with and without (Raw format) stemming- which indicate 5.41 and 4.4 percent relative improvement for raw and stemmed mode respectively and 4.9 percent improvement averagely. Also, it is interesting that stemming shows 3.55 % relative accuracy improvement versus raw format in the state of the arts, but 2.6 % relative accuracy improvement in LCS mode; so, it shows low improvement in LCS mode and it can be due to LCS nature that consider a kind of stemming in itself and can detect partial common sequence in two texts, so if a word has different form, LCS can detect the similarity of both forms without stemming and it is another goodness of LCS.

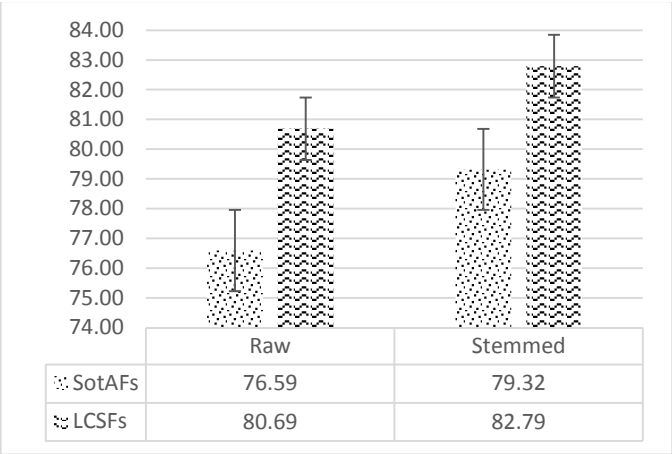

Fig. 6. Accuracy of LCS features versus state of the arts features

V. Conclusion

In this paper Longest Common Sequence (LCS) used for feature extraction as a new textual feature to improve the performance of similarity detection of bug reports in software triage systems. The experimental results show the proposed features can improve the performance of duplicate bug report detection greatly. It should be considered new features can be introduced and used too, but another question is which feature is more important in duplicate bug report detector because the feature vector space is growing and it is very important to reduce the feature space for runtime performance issues especially in real-time usages.

Acknowledgment

It is our duty to appreciate Ms. Razieh Hadian for her kindness and help to implement the LCS method in the Java programming language and running some experiments. Also, thanks a lot for those who strong us by their objections.

References

[1] K. Koochekian Sabor, A. Hamou-Lhadj, and A. Larsson, "DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Prague, Czech Republic, 2017, pp. 240-250.

[2] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan, "Revisiting the Performance Evaluation of Automated Approaches for the Retrieval of Duplicate Issue Reports," *IEEE Transactions on Software Engineering,* 2017.

[3] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Software Engineering,* vol. 21, pp. 368-410, April 01 2016.

[4] Y. C. Cavalcanti, P. A. d. M. S. Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, "The bug report duplication problem: an exploratory study," *Software Quality Journal,* vol. 21, pp. 39-66, 2013.

[5] B. Soleimani Neysiani and S. M. Babamir, "Methods of Feature Extraction for Detecting the Duplicate Bug Reports in Software Triage Systems," presented at the International Conference on Information Technology, Communications and Telecommunications (IRICT), Tehran, Iran, 2016.

[6] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2011, pp. 253-262.

[7] S. Banerjee, B. Cukic, and D. Adjeroh, "Automated duplicate bug report classification using subsequence matching," in *IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE)*, 2012, pp. 74-81.

[8] D. Čubranić and G. C. Murphy, "Automatic bug triage using text categorization," in SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, 2004.

[9] L. Hiew, "Assisted detection of duplicate bug reports," Master of Science, Faculty of Graduate Studies(Computer Science), The University Of British Columbia, 2006.

[10] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE)*, 2007, pp. 499-510.

[11] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *IEEE International Conference on Dependable Systems and Networks (DSN) With FTCS and DCC*, 2008, pp. 52-61.

[12] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," in *Proceedings of the 2008 international working conference on Mining software repositories*, Leipzig, Germany, 2008, pp. 27-30.

[13] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, Leipzig, Germany, 2008, pp. 461-470.

[14] N. K. Nagwani and P. Singh, "Weight similarity measurement model based, object oriented approach for bug databases mining to detect similar and duplicate bugs," in *Proceedings of the International Conference on Advances in Computing, Communication and Control*, 2009, pp. 202-207.
[15] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *17th Asia Pacific Software Engineering Conference (APSEC)*, 2010, pp. 366-374.
[16] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 45-54.
[17] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2012, pp. 70-79.
[18] A. Alipour, A. Hindle, and E. Stroulia, "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, San Francisco, CA, USA, 2013, pp. 183-192.
[19] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories*, Hyderabad, India, 2014, pp. 308-311.
[20] S. Wang, F. Khomh, and Y. Zou, "Improving bug management using correlations in crash reports," *Empirical Software Engineering, Springer,* pp. 1-31, 2014.
[21] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," in *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Montreal, QC 2015, pp. 211-220.
[22] A. Sharma and S. Sharma, "Bug Report Triaging Using Textual, Categorical and Contextual Features Using Latent Dirichlet Allocation," *International Journal for Innovative Research in Science and Technology (IJIRST),* vol. 1, pp. 85-96, Feb 2015 2015.
[23] J. Zou, L. Xu, M. Yang, X. Zhang, J. Zeng, and S. Hirokawa, "Automated Duplicate Bug Report Detection Using Multi-Factor Analysis," *IEICE Transactions on Information and Systems,* vol. E99.D, pp. 1762-1775, 2016.
[24] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining word embedding with information retrieval to recommend similar bug reports," in *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 127-137.
[25] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "DWEN: deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, 2018, pp. 193-194.
[26] E. Su and S. Joshi, "Leveraging product relationships to generate candidate bugs for duplicate bug prediction," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, 2018, pp. 210-211.
[27] S. P. Bommaraju, A. Pasala, and S. Rao, "System and method for detection of duplicate bug reports," ed: Google Patents, 2018.
[28] A. Hindle and C. Onuczko, "Preventing duplicate bug reports by continuously querying bug reports," *Empirical Software Engineering,* pp. 1-35, 2018.
[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*: MIT press, 2009.
[30] K.-T. Tseng, D.-S. Chan, C.-B. Yang, and S.-F. Lo, "Efficient merged longest common subsequence algorithms for similar sequences," *Theoretical Computer Science,* vol. 708, pp. 75-90, 2018.
[31] B. Soleimani Neysiani and S. M. Babamir, "Automatic Typos Detection in Bug Reports," presented at the IEEE 12th International Conference Application of Information and Communication Technologies, Kazakhstan, 2018.
[32] B. Soleimani Neysiani and S. M. Babamir, "Automatic Interconnected Lexical Typo Correction in Bug Reports of Software Triage Systems," presented at the The International Conference on Contemporary Issues in Data Science, Zanjan, Iran, 2019.
[33] A. Alipour, "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection," Master of Science, Department of Computing Science, University of Alberta, Faculty of Graduate Studies and Research, 2013.
[34] A. Alipour, A. Hindle, T. Rutgers, R. Dawson, F. Timbers, and K. Aggarwal. (2013). *Bug Reports Dataset*. Available: https://github.com/kaggarwal/Dedup