An Approach to Detecting Duplicate Bug Reports using N-gram Features and Cluster Chrinkage Technique

Phuc Nhan Minh*

[*] Faculty of Engineering and Technology, Tra Vinh University, Viet Nam

Abstract- Duplicate bug report describes problems for which there is already a report in a bug repository. For many open source projects, the number of duplicate reports represents a significant percentage of the repository, so automatic identification of duplicate reports are very important and need let's avoid wasting time a triager spends in searching for duplicate bug reports of any incoming report. In this paper we want to present a novel approach which it can help better of duplicate bug report identification. The proposed approach has two novel features: firstly, use n-gram features for the task of duplicate bug report detection. Secondly, apply cluster shrinkage technique to improve the detection performance. We tested our approach on three popular open source projects: Apache, Argo UML, and SVN. We have also conducted empirical studies. The experimental results show that the proposed scheme can effectively improve the detection performance compared with previous methods.

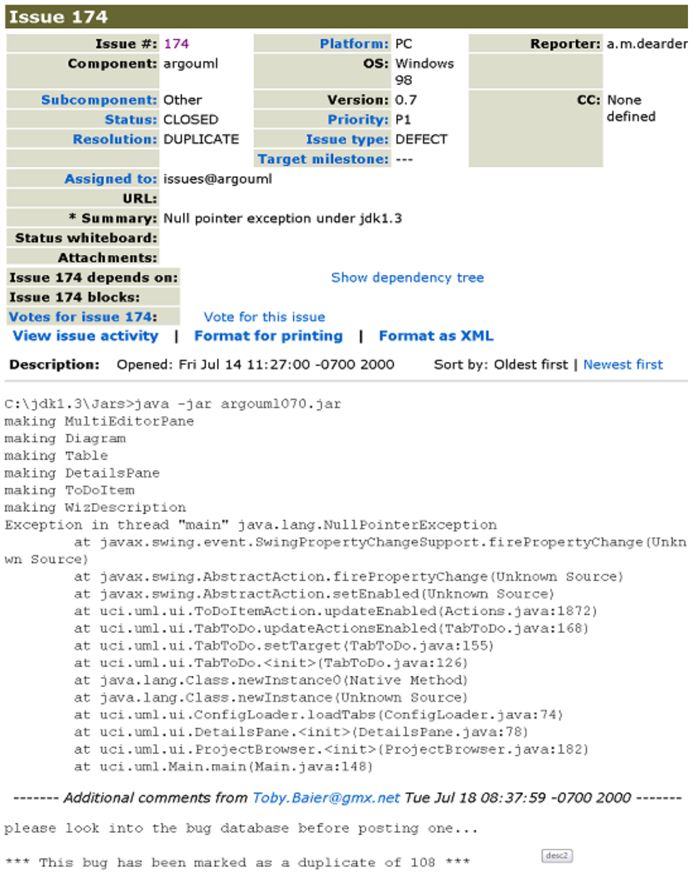
Index Terms- Bug Reports, Duplicate Bug Detection, N-gram feature, Bug Report Analysis, Cluster Shrinkage

I. INTRODUCTION

In software maintenance, discovering software abnormal behavior is an important process to avoid serious damages. Typically, these abnormal situations are collectively described in bug reports which are submitted to a bug report management system (BRMS) such as Bugzilla, Eclipse so on for further handling. After the bug reports are submitted, one or more triagers will be assigned to analyze them and then pass them to the appropriate programmers for bug fixing. As reported in recent literature [1, 4, 5, 6], an important concern of duplicate detection on bug reports in bug report handling has been addressed. The main reason behind this concern is that the percentage of duplicate bug reports can be significantly up to 36% [1, 2, 5, 6], for example, the Eclipse dataset collected from October 2001 to August 2005 has 18,165 bug reports, and 20% of the reports are duplicate [1].The same paper also reports that 30% of 2,013 reports in the Firefox dataset collected from May 2003 to August 2005 are duplicate [1]. In 2006, Hiew reported that the duplicate reports in the Firefox repository could even reach 36% [4]. The significant number of the duplicate bug reports shows the importance of how to properly process them in the debugging and testing work. So, identifying duplicate bug reports play very important role. First, the analysis time for triaging bug reports can be highly reduced. Second, the information contained in duplicate bug reports may be very helpful in the debugging process because they provide more information on software abnormal operations that was not described in the first bug report (the master bug report).

II. DUPLICATION DETECTION PROBLEM

The duplication detection problem in explained as follows: As stated in [3, 8], the problem of duplication detection can be characterized by identifying two or more bug reports that describe the same software fault. In [5], the duplicates can be further classified into two classes: (1) the duplicate bug reports describe the same failure situation, and (2) the bug reports describe the different failures with the same source of the software fault. Since the second duplication type may involve different vocabulary for different bug reports, its detection cannot be effective by only exploring the textual information of the bug reports [5, 7]. In order to detect effectively the second type duplication requires program-specific information, such as execution traces in [7]. However, this may raise the privacy problem. Therefore, this research only focuses on detecting the first type of duplication which only considers the textual information of bug reports.



Issue 174

Issue #:	174	**Platform:**	PC	**Reporter:**	a.m.dearder
Component:	argouml	**OS:**	Windows 98		
Subcomponent:	Other	**Version:**	0.7	**CC:**	None defined
Status:	CLOSED	**Priority:**	P1		
Resolution:	DUPLICATE	**Issue type:**	DEFECT		
		Target milestone:	---		
Assigned to:	issues@argouml				
URL:					
*** Summary:**	Null pointer exception under jdk1.3				
Status whiteboard:					
Attachments:					

Issue 174 depends on: Show dependency tree
Issue 174 blocks:
Votes for issue 174: Vote for this issue
View issue activity | **Format for printing** | **Format as XML**

Description: Opened: Fri Jul 14 11:27:00 -0700 2000 Sort by: Oldest first | Newest first

```
C:\jdk1.3\Jars>java -jar argouml070.jar
making MultiEditorPane
making Diagram
making Table
making DetailsPane
making ToDoItem
making WizDescription
Exception in thread "main" java.lang.NullPointerException
        at javax.swing.event.SwingPropertyChangeSupport.firePropertyChange(Unkn
wn Source)
        at javax.swing.AbstractAction.firePropertyChange(Unknown Source)
        at javax.swing.AbstractAction.setEnabled(Unknown Source)
        at uci.uml.ui.ToDoItemAction.updateEnabled(Actions.java:1872)
        at uci.uml.ui.TabToDo.updateActionsEnabled(TabToDo.java:168)
        at uci.uml.ui.TabToDo.setTarget(TabToDo.java:155)
        at uci.uml.ui.TabToDo.<init>(TabToDo.java:126)
        at java.lang.Class.newInstance0(Native Method)
        at java.lang.Class.newInstance(Unknown Source)
        at uci.uml.ui.ConfigLoader.loadTabs(ConfigLoader.java:74)
        at uci.uml.ui.DetailsPane.<init>(DetailsPane.java:78)
        at uci.uml.ui.ProjectBrowser.<init>(ProjectBrowser.java:182)
        at uci.uml.Main.main(Main.java:148)
```

------- *Additional comments from Toby.Baier@gmx.net Tue Jul 18 08:37:59 -0700 2000* -------

```
please look into the bug database before posting one...

*** This bug has been marked as a duplicate of 108 ***
```

Figure 1: An example of a duplicate bug report in ArgoUML project

To detect the duplicate bug reports, the textual information contained in the bug reports must be first passed and extracted. A bug report generally consists of some Meta- information, a short summary, textual description, and some error messages. It may also have some comments from other reporters. Figure 1 is an example for the ArgoUML project. This bug report was submitted to a bug tracking system Tigris by A. M. Dearden. In the Description field, A. M. Dearden provided abundant information about an exception in ArgoUML execution. If the bug report is the original report for the software failure, it is called the master bug report (MBR). Otherwise, it will be labeled as duplicate after the triaging process. In Figure 1, this bug report # 174 is identified as a duplicate in the Resolution field. On the bottom of the figure, it can be seen that this bug report is a duplicate of bug report # 108. In this case, these two bug reports belong to the same report cluster (RC) in this paper.

The duplication detection problem in this research is processed as follows. For a software project, the historical bug reports are first classified into *n* report clusters (RCs). Each RC has a master bug report (MBR). If an RC has more than one bug report, the bug reports in the RC have the duplicate relationships. For each incoming bug report BR_x, the duplication detection is performed to generate a recommendation list that shows the likelihood order of BR_x being a member of RC_i. The duplication relationship is determined if one of the following conditions is satisfied:

1. For a master report BR_m, a bug report BR_i has been resolved as duplicate with a reference to BR_m in the bug tracking system, and the report status is closed.

2. For two bug reports BR_i and BR_j, if they are marked as the duplicate of BR_m, BR_i is a duplicate of BR_j, and vice versa.

3. If there is another bug report BR_k that is marked as duplicate of BR_i, BR_k is also a duplicate of BR_m. This property is called the transitivity.

The rest of the paper is organized into four sections. Section III gives a brief overview of related work. Section IV presents the proposed scheme in which n-gram features and cluster shrinkage are used to improve the performance of duplicate detection. In section V, the empirical study on different open-source projects is elaborated to demonstrate the effectiveness of the proposed scheme. Finally, section VI concludes the paper.

III. RELATED WORK

This section presents the previous studies on the n-gram approach and the cluster shrinkage technique.

In 2006, Hiew proposed an incremental clustering model using natural language processing (NLP) techniques to identify duplicate bug reports in his master thesis [4]. According to his report, the detection recall rate can achieve 20%-50% in four software projects when the recommendation list has 7 items (Eclipse: 20%, Fedora: 31%, Apache: 32%, and Firefox: 50%).

In 2007, a study similar to Hiew´s work for Sony Ericsson software projects was reported in [5]. Compared with Hiew´s work the scheme proposed by Runeson, Alexandersson, and Nyholm show that using only basic NLP techniques can achieve comparable performance. In 2008, Jalbert and Weimer proposed a detection scheme in which they used a specific feature weighting equation and a graph cluster algorithm to improve the detection performance [9]. However, their scheme only performs up to 1% better than the work of [5].

In 2010, Sureka and Jalote proposed an n-gram-based approach to detect duplicate bug reports [6]. However, the performance of their scheme still remains moderate from their experiments in which a 40.22% recall rate is achieved for the *top-10* recommendations. The indifferent performance is mainly because their scheme uses character-level n-gram-based features that may contain a lot of noisy information for similarity computation.

Based on the observations of the previous studies, we proposed a detection scheme based on the n-gram features and the cluster shrinkage technique. With the n-gram features, the proposed scheme effectively improves the classification power for duplication detection. With the assistance of cluster shrinkage, the divergence problem due to n-gram features is mitigated.

IV. DUPLICATION DETECTION APPROACHES

The proposed approaches use the NLP technique, N-gram, and the cluster Shrinkage technique. For a software project, the historical bug reports are classified into *n* report clusters (RCs). To form the clusters, we use the comments of bug reports to create a mapping file. In a cluster that has more than one bug report, we use the last bug report as the test data. In other words, the largest bug ID in each cluster is the incoming bug report. The other bug reports in the cluster are the historical data. In our observation on bug reports, we find they may have weak semantic similarity with other reports. The main reason is that the submitter may not described the bug in details just use different words to describe the same bug. In addition, there are many compound words. Therefore, n-gram is used to extract more information from these diversities. It can improve similarity identification between reports of the same RC. Then, we use the cluster shrinkage technique (CS) that can also improve the similarity identification by reweighting the features of bug reports. The proposed scheme has the following four basic processing steps:

1. Feature extraction
2. Feature reweighting
3. Similarity calculation
4. Recommendation generation

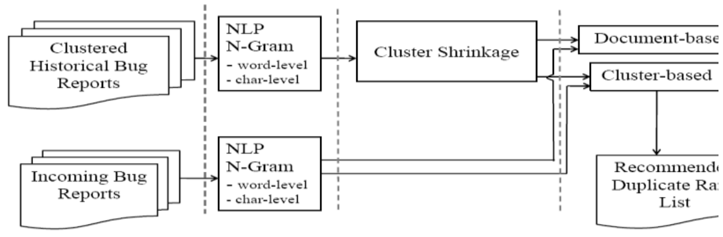


Figure 2: The processing flow

1. Feature extraction

1.1 Data Types

We have two types of bug reports historical bug reports and incoming bug reports. The incoming bug report is a set that include the largest bug report ID in all clusters. In other word, it is collects the last bug report that be submitted in each cluster. The historical bug report is already in our bug repository. For each incoming bug report, the historical bug report set is the bug report that bug ID is smaller than incoming bug report. We will design the approaches in historical bug reports to help us find the duplicate bug report.

1.2 Bug Report Cluster

The information that is marked as a duplicate in bug reports helps to create a mapping file to form the report clusters (RCs). An example mapping file is shown is in Table 1.1. The number in Table 1.1 is the bug report ID and the sequence is sorted by the bug report ID. The smallest of cluster size is 2. In other word, the smallest of cluster combine by one incoming bug report and one duplicate. In our case study to see the Table 1.2, we can find the most of cluster size between 2 and 4.

Table 1.1: Duplicate bug report mapping file

	Incoming Bug Report	Duplicate Bug Report
1	330,	329,
2	433,	387,
…	…	…
n	2610,	2520,2407,525,

Table 1.2: Cluster size

Cluster Size	2	3	4	5	6	7	8	9	10	…	Sum
SVN	103	24	6	1	1	0	0	0	0	…	135
ArgoUML	230	47	16	6	3	4	0	0	0	…	307
Apache	138	43	11	8	7	1	2	0	0	…	214

1.2 N-gram Feature Extraction

We use the vector space model to represent bug reports. In this step, we use NLP and n-gram techniques to help us build bug report vector. The Word Vector Tool is a Java library that can help us to calculate vectors. In WVTool tool, we construct a bug report with three parts. First, we use NLP that can help us to extract the tokens of word. Second, we use character-level n-gram that can help us to find the similarity between lexical words in detail that means is to find the common word between original word and his abbreviation. Third, we use word-level n-gram to find the sequence relationship between the words. It also can find the compound word. In section V, we will show the experiment results and discuss the parameter settings.

Table 1.3 is an example SVN bug report that ID is 330. Table 1.4 is the vector after our pre-processing approaches. We can get more information from different tokenization and improve the performance.

Table 1.3: SVN bug report context

SVN Bug Report 330
short description : śvn ci´ canť deal with mixed-revision working copies
short description: śvn ci´ canť deal with mixed-revision working copies
Long Description : WHO : sussman — BUG_WHEN : 2001-03-19 10:41:03 — Ben rewrote the fs commit editor so that replace_*() would call svn_fs_copy() ifthe base_rev argument was unexpected. This should result in a transaction beingbuilt that propely mirrors the mixed-revision working copy.However- weŕe now getting a conflict in merge() when we commit the transaction. Mike Pilato is looking into it.WHO : sussman — BUG_WHEN : 2001-03-19 10:48:59 — *** This bug has been marked as a duplicate of 329 ***

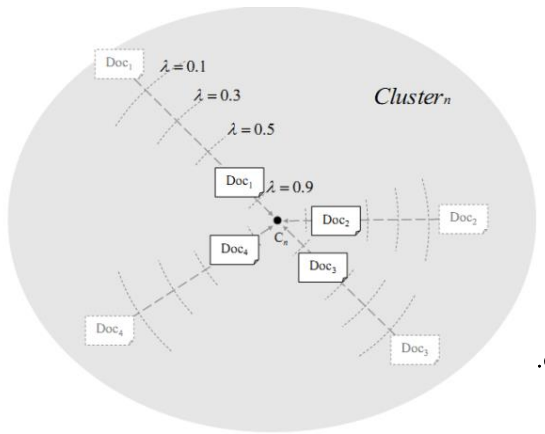
Table 1.4: SVN bug report item

	SVN Bug Report 330 Vector
NLP	result deal edit ...
Char-based N-Gram	mixed- ixed-r xed-re ...
Word-based N-Gram	mirrorsthemixed-revision mixedrevisionworking revisionworkingcopy ...

2. Feature reweighting

We use the cluster shrinkage to help us find the semantics of bug report overlap. In this way, it will increase the member of cluster relationship by the threshold. The first, we have to find a center of cluster. The second, we shrink all of bug report to its center.

1. Centroid of Clusters: the centroid, we use it to represent the cluster, is a center vector. Each cluster has a centroid that with all information in its cluster. We use the average vector that in a cluster to calculate centroid. Because the submitter does not always describe the bug in detail, it will make the similarity calculation inefficient. The reason is two duplicate bug reports with seldom same words and it will make to determine whether



they are duplicate bug report become difficult. So the centroid can help us increase similarity between duplicate bug reports, it have more words.

Figure 3: Documents in each cluster are moved toward the cluster centroid c in cluster shrinkage

2. Using Cluster Shrinkage: After we find the each centroid of cluster, we shrink all of bug report to its centroid of cluster. The symbol is a threshold. The symbol v represents a bug report vector. The symbol v' means the new vector.

```
For each cluster {
        N is the number of bug reports in S
                Compute its centroid:
                $C=\frac{1}{N}\sum^{N} v$
                For each bug report $v \in S$
                {
                        $v'=(1 - \lambda)\, v + \lambda c$
                        Where $0 \le \lambda \le 1$
                }
        }
```

3. Similarity Computation

We follow the past research result that using cosine can get the better performance. We have two similarity calculations. One of the similarity calculations is document-based ranking and another is cluster-based ranking. In the document-based ranking, we compare the incoming bug report with all bug report in bug repository and sort by similarity calculation value to determine duplicate bug report. There is a problem in document-based ranking. It is inevitable that different submitter use different word to describe the bug. Although we use the cluster shrinkage technique to resolve this problem, it cannot be completely avoided. For example, a cluster has three bug reports, two bug reports use the word "bug" to describe and one bug report uses the word defect. The word "defect" will become a noise and the bug report will has low similarity. So, we use cluster-based ranking. In the cluster-based ranking, we re-calculate the cosine value before determining duplicate bug report. We average the cosine of members of cluster. Then we compare the incoming bug report with all bug report with new cosine value in bug repository and sort by similarity calculation value to determine duplicate bug report. This way can resolve the seldom bug reports in his cluster has low similarity.

4. Top-N Recommendation

We present the result like as previous work [4]. Using the top-n recommendation system can help user to find the duplicate bug reports. We list rank from 1 to 22 to and observe the performance in every rank. Then, we compare the top-N recommendation with past researches. In our approaches, we get the better performance than others.

V. EXPERIMENTS

In this section, we introduce our experimental environment and the experiments. The open-source data sets are considered in our empirical study. We use ArgoUML, SVN, and Apache. We also implement past research work for performance comparison. In the work of Runeson et al. [5], we only use the cosine similarity because it has the best performance in their study.

1. Environment

In the experimental environment, we have three open-source projects. The bug reports of ArgoUML and SVN are respectively collected from Tigris.org. The bug reports of Apache are collected from Bugzilla.org. ArgoUML is a leading open-source UML modeling tool which supports all standard UML diagrams. The source code of the ArgoUML project is developed in Java and can run on any Java platform. Apache is an open-source http server project. SVN is an open-source software project that performs version control. More descriptive statistics of Argo UML, Apache, and SVN can be found in Table 1.5.

2. Experimental Setup

In our scheme, we have three parameters. The first n_c is the size of character-level n-gram (CN). The second n_w is the word-level n-gram length. Both n_c and n_w direct influence how many features are extracted. In our experiments, we find that n_c and n_w are 6 and 3 to have a slightly better performance among other values. The main reason is that on average the word length is between 5 and 6. The symbols CBR and DBR denote the cluster-based ranking and document-based ranking. The third parameter is λ. In the experiments, λ =0.9 is used because it has the best performance.

Table 1.5: Descriptive statistics of the experimental data.

Description	ArgoUML	Apache	Subversion
Language	Java	Java	C
Software Type	UML Tool	HTTP Server	SCM Tool
SCM	Subversion	Subversion	Subversion
Repository	Tigris	Bugzilla	Tigris
Data Period	00/02–07/05	01/01–07/02	01/03–07/05
# of Bug Repts.	4,613	2,773	2,296
# of Duplicates	**294**	**588**	**305**

To evaluate the detection schemes, we use the recall rate metrics. The recall rate defined as the percentage of the duplicates that can correctly find the corresponding master bug reports in the top-*n* recommendations.

$$Recall\ Rate = \frac{N_{corr}}{N_{total}} \qquad (1)$$

Equation (1) illustrates how to calculate the recall rate, where N_{corr} is the number of duplicate reports that are correctly identified, and N_{total} is the total number of duplicate reports.

3. Performance Study

The performance evaluation is based on the duplicate bug report recommendation list. The recommendation list can help software analysts to rapidly find the duplicate bug reports. From figure 4 to figure 9, the horizontal axis is the ranking size of the list and the vertical axis is the recall rate. The ranking size is equal to the recommendation list size. Because n-gram and Cluster Shrinkage have many parameter settings, different experiments have been conducted to study their influences.

The first experiment is to study the λ value in CS. In this experiment, we only use the simple NLP processing and CS. From Tables 1.6 to 1.8, we can find that CS performs well when λ = 0.9 and λ =1.0. In the following experiments, we use λ =0.9 because its performance is slightly better than λ =1.0 in three projects.

The second experiment is to study the combinations of different approaches. From figure 4 to figure 6, NLP means the basic natural language processing, CN means character-level n-gram, WN means word-level n-gram, CS means the cluster shrinkage technique, and ALL means the combination of all approaches. From these figures, we can find that ALL outperforms others.

The third experiment is to study *n* value in character-level n-gram. In Table 1.9, we can find these are no significant difference between different n values. In our experiments, we use n = 6 because its performance is slightly better. In figure 7 to figure 9, we also compare our approaches ALL+CBR and ALL+DBR with past work of Hiew[4], Runeson et al. [5], and Sureka et al. [6]. We can see that our approache have the better performance than others.

Table 1.6: The detection performance of the SVN project for different λ value in CS.

	Document-based Rank (DBR)							Cluster-based Rank (CBR)						
Rank	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0
1	28.65	28.09	28.09	29.21	29.21	28.65	28.65	26.40	27.53	27.53	28.09	28.09	28.65	28.65
2	35.96	36.52	35.39	37.64	38.20	37.08	37.08	32.58	34.83	34.27	35.39	35.39	37.08	37.08
3	39.33	39.89	41.01	41.01	41.57	41.01	41.57	36.52	37.64	37.08	38.76	39.89	41.57	41.57
4	42.13	42.70	44.94	45.51	45.51	45.51	44.94	39.33	39.33	39.89	42.70	43.82	44.94	44.94
5	46.07	47.19	49.44	48.88	50.00	51.12	50.56	43.26	43.82	46.07	47.75	48.88	50.56	50.56
6	48.88	50.56	52.81	53.93	55.06	55.06	55.06	47.75	49.44	50.56	51.69	53.93	55.62	55.62
7	53.93	54.49	55.06	56.74	57.87	58.43	58.99	50.56	51.69	51.69	54.49	57.30	58.99	58.99
8	55.62	55.62	57.30	57.87	61.24	61.80	61.24	53.37	53.37	55.06	58.43	59.55	61.24	61.80
9	58.43	57.87	58.99	60.67	63.48	65.17	64.04	55.06	56.74	57.87	60.67	61.80	64.04	64.61
10	58.43	58.43	59.55	63.48	65.17	66.29	67.42	57.87	60.11	60.67	62.36	65.17	67.42	67.98
11	58.43	58.99	60.67	65.17	65.73	67.42	68.54	59.55	60.11	61.24	62.92	66.85	68.54	69.10
12	58.99	60.67	61.80	65.17	65.73	68.54	70.22	61.24	61.24	61.80	63.48	69.10	70.22	70.22
13	60.67	61.80	65.17	66.85	66.85	69.10	70.79	61.24	61.24	62.36	64.04	69.66	70.79	70.79
14	60.67	61.80	65.17	66.85	67.42	70.79	71.91	61.80	61.80	63.48	65.17	70.22	71.91	71.91
15	60.67	61.80	65.17	66.85	67.42	70.79	72.47	62.36	62.36	64.04	65.73	70.22	72.47	72.47
16	60.67	62.36	65.73	66.85	67.42	70.79	73.03	62.92	62.92	64.61	66.85	71.35	73.60	73.60
17	61.24	63.48	66.29	67.42	67.98	72.47	73.60	62.92	63.48	64.61	66.85	72.47	74.16	74.16
18	62.36	65.73	67.42	68.54	70.79	74.16	74.72	63.48	64.61	65.17	67.42	74.16	75.28	75.28

Table 1.7: The detection performance of the ArgoUML project for different λ value in CS.

	Document-based Rank (DBR)							Cluster-based Rank (CBR)						
Rank	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0
1	23.66	23.88	24.55	24.11	24.11	23.44	23.44	20.98	21.21	21.88	22.99	23.21	23.44	23.44
2	33.04	33.48	34.82	35.49	36.16	36.38	36.38	31.03	31.92	33.26	35.04	35.27	35.49	35.49
3	36.61	37.50	39.29	41.07	42.86	43.08	43.53	35.49	36.83	38.84	41.07	41.74	42.41	42.41
4	40.85	41.74	44.64	46.43	48.44	48.88	49.11	39.51	41.29	43.53	45.76	46.88	47.99	47.99
5	44.64	45.09	48.44	50.22	51.79	52.90	52.46	43.53	45.09	47.77	49.78	50.89	52.01	52.01
6	46.65	47.99	50.89	53.13	54.24	55.80	55.13	47.32	48.44	51.34	52.46	53.79	54.91	54.91
7	48.88	50.45	52.46	55.36	55.58	57.59	56.92	49.55	50.45	53.13	54.24	55.80	56.70	56.70
8	51.56	52.90	55.80	57.59	58.04	59.82	59.82	52.01	52.68	55.36	56.47	58.48	59.38	59.38
9	53.13	54.69	56.92	58.93	59.82	62.05	61.61	54.02	54.69	57.14	58.26	60.27	60.94	60.94
10	55.13	56.03	58.04	60.49	62.05	63.62	62.95	55.36	56.47	58.48	59.60	61.83	62.05	62.05
11	56.25	57.81	59.60	61.83	63.39	64.06	64.06	56.25	57.59	58.93	60.27	62.95	62.72	62.72
12	58.26	59.15	60.94	63.62	65.40	66.52	65.18	58.48	59.38	60.49	62.05	64.06	64.06	64.06
13	59.38	59.82	62.28	63.84	65.85	67.41	66.52	59.82	60.27	61.38	63.39	64.73	64.96	65.18
14	60.27	60.49	62.72	64.73	66.74	68.53	68.30	60.94	61.16	61.83	64.51	65.85	66.52	66.96
15	60.94	61.16	63.17	65.40	67.19	68.97	69.20	61.83	62.05	62.72	65.63	66.74	67.41	67.86
16	61.83	62.50	63.84	66.52	68.30	69.87	69.87	62.95	62.95	63.62	66.96	68.08	68.53	68.75
17	62.28	62.95	64.73	67.41	69.42	70.98	70.76	63.62	64.06	64.51	67.86	69.42	69.42	69.42
18	62.95	64.06	65.85	68.08	70.31	71.43	71.43	64.06	64.96	65.63	68.53	69.87	70.09	70.09

Table 1.8: The detection performance of the Apache project for different λ value in CS.

	Document-based Rank (DBR)							Cluster-based Rank (CBR)						
Rank	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0	λ =0.0	λ =0.1	λ =0.3	λ =0.5	λ =0.7	λ =0.9	λ =1.0
1	23.50	23.50	24.00	24.00	24.00	23.50	22.75	21.50	21.50	21.75	21.75	22.50	22.75	23.25
2	34.50	35.00	35.25	35.25	35.75	35.25	35.00	30.25	30.50	31.75	32.50	34.00	34.75	35.50
3	40.25	40.00	40.50	42.75	43.75	45.25	45.00	37.00	37.50	39.75	41.75	43.75	44.75	45.75
4	44.00	44.00	46.00	48.25	49.50	50.75	51.25	40.75	41.00	43.50	46.50	49.00	50.50	51.75
5	46.25	46.75	49.00	51.50	54.25	56.00	56.50	43.50	44.00	47.50	50.75	53.75	55.75	57.00
6	47.25	48.25	51.00	54.75	57.75	58.75	59.50	46.00	46.50	50.75	53.25	56.75	59.00	60.25
7	49.50	51.25	54.50	57.75	60.00	61.00	62.75	48.00	49.50	53.75	56.00	59.75	62.00	63.50
8	52.00	53.50	56.50	60.00	62.75	64.50	65.00	50.50	52.25	55.75	59.00	62.75	64.75	66.00
9	54.75	56.25	58.75	62.25	66.00	67.00	68.00	53.50	54.50	57.75	61.50	65.50	67.75	68.50
10	57.00	57.75	60.75	63.75	67.75	69.75	70.50	55.25	56.00	59.50	64.00	68.25	70.50	70.75
11	58.25	59.75	63.00	65.50	70.25	72.25	73.00	56.50	57.00	61.25	66.50	71.50	72.75	73.00
12	60.00	61.50	65.00	68.50	72.00	74.50	75.50	58.00	58.50	63.50	69.25	74.75	75.50	75.75
13	61.50	62.75	66.25	70.50	73.25	75.75	76.50	58.75	59.75	65.00	71.25	76.50	77.00	77.00
14	62.50	64.50	68.00	72.00	74.75	77.25	78.00	60.50	61.75	67.00	74.00	78.00	78.25	78.25
15	64.00	66.25	68.75	72.75	75.50	78.25	79.00	61.50	63.00	68.25	76.00	78.75	79.00	79.00
16	65.25	67.00	69.50	74.25	76.75	79.25	80.00	63.00	64.25	70.50	77.75	79.75	80.25	80.00
17	66.25	67.25	70.50	75.25	78.00	80.00	80.75	64.00	65.50	72.50	79.00	80.50	81.00	80.75
18	66.50	67.75	71.75	75.75	78.25	80.50	81.25	65.75	66.75	74.50	80.25	81.25	81.50	81.25

Table 1.9: The detection performance for of ALL (NLP+WN+CN+CS, $\lambda = 0.9$) for different n values in character-level n-gram

	SVN-Document-based Rank(DBR)						ArgoUML-Document-based Rank (DBR)						Apache-Document-based Rank (DBR)					
Rank	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
1	37.08	36.52	38.20	34.83	34.83	33.71	26.56	28.13	28.35	27.23	27.01	27.01	33.00	33.00	33.00	32.25	32.50	32.25
2	53.37	53.93	54.49	51.69	51.69	50.00	41.29	43.08	43.08	43.08	42.19	42.41	48.50	48.25	48.00	47.50	47.50	47.25
3	63.48	62.92	62.36	62.36	60.67	56.74	49.55	50.89	50.89	50.22	50.00	49.11	56.75	56.75	56.75	56.75	56.50	56.75
4	69.10	70.79	68.54	67.98	65.17	64.04	56.25	58.04	57.81	56.70	55.58	54.69	64.25	63.75	64.50	64.25	64.25	64.50
5	71.35	72.47	70.79	70.79	70.22	66.85	61.83	61.61	61.61	60.27	59.38	59.15	68.75	68.50	68.00	69.00	69.00	68.75
6	73.03	73.03	72.47	73.03	73.60	70.79	64.96	64.51	63.62	62.95	61.83	61.61	71.50	71.50	72.00	72.00	72.00	71.75
7	74.16	74.72	74.16	74.16	74.16	71.35	66.96	65.63	65.40	64.51	63.84	64.29	74.00	73.75	74.50	73.75	74.00	74.75
8	75.84	75.84	74.72	74.72	74.72	73.60	68.30	67.63	66.96	66.29	66.07	66.29	75.75	76.25	76.50	76.50	76.25	76.25
9	76.40	76.40	74.72	74.72	75.28	73.60	69.87	69.20	67.86	67.41	67.63	67.41	77.50	77.75	78.75	79.00	78.75	78.50
10	78.09	76.97	75.84	75.84	75.28	73.60	71.43	70.54	69.20	68.75	68.97	68.53	79.50	80.25	80.75	81.50	81.50	81.00
11	78.65	77.53	76.97	77.53	77.53	75.28	71.88	71.43	70.31	70.09	70.09	70.09	81.75	82.25	82.75	83.00	83.00	83.00
12	80.90	78.65	77.53	79.21	79.21	76.97	73.21	72.99	71.65	70.76	70.98	70.98	84.00	83.50	84.00	84.25	84.00	84.00
13	81.46	80.90	79.21	79.21	79.78	77.53	74.55	74.11	73.21	71.65	72.10	71.88	85.25	85.25	85.50	85.50	85.00	85.25
14	81.46	80.90	81.46	79.78	80.90	78.09	75.00	74.55	74.11	73.21	72.54	72.32	86.50	86.25	86.50	86.50	86.00	86.00
15	84.27	82.02	82.58	80.34	82.02	78.65	75.67	75.00	74.55	73.66	73.21	72.77	87.50	87.25	87.50	87.25	87.25	87.00
16	84.83	82.02	82.58	80.90	82.58	79.78	75.89	75.67	75.00	74.11	73.66	73.66	88.50	87.75	88.00	87.75	87.75	87.50
17	84.83	83.15	82.58	82.02	83.15	80.34	76.34	75.67	75.67	75.00	74.78	74.33	89.00	88.50	88.75	88.25	88.50	88.50
18	84.83	83.71	83.15	83.15	84.27	81.46	76.79	76.56	76.12	75.67	75.22	75.00	89.75	89.50	89.50	89.50	89.75	90.00
19	85.39	84.27	83.71	84.27	84.83	82.58	76.79	77.01	76.34	76.12	75.22	75.22	90.50	90.25	90.25	90.25	90.25	90.75

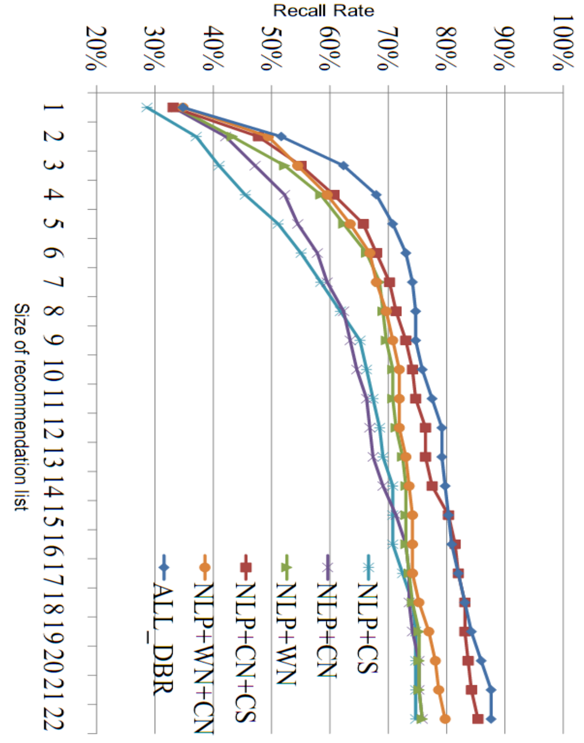


Figure 4: SVN combination

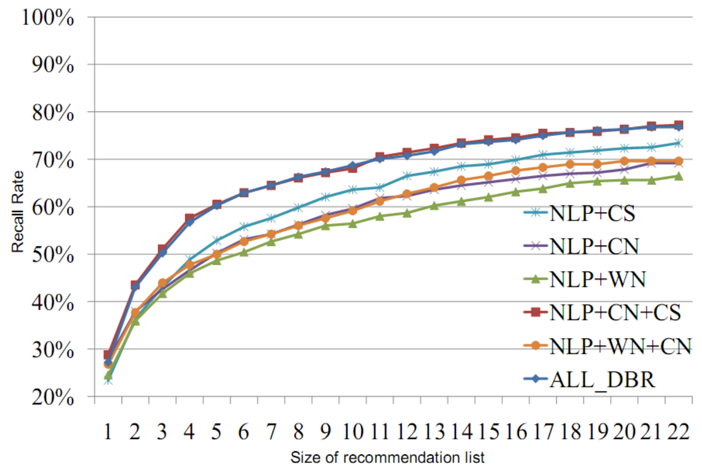


Figure 5: ArgoUML combination

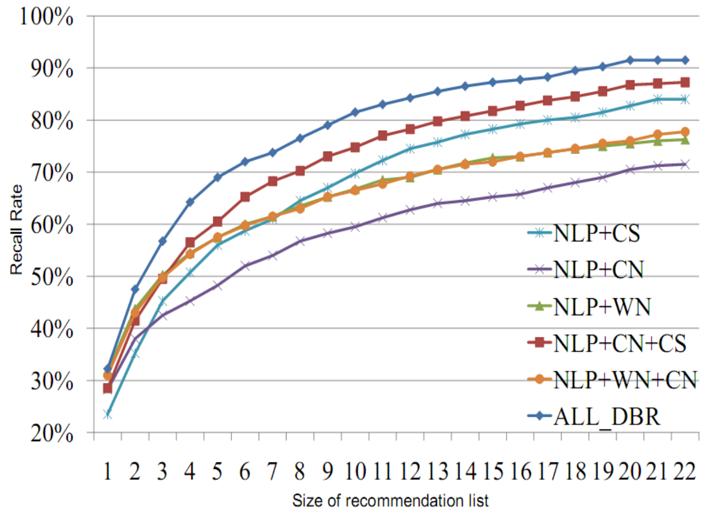


Figure 6: Apache combination

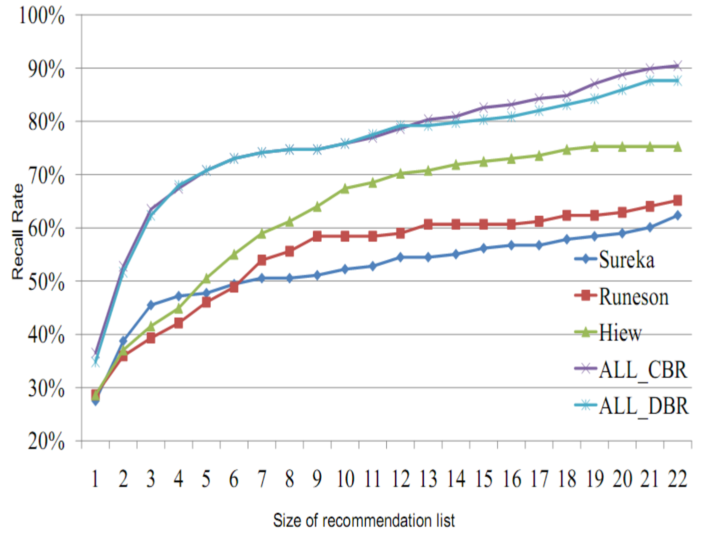


Figure 7: Detection performance with previous work for the SVN project

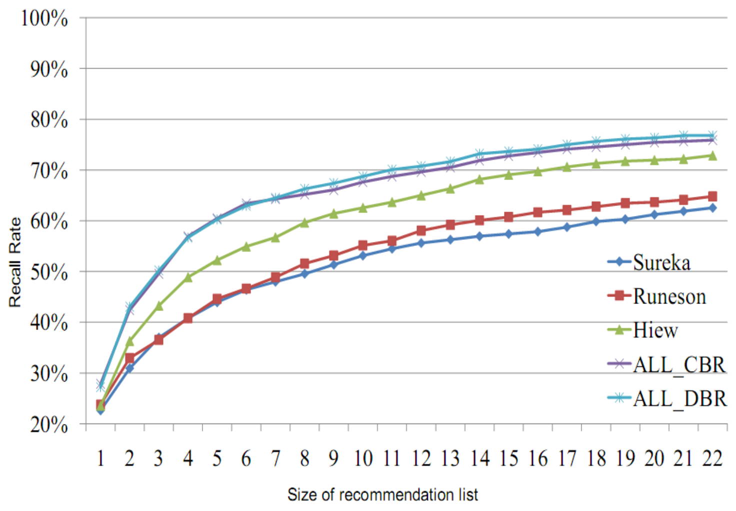


Figure 8: Detection performance with previous work for the ArgoUML project

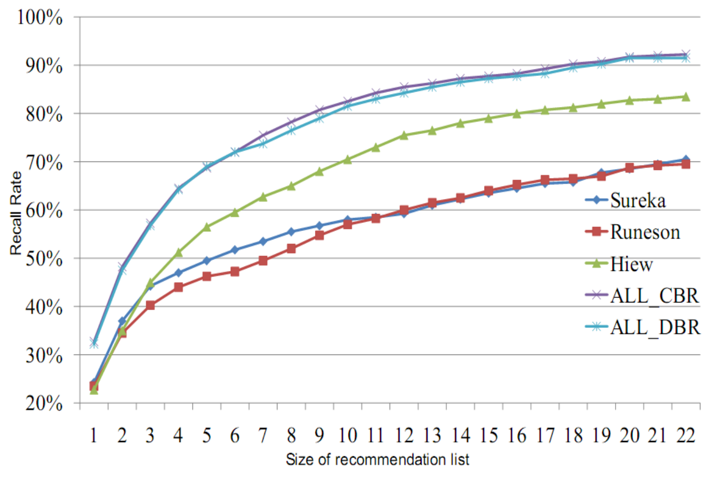


Figure 9: Detection performance with previous work for the Apache project

VI. CONCLUSION

Duplication detection is an important issue for software maintenance in recent years. In this study, we propose a detection scheme using n-gram features and the cluster shrink-age technique. From the empirical experiments on three open-source software projects, the proposed scheme shows its effectiveness in duplication detection.

There are some advanced issues in this research direction. For example, the se- mantic relationships among bug reports can be extracted to identify the bug reports with similar semantic meaning. The implicit domain knowledge may also help the duplication detection work. We believe that the discussion of these issues can further promote the performance advance in the duplication detection work.

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy, *Coping with an Open Bug Repository*, in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eX-change (eclipse '05), 2005, pp. 35–39.
[2] John Anvik, Lyndon Hiew, and Gail C. Murphy, *Who Should Fix this Bug?* in Proceedings of the 28th International Conference on Software

Engineering (ICSE'06*)*.New York, NY, USA: ACM, 2006, pp. 361–370.

[3] Yguaratã Cerqueira Cavalcanti, Eduardo Santana de Almeida, Carlos Eduardo Al- buquerque da Cunha, Daniel Lucrédio, and Silvio Romero de Lemos Meira, *An Initial Study on the Bug Report Duplication Problem*, in Proceedings of the 14th European Conference on Software Maintenance and Reengineering, 2010, pp. 264–276

[4] Lyndon Hiew, *Assisted Detection of Duplicate Bug Reports*, Master Thesis, The University of British Columbia, May 2006.

[5] Per Runeson, Magnus Alexandersson, and Oskar Nyholm, *Detection of Duplicate Defect Reports Using Natural Language Processing*, in Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), 2007, pp. 499–510.

[6] Ashish Sureka and Pankaj Jalote, *Detecting Duplicate Bug Report Using Character N-Gram-based Features,* in Proceedings of the 17th Asia Pacific Software Engineering Conference, 2010, pp. 366–374

[7] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun, *An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information*, in Proceedings of the 30th International Conference on Software Engineering (ICSE '08).New York, NY, USA: ACM, 2008, pp. 461–470.

[8] Yguarat˜ a Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Ed-uardo Santana de Almeida, Daniel Lucr´ edio, Carlos Eduardo Albuquerque da Cunha, and Silvio Romero de Lemos Meira, "*One Step More to Understand the Bug Report Duplication Problem*," in Proceedings of the 24th Brazilian Symposium on Software Engineering (SBES'10), 2010, pp. 148–157. [Online]. Available: http://dx.doi.org/10.1109/SBES.2010.12.

[9] Nicholas Jalbert and Westley Weimer, "*Automated Duplicate Detection for Bug Tracking Systems*," in Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008), 2008, pp. 52–61.

AUTHORS

Phuc Nhan Minh - received the B.Sc in Information Technology from the University of Natural Sciences- Ho Chi Minh City, Viet Nam and M.Sc in Computer Science from Yuan Ze University, Taiwan. He is now working in Tra Vinh University, Viet Nam. Email: nhanminhphuc@tvu.edu.vn.