applied sciences

MDPI

Article

A Weighted PageRank-Based Bug Report Summarization Method Using Bug Report Relationships

Beomjun Kim [1], Sungwon Kang [1] and Seonah Lee [2,3,*]

[1] School of Computing, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Korea; bjk129@kaist.ac.kr (B.K.); sungwon.kang@kaist.ac.kr (S.K.)
[2] Department of Aerospace and Software Engineering, Gyeongsang National University, 501 Jinju-daero, Jinju City, Gyeongsang-namdo 52828, Korea
[3] Department of of Informatics, Gyeongsang National University, 501 Jinju-daero, Jinju City, Gyeongsang-namdo 52828, Korea
* Correspondence: saleese@gnu.ac.kr; Tel.: +82-55-772-1377

Received: 15 October 2019; Accepted: 8 December 2019; Published: 11 December 2019

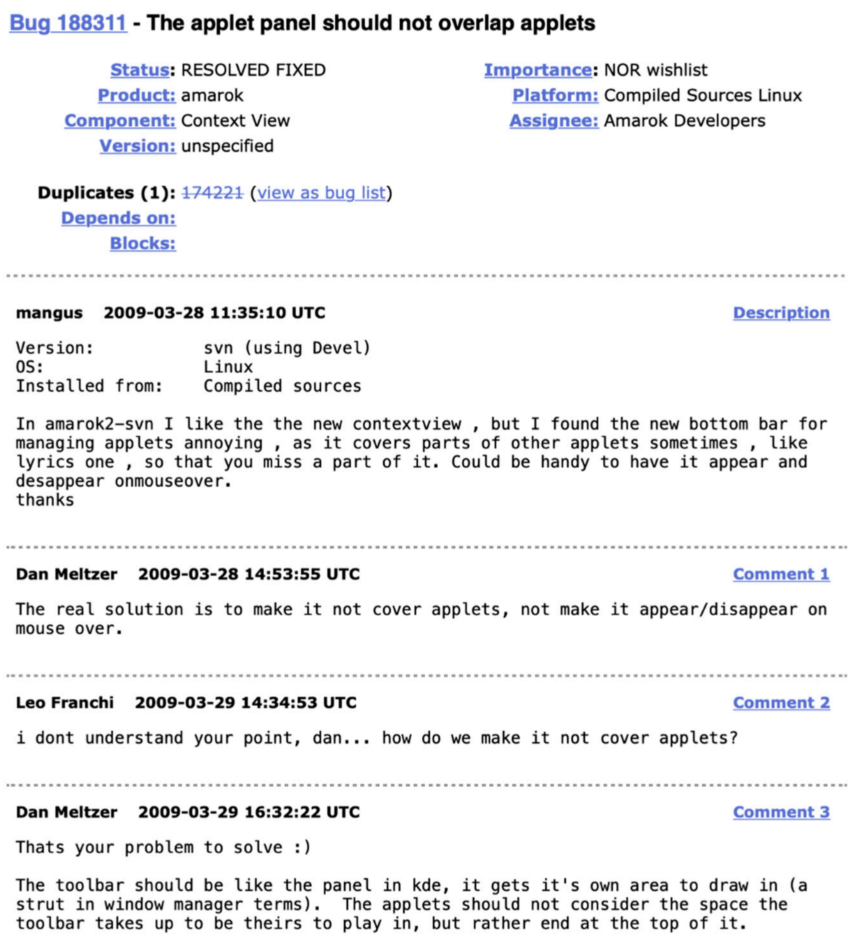
Abstract: For software maintenance, bug reports provide useful information to developers because they can be used for various tasks such as debugging and understanding previous changes. However, as they are typically written in the form of conversations among developers, bug reports tend to be unnecessarily long and verbose, with the consequence that developers often have difficulties reading or understanding bug reports. To mitigate this problem, methods that automatically generate a summary of bug reports have been proposed, and various related studies have been conducted. However, existing bug report summarization methods have not fully exploited the inherent characteristics of bug reports. In this paper, we propose a bug report summarization method that uses the weighted-PageRank algorithm and exploits the 'duplicates', 'blocks', and 'depends-on' relationships between bug reports. The experimental results show that our method outperforms the state-of-the-art method in terms of both the quality of the summary and the number of applicable bug reports.

Keywords: data-based software engineering; text summarization; PageRank; issue tracking system; bug report relationships

1. Introduction

Bug reports provide important information for developers to maintain a software system. Bug reports are used for developers to share and discuss information with others when fixing bugs, which account for approximately 40% of software maintenance tasks. Even after a bug is fixed, bug reports are used for various purposes, such as checking the history of past changes and referring to relevant bug fixes [1]. In general, bug reports are maintained in bug-tracking systems such as Bugzilla and Jira. Figure 1 shows an example of a bug report on the Bugzilla platform.

As shown in Figure 1, each bug report contains a detailed description of the bug with a unique number and a title. Below the description of the bug, comments are listed that contain the developer's discussion on how to reproduce or fix the bug. Due to the characteristics of these bug reports, bug reports are generally considered as a document in the form of a sequential conversation. In addition, there are various metadata in bug reports, such as tags indicating the type or status of bugs and information about developers (e.g., bug manager and assignee), and links to the relevant bug reports.



Bug 188311 - The applet panel should not overlap applets

Status: RESOLVED FIXED
Product: amarok
Component: Context View
Version: unspecified
Importance: NOR wishlist
Platform: Compiled Sources Linux
Assignee: Amarok Developers

Duplicates (1): ~~174221~~ (view as bug list)
Depends on:
Blocks:

mangus 2009-03-28 11:35:10 UTC Description

```
Version:           svn (using Devel)
OS:                Linux
Installed from:    Compiled sources

In amarok2-svn I like the the new contextview , but I found the new bottom bar for
managing applets annoying , as it covers parts of other applets sometimes , like
lyrics one , so that you miss a part of it. Could be handy to have it appear and
desappear onmouseover.
thanks
```

Dan Meltzer 2009-03-28 14:53:55 UTC Comment 1

```
The real solution is to make it not cover applets, not make it appear/disappear on
mouse over.
```

Leo Franchi 2009-03-29 14:34:53 UTC Comment 2

```
i dont understand your point, dan... how do we make it not cover applets?
```

Dan Meltzer 2009-03-29 16:32:22 UTC Comment 3

```
Thats your problem to solve :)

The toolbar should be like the panel in kde, it gets it's own area to draw in (a
strut in window manager terms).  The applets should not consider the space the
toolbar takes up to be theirs to play in, but rather end at the top of it.
```

Figure 1. An example of a bug report.

One example of when bug reports are utilized during software maintenance is a bug report duplicate detection task [2]. This task determines whether similar bug reports exist for the same bug and tag the bug reports with "duplicates." To determine whether similar bug reports exist for the same bug, developers often perform keyword-based searches in bug-tracking systems. When bug reports are retrieved, a developer reads the bug reports one by one, which is very time consuming. Because a bug report is in the form of a conversation, the bug report is often long and verbose. Therefore, developers sometimes have difficulty finding the desired information in the bug report. To solve the problem, researchers have proposed using a summary of bug reports. As an alternative to this approach, a developer can write a summary of the bug when a bug is fixed. However, this requires additional work by the developer and, thus, may not be adopted in practice. As a result, researchers are working on automatic methods for summarizing bug reports.

To propose a method for automatically generating a summary of bug reports, Rastkar et al. [2] applied an existing supervised learning technique to bug reports and conducted experiments using the summary of bug reports so that actual developers can detect duplicate bug reports. The experiment showed that the time spent on the detection work was reduced by approximately 30% without affecting the quality of the output. In addition, when conducting a questionnaire survey on the developers who participated in the work, they found that developers responded that when they used the summary of bug reports, they could easily understand the bug reports. This showed the effectiveness of using the summary statement. Since then, various bug report summarization methods have been studied to improve the quality of bug report summaries. In most cases, however, researchers have focused on learning methods, such as adopting machine learning algorithms [3–5]. They have not focused on how to exploit specific domain knowledge, such as the characteristics of bug reports. Jiang et al. [6] used the characteristics of the bug report to improve the quality of bug report summaries. They considered that existing bug report summarization methods are used for detecting duplicate bugs and proposed a bug report summarization method using duplicate bug reports. Jiang et al. [6] showed that using duplicates relationships among bug reports can improve the quality of bug report summaries. However, among various possible bug report relationships (Cf. Table 1), they used only duplicates relationships with the consequence that the effect of applying their method to bug reports is limited.

Table 1. The relationships between bug reports on the Jira platform.

Relationship Name	Inward Description	Outward Description
Blocked	Blocks	Blocked by
Bonfire testing	Testing discovered	Discovered while testing
Cloners	Is cloned by	Clones
Dependent	Depends-on	Is depended on by
Derived	Derived from	Derives
Design	Has design on	Is design for
Duplicate	Duplicates	Is duplicated by
Epic	Belongs to Epic	Is the Epic for
Relates	Relates to	Relates to
User test	Was in test	Participants were
Caused by	Causes	Caused by

In this paper, we propose a method to automatically generate a summary of a bug report, which we call the Relation-based Bug Report Summarizer (RBRS). It extends the existing summary method [6] with the 'blocks' and 'depends-on' dependencies as well as the 'duplicates' relationship. In order to consider the significance of multiple relationships simultaneously, we used weighted-PageRank algorithm, which extends PageRank algorithm used in previous study [6]. In addition, we extended the BRC (Bug Report Corpus) corpus, which was previously used as the subject of a bug report summary study, and created a corpus that can show the relationship between various bug reports. We evaluated our proposed method with the corpus as test data. As a result, we showed that the precision, recall, and F1 score all improved in terms of the quality of the bug report compared to the existing method [6]. In addition, we also showed that using more relationships can extend the application scope of the summary method [6].

This paper is organized as follows. Section 2 introduces existing studies related to bug report summarization methods. Section 3 explains the background knowledge of bug reports and then describes our proposed method for summarizing the bug reports based on weighted-PageRank using dependency relationships. Section 4 explains the experimental set-up. Section 5 evaluates our implemented bug report summary method. Section 6 discusses the additional experimental result. Section 7 discusses the limitations and improvements in our work. Finally, Section 8 concludes our work and discusses future research directions.

2. Related Work

We divide our discussion on related work in three sub-sections: bug report summarization (Section 2.1), bug report summarization using bug report relationships (Section 2.2), and PageRank algorithm (Section 2.3).

2.1. Research on Bug Report Summarization

In 1958, Luhn [7] first proposed a text summarization method. Since then, various summarization methods have been studied, ranging from simple tf-idf (Term Frequency – Inverse Document Frequency) based methods to complex machine learning-based methods [8]. In general, text summarization can be classified into extractive summarization and abstract summarization according to the method of generating summary statements [2]. Extractive summarization is an approach for organizing a summary by extracting a paragraph, sentence, or word of high importance from the original text. An abstract summarization is a method for generating abstract sentences based on the meaning of the original text. In general, extractive summarization has been used mainly because abstract summarization is much more difficult. In recent years, with the advance of machine-learning technology, research into abstract summarization has increased.

Bug report summarization methods are mainly developed in the direction of extractive summarization. The methods can be divided into supervised learning and unsupervised learning. Supervised learning is an approach for learning a model with tagged data. Rastkar et al. [9] first proposed a bug report summarization method using a supervised learning technique. Rastkar et al. [9] noted that bug reports can be treated as interactive documents. They adopted the Murray and Carenini-supervised [10] learning-based summary method that can be applicable to the minutes and email threads for bug reports. For an experiment, Rastkar et al. [2] wrote a hand-written summary called the golden summary of 36 bug reports. They then trained the summarizer using the golden summary. Unsupervised learning is an approach for learning a model with untagged data. Mani et al. [4] proposed an unsupervised summarization method. They proposed a method for removing noise from bug reports, and based on this, they applied four kinds of unsupervised learning-based algorithms for bug reports. Lotufo et al. [3] also proposed an unsupervised summarization method, which applied the PageRank algorithm to take advantage of similarities between sentences in bug reports.

2.2. Research on Bug Report Summarization using Bug Report Relationships

Jiang et al. [6] considered the fact that duplicated bug reports have textual similarity to the original bug report. They proposed a bug report summarization method using the PageRank algorithm, where they created new OSCAR data that contained duplicated bug reports. Based on these data, they combined Rastkar et al.'s [2] supervised summarization method with the PageRank-based unsupervised summarization method to improve the quality of bug report summaries.

In addition to the duplicates relationship, there are also various associations between bug reports. Table 1 shows the relationships between bug reports, such as blocked and bonfire testing, defined in the user guide for the Jira bug tracking system.

Robert et al. identified three types of relationships among bug reports including duplicate, dependency and reference [11]. In particular, duplicate and dependency relationships were classified as formal relationships since they are explicitly presented in bug reports. While the duplicates relationship has already been used in previous work [6], dependency relationship has received little focus in bug report summarization. A dependency relationship is bidirectional and can be refined into blocks and depends-on relationships according to the direction. The two relationships are often used to indicate the precedence of bug fixes or feature implementations. Many bug tracking platforms, in particular, provide the ability to display dependency graphs or trees using these two relationships. That is, the two dependencies have higher utilization and a higher frequency of use than the other relationships. Therefore, we choose the three relationships of duplicates, blocks, and depends-on to use between bug reports for our bug report summarization method.

2.3. PageRank Algorithm

The PageRank algorithm is described in Page et al. [12] as a method for weighting a web document based on relative importance. It was developed for search engine research, and the Google search engine was developed and is serviced based on this. The PageRank algorithm focuses on the link relationship between webpages and uses the link structure existing on the webpages. In the graph, all webpages become nodes, and link relationships among webpages become edges. Every node has a unique PageRank value. Afterward, the PageRank value of the node is updated by summing the values normalized by the number of links from other pages to the page. This process can be expressed by the following formula: $PR(A)$ is the PageRank value of the A web page, T_x represents the other pages that point to the A web page, and $C(T_x)$ is the total number of links the T_x web page has. If the process of propagating the PageRank value is repeated, all nodes converge to a unique PageRank value, which represents the importance of the webpage.

$$PR(A) = (1 - d) + \mathrm{d}\left(\frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right) \quad (1)$$

In the PageRank algorithm, node A's PageRank score PR(A) is propagated equally by PR(A)/C(A) to all nodes connected to A. C(A) is the number of nodes connected with node A.

In addition to calculating the importance of webpages, the PageRank algorithm has been applied to various fields, including summarization tasks. For example, Mihalcea and Tarau [13] proposed TextRank, which is a document summary and keyword extraction algorithm. For a summarization task, TextRank considers each sentence as a node and takes the similarity value between each pair of sentences as the weight of the edge. The importance of a sentence can be measured and prioritized based on the assumption that a sentence similar to other sentences is an important sentence. However, the TextRank method generally uses only one target document. In our method, the bug reports in the duplicate, depends-on, blocks relationships are used as an additional information to increase summarization quality.

3. A Bug Report Summarization Method Based on Dependency Relationships

We propose a summary method that utilizes the dependency relationships between two bug reports in addition to duplicate relationships. Our proposed method, relation-based bug report summarizer (RBRS), is shown in Figure 2. As shown in Figure 2, the input is the bug report that a user wants to summarize. Then, RBRS analyzes the duplicates, blocks, or depends-on associations of the bug report in the bug report repository and collects the relevant bug reports. Next, RBRS preprocesses the bug reports by using the preprocessing module. Afterward, RBRS forms a bug report graph with the preprocessed bug reports and their relationships. Then, RBRS performs the weighted-PageRank algorithm to assign a PageRank score to each sentence of the bug report. Additionally, RBRS assigns a score to each sentence of the bug report, based on Rastkar et al.'s [9] supervised summarization method. RBRS combines the two scores by weighting them using the ranking merger module. The combined score is the final score for each sentence. RBRS composes a summary by selecting the top 25% of sentences according to their final scores.

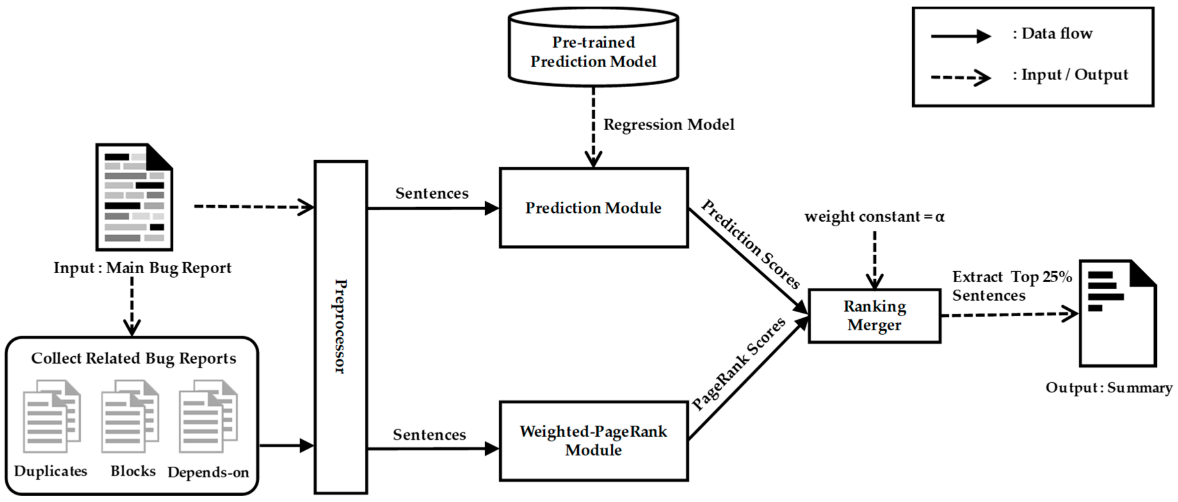
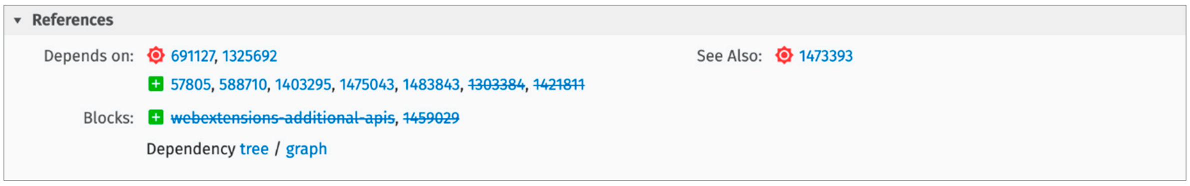


Figure 2. An overview of the proposed method.

Section 3.1 introduces the relationships between bug reports. Section 3.2 describes the process of forming a graph among bug reports to apply the weighted-PageRank algorithm. Section 3.3 describes the preprocessing of bug reports. Section 3.4 explains how to apply the weighted-PageRank algorithm to our method. Finally, Section 3.5 describes the process of integrating the weighted-PageRank algorithm with Rastkar et al.'s [2] method by using the ranking merger module.

3.1. Association Relationships between Bug Reports

In general, a bug-tracking platform uses a variety of metadata to manage bug reports. A bug report contains a variety of information, not only the issue name, the author, and the date of the issue but also the type of bug, the situation, and the assignee who is responsible for bug resolution. In a bug report, association relationships between bug reports are specified as references metadata to help fix the bug or to help resolve other relevant bugs after the bug fix. For example, as shown in Figure 3, on the Bugzilla platform, various bug reports are explicitly linked in the references of the bug report according to the type of relationship. The association relationship of these bug reports occurs at various times. When creating a bug report, the author of the bug report may link other bug reports if needed. In the bug fix phase, developers may discuss the issue and add additional links to other bug reports. Even after the bug is fixed, developers can add links if the bug has an additional relationship with other bug reports.



▾ References

Depends on: 691127, 1325692
57805, 588710, 1403295, 1475043, 1483843, ~~1303384~~, ~~1421811~~

See Also: 1473393

Blocks: ~~webextensions-additional-apis~~, ~~1459029~~

Dependency tree / graph

Figure 3. References metadata in a bug report.

There are various kinds of association relationships between bug reports. Various bug-tracking platforms, such as Jira and Bugzilla, have defined the association relationships that are possible. As shown in Table 1, there are 10 or more relationships.

In practice, three association relationships are most commonly used, of which the first is the duplicates. When A duplicates B, A and B are duplicated bug reports that contain the same content. Duplicated bug reports exist between bug reports because when a user or developer reports a bug or issue, there is no guarantee that the bug or issue has already been reported. A user or developer can perform keyword-based searches on a bug-tracking platform. However, there are practical difficulties in having all users and developers search prior to writing bug reports. Therefore, once a bug report is created, it is common for the developer assigned the bug fix to recognize the duplicated bug report, create a duplicates link to the bug report, and close the bug report. The duplicated bug reports contain the same bugs or issues, so textual similarities between the duplicated bug reports exist.

The second and third association relationships are blocks and depends-on relationships. Here, we define the two relationships, blocks and depends-on, as the dependency relationship between bug reports. The two are inverses of each other. If A blocks B or B depends on A, then bug A must be fixed before bug B can be fixed. For requirements such as adding a feature, not just bug fixing, you can use blocks and depends-on relationships, which means that adding A functionality must be preceded by adding B functionality. For example, in the Firefox browser on iOS, Bug 1473704 blocks Bug 1473716. In this case, Bug 1473704 is a feature that requires a user-configurable setting for the Firefox browser's dark mode (dimming the screen). Bug 1473716 is a report related to the bug that the slide bar to control dark mode needs to be redesigned. As in the example, most of the bug reports that have dependency relationships with each other tend to deal with similar subjects, which means that major keywords are shared and there is also a textual similarity. Therefore, this shows the textual similarity between these bug reports. We view that the dependency relationships can be used to improve the summary quality of bug reports as the duplicates relationship was used to do so in [6].

3.2. Generating a Bug Report Graph

To apply the PageRank algorithm, there is a need to build a graph using bug reports. This process consists of two steps. The first step is to build a graph with directionality of bug reports as nodes using a breadth-first-search (BFS) algorithm. The second step is to refine the node of the bug report graph at the sentence level. The graph completed through these two steps is called a bug report correlation graph. When building a bug report correlation graph, there are two configurable variables. The first is the depth of search when applying the BFS algorithm. The second is a weighted value set for each edge.

The first step is to create a graph with a direction that uses bug reports as nodes. For this purpose, the BFS algorithm is applied. First, RBRS sets the main bug report as the root node. Next, RBRS analyzes the metadata of the bug report to find all associated bug reports for which there is an explicitly linked association relationship. Then, RBRS makes each bug report a new node and associates it with the main bug report. At this time, a graph is formed according to the relationships with the main bug report as follows.

3.2.1. Duplicates Relationship

This adds a bidirectional edge between nodes of all duplicates relationships to form a complete graph. Then, each edge is weighted with W_{dup} (default value = 1.0).

3.2.2. Blocks Relationship

This creates an edge with a weight of W_{blo} (default value = 1.0) from the main bug report to a bug report of a blocks relationship and an edge with a weight of W_{dep} (default value = 1.0) in the opposite direction.

3.2.3. Depends-On Relationship

This creates an edge with a weight of W_{dep} (default value = 1.0) from the main bug report to a bug report of a depends-on relationship and an edge with a weight of W_{blo} (default value = 1.0) in the opposite direction.

The above step describes a BFS search process with a depth of 1. Figure 4 shows a bug report graph at depth 1 for a bug report with one block report, two depends-on bug reports, and three duplicates bug reports. The process is repeated for all nodes according to the predefined depth of search. A bug report graph with depth N is a graph that contains all the bug reports that can be reached at most N links (duplicates, blocks, depends-on relationship links) from the main bug report.

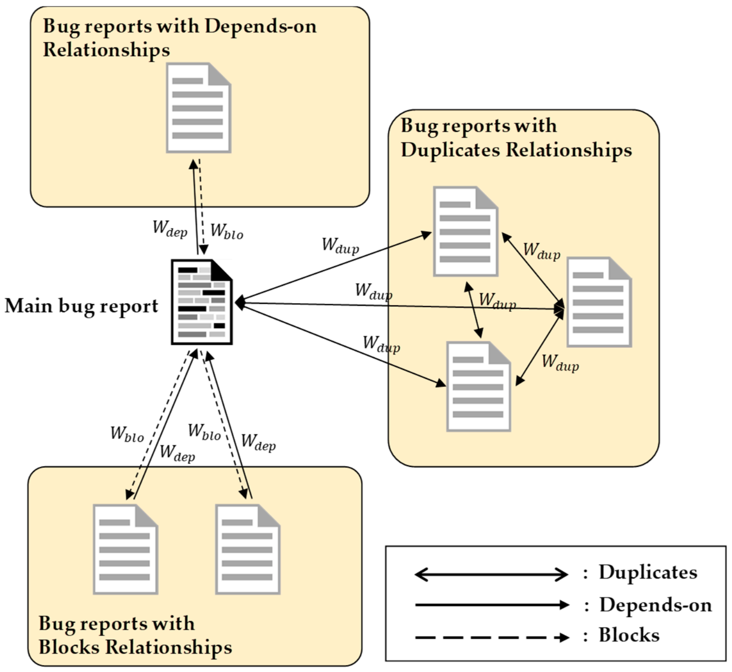

Bug reports with Depends-on Relationships
Bug reports with Duplicates Relationships
Main bug report
Bug reports with Blocks Relationships
W_{dep}
W_{blo}
W_{dup}
: Duplicates
: Depends-on
: Blocks

Figure 4. A bug report graph at depth 1.

The second step is to refine the nodes of the bug report graph. In the first step, each node in the bug report graph represents a bug report. However, to calculate the similarity and score of the sentences using the PageRank algorithm, each node needs to be separated in units of sentences. Therefore, RBRS refines a node of the bug report into sentence nodes of the bug report. At this time, the edge of the graph is processed as follows. Let us assume that bug report node A and bug report node B exist. If bug report A has 10 sentences, and bug report B has 5 sentences, node A is divided into nodes A_1 through A_{10}, and node B is divided into nodes B_1 through B_5. After that, if there is an edge from node A to B, the edge is refined into the edges from node A_1 to nodes B_1–B_5, from node A_2 to nodes B_1–B_5, ... from node A_{10} to nodes B_1~B_5. In total, $10 \times 5 = 50$ edges are created. The opposite edge is refined in the same way. Additionally, as the nodes from A_1 to A_{10} are the sentences belonging to the same bug report A, the internal nodes are connected to each other in a bidirectional edge of weight $\text{W}_{internal}$ (default value $=$ 1.0) in a complete graph where each pair of distinct nodes is connected. At the end of this step, a bug report correlation graph is completely formed.

3.3. Preprocessing

Preprocessing consists of three steps. The first step is bug report filtering, the second step is comment filtering of bug reports, and the third step is sentence preprocessing.

The bug report filtering step excludes certain kinds of bug reports from the graph. The bug tracking platform groups bugs or issues in a variety of ways. The most common method uses meta bug reports. For example, if there are various bug reports in the application's push notifications, the platform creates a META report named <[META] Tracking bugs for push notification>, and then links the depends-on relationship to the bug reports related to the push notifications. The platform can also link block associations in the opposite direction. In this case, bug reports can be easily categorized and managed, and developers can discuss the overall situations in META reports. However, as these META reports cover a wide range of topics, the specified relationships may not be used in their meaning; thus, the textual similarities between the linked bug reports in the META report are low. The bug reports also share few keywords. Therefore, RBRS removes META reports from the graph.

The second step is to filter the comments in the bug report. One of the major differences between a bug report and a regular document is that the bug report contains not only natural language but also non-natural statements such as code snippets and stack traces. The bug report also contains free expressions, such as listing keywords that are not finalized in sentences. Since this feature is not suitable for applying natural language processing-based methods, research has been conducted to remove such information as noise. For example, Huai et al. [5] categorized the bug report comments into seven categories: Bug Description, Opinion Expression, Information Giving, Emotion Expressed, Fix Solution, Information Seeking, and Meta/Code.

In addition, Mani et al. [4] improved the bug report summary quality by eliminating stack traces and code snippets. We use the results in Huai et al. [5] and Mani et al. [4] to remove Meta/Code, stack traces, and code snippets.

The final step of preprocessing covers sentence separation, stopword removal, and stemming. For the preprocessing, we used NLTK (Natural Language Toolkit), a Python-based natural language processing library. In our method, we used a list of more than 100 stopwords of NLTK. Additionally we defined and used stopwords such as bug, build, tag, and developer that are commonly used in bug reports.

3.4. Weighted-PageRank

The PageRank algorithm equally weights all edges connected in a node. However, when PageRank is applied to various fields, there is a need to give different weights to edges. Xing and Ghorbani [14] and Xie et al. [15] proposed a weighted-PageRank algorithm that assigns weights differently according to the types of links in web pages. In this paper, it was also necessary to measure the effects of different relationships (i.e., duplicates, blocks, and depends-on) on the summary. Additionally, to achieve the highest summary quality, RBRS applies the weighted-PageRank algorithm and gives different weights to the edges in different types of relationships.

To apply the weighted-PageRank algorithm, RBRS first assigns each weight value to the correlation-based bug report graph created in Section 3.2. Next, RBRS assigns a PageRank value of 1 / (number of nodes) to all nodes. Then, when the PageRank values of all nodes converge using the PageRank algorithm, each value becomes a PageRank score of the corresponding sentence. The higher the score, the higher the importance of the sentence, and the more similar sentences the sentence has.

The score propagation of the weighted PageRank algorithm is similar to that of the original PageRank algorithm. In the case of the weighted PageRank algorithm used in our proposed method RBRS, PR(A)/C(A) multiplied by edge weight value W, $0 \leq W \leq 1$, is propagated, instead of PR(A)/C(A).

3.5. Ranking Merger

In their method in [6], Jiang et al. used not only the scores of the sentences calculated by the PageRank algorithm but also Rastkar et al.'s [2] scores of sentences obtained through the supervised regression model because, when duplicated bug reports do not exist, it is difficult to apply their method to bug reports. Additionally, the combination of the two methods yields higher performance. In this paper, RBRS also uses both the scores of the sentences calculated by the PageRank algorithm and Rastkar et al.'s [2] score obtained from the supervised learning-based regression model to calculate the final score of each sentence.

Rastkar et al. [2] created BRC data based on 36 bug reports and a handwritten summary. Additionally, Rastkar et al. [2] proposed a method for summarizing bug reports by learning a logistic regression model by extracting 24 features from each sentence of a bug report. Based on the summary method in [2], RBRS trains a logistic regression model based on the same 24 features. Using the logistic regression model, the probability that each sentence of the bug report will be included in the summary sentence can be calculated, and this is defined as the score S_{pred} of the prediction module. Likewise, the score of each sentence obtained through the weighted-PageRank algorithm is defined as S_{WPR}, and each score is normalized to have a value between 0 and 1, and then those scores are summed. The score is calculated as follows.

$$Score = \alpha \times S_{WPR} + (1 - \alpha) \times S_{pred} \tag{2}$$

The weight α has a range of $0 \leq \alpha \leq 1$. We experimented with the values of a by changing it from 0 to 1 by 0.25 to find its optimal value. The score obtained through the weighted sum is considered to be the final score of each sentence. Each sentence is sorted according to the score, and the summary is extracted by using the top 25% of the sentences. This extraction method by 25%, which is described in Rastkar et al. [2], is widely used in various existing bug report summarization, including the method in [2] and is considered to be the most appropriate length for a summary.

4. Experimental Set-Up

4.1. Research Questions

There are mainly three kinds of relationships between bug reports. Past research only used the duplicates relationship to improve the quality of a bug report summary. In this paper, we intend to use two other relationships, blocks and depends-on, to improve the quality of a bug report summary. Therefore, we select the following main research questions:

RQ 1. Can our proposed bug report summarization method improve the quality of a summary by using the dependency relationships (blocks, depends-on)?
RQ 2. How does the number of bug reports' dependency relationships affect summary quality?

4.2. Experimental Subjects

The experimental data commonly used in existing bug report summarization methods are BRC data, which are described in Rastkar et al. [2]. Based on 36 bug reports and a summary of three commenters, BRC selects approximately 25% of sentences per bug report to form a summary and defines the summary as the golden summary. However, out of the 36 bug reports in the BRC data, only 13 bug reports have relationships with each other. Jiang et al. [6] noted this problem and built MBRC(Modified BRC) and OSCAR data. In this paper, we propose a summary method that uses all three associations of bug reports. Therefore, we investigated 50,000 bug reports created in 2018 for the Mozilla project. Based on these Mozilla project's one-year bug reports, we selected 36 bug reports that satisfy the following conditions (the 36 bug reports are in [16]):

Condition 1. A bug report has a duplicates relationship with other bug reports.
Condition 2. A bug report has a blocks or depends-on relationship with other bug reports.
Condition 3. A bug report is longer than 300 words.

In the case of condition 3, since a short report does not need summarization, a bug report having a length of at least 300 words is generally selected as an experimental subject. The information on selected bug reports is shown in Table 2. For example, the total number of bug reports that have duplicates relationships is 36. This means that each bug report has one or more duplicates relationships. The total number of duplicates relationships in the bug reports is 47, and the average number of duplicates relationships per bug report is 1.31.

Table 2. Information about the relationships of 36 selected bug reports.

Relationships	**The Number of Bug Reports that Have Relationships**		**The Number of Relationships in the Bug Reports**	
	Total Number	**Average Number**	**Total Number**	**Average Number**
Duplicates	36	1.0	47	1.31
Depends-On	18	0.5	24	0.67
Blocks	26	0.72	36	1.0

After selecting 36 bug reports as an experimental subject, an annotation process is performed to generate summary data for our experiment. The first author of this paper selects approximately 25% of the important sentences for all 36 bug reports. As the commenting work, the first author proceeds in the same way that is described by Rastkar et al. [2] for building the BRC data. Commenting work follows the steps below:

1. The participant reads and understands a bug report.
2. The participant writes his/her own summary statements with 25% of the total number of sentences in the bug report.
3. The participant associates each sentence of the summary with the original sentences.
4. The participant selects the linked sentences as the golden summary.

The difference of our commenting work from Rastkar et al. [2] is that the first author of this paper creates the golden summary, while Rastkar et al. [2] recruited three participants to perform the commenting work and create the golden summaries consisting of the sentences that two participants selected. Due to the difference, our commenting work can be more subjective than Rastkar et al.'s work [2]. To mitigate this risk, the first author carefully reads each bug report and, after feeling confident that he completely understands the report, creates summaries. We also conducted an additional experiment, described in Section 6, by recruiting three participants.

We completed the annotation work to build the relationship-based bug report summary data. The average number of sentences in the summary data accounts for 24% of the original text.

4.3. Experimental Procedure

4.3.1. Experiment for Research Question 1 (RQ 1)

Three experiments are conducted to answer research question 1. The first and second experiments compare our method with Jiang et al.'s [6] method. By doing so, we intend to show that the dependency relationships of bug reports can improve the summary quality.

In the first experiment, we intend to show that our method is effective when Jiang et al.'s [6] method is not applied due to no duplicates relationships. Therefore, when we use the experimental subjects, 36 bug reports, we do not use their duplicates relationships but use their blocks and depends-on relationships. Under the circumstance, we compare the summary quality of our method RBRS with the previous method PRST (PageRank-based Summarization Technique) [6]. This experiment is for bug reports that have no duplicates relationships but dependency relationships.

In the second experiment, we compare the summary quality for the bug reports that have both duplicates relationships and dependency relationships. In this case, we use all of blocks, depends-on and duplicates relationships of 36 bug reports. We then compare our method RBRS with Jiang et al.'s method PRST [6].

The third experiment attempts to show that both dependencies of blocks and depends-on have a positive effect on improving the summary quality. For the experiment, we alter the weights of the relationships.

4.3.2. Experiment for RQ 2

For research question 2, two experiments are conducted to analyze summary quality according to the number and extent of dependency relationships. The first experiment analyzes the trend of precision change by changing the maximum depth during the BFS search in the bug report graph formation step. The second experiment analyzes the trend of precision change by limiting the number of dependency relationships of the bug report to a maximum of N.

While some bug reports do not have dependency relationships, many bug reports have dozens of dependency relationships. When applying our method to real cases, we need to know how the number of relationships affects summary quality. We also need to find optimal numbers of relationships.

In addition, when a bug report graph is formed to apply the PageRank algorithm, the maximum depth of the search can be set. The higher the maximum depth, the greater the number of bug reports used for summarization. Therefore, the two experiments for RQ 2 are conducted to analyze the trend of summary quality as the number of relationships changes and to suggest the variable setting that shows the highest quality when our method is applied.

4.4. Measurement

We use three metrics to measure the accuracy of bug report summaries: precision, recall, and F1-score, which are commonly used in bug report summaries. For a bug report, approximately 25% of the sentences constitute an extracted summary. A statement included in an extracted summary is called golden summary sentences (GSS) [2,6]. In addition, the set of sentences selected by a bug summarization method is defined as S.

The precision rate is defined as the ratio of sentences included in the golden summary sentences among the sentences selected by a bug summarization method.

$$Precision = \frac{|S \cap GSS|}{|S|} \tag{3}$$

The *Recall* rate is defined as the percentage of sentences selected by a bug summarization method among the sentences in the golden summary sentences.

$$Recall = \frac{|S \cap GSS|}{|GSS|} \tag{4}$$

The *F1* score is a harmonic mean of precision and recall and is defined as follows:

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5}$$

We measure the precision, recall and F1-score per the summary for each bug report and average the values of precision, those of recall and those of F-score.

5. Evaluation Results

Before analyzing the effect of dependency relationships on summary quality, this section first analyzes the scope of application. Jiang et al.'s [6] PRST effectively improved the summary quality for bug reports by using the duplicates relationships in bug reports. However, there was no significant performance improvement for bug reports that had no duplicates relationships. In contrast, we use more relationships, blocks and depends-on as well as duplicates associations, which can greatly increase the scope of bug reports that can be applied. For this coverage analysis, we analyzed 50,000 bug reports created in 2018 for the Mozilla project. Table 3 shows the analysis results of 44,600 bug reports, except for the 5400 bug reports that were closed.

Table 3. Analysis results for the relationships of 44,600 bug reports.

Presence or Absence of Duplicates Relationships / Presence or Absence of Dependency Relationships	Number of Bug Reports Containing Duplicates Relationships	Number of Bug Reports not Containing Duplicates Relationships
Number of Bug Reports Containing Dependency Relationships	1345 (3.0%)	14,317 (32.1%)
Number of Bug Reports not Containing Dependency Relationships	978 (2.2%)	27,960 (62.7%)

As a result, Jiang et al.'s [6] PRST can be effectively applied to only 5.2% of all bug reports. However, our method, which uses all three relationships, is expected to be effectively applied to 37.5% of all bug reports.

Section 5.1 describes the experiments and results for the main research questions, and Section 5.2 describes the experiments and results for the subquestions.

5.1. RQ 1. The Effect of Dependency Relationships on Summary Quality

PageRank algorithm-based summarization works on the assumption that sentences similar to other sentences are important. Jiang et al.'s [6] PRST used the PageRank algorithm to better identify important sentences using the textual similarities of the bug reports in the duplicates relationship. Similarly, our method RBRS focuses on the fact that there are text similarities among subjects and keywords among the bug reports that are in dependency relationships. To demonstrate this, we conducted experiments to show that this text similarity can be used to summarize bug reports.

5.1.1. Experiment 1-1. Summary Quality Comparison of Bug Reports without Duplicates Relationships

If the duplicates relationships do not exist while dependency relationships of blocks and depends-on exist, PRST [6] cannot be applied. To evaluate this condition, we compared the summary quality of the proposed method RBRS with the PRST of the previous study when the duplicates relationships of 36 bug reports were ignored. The results are shown in Table 4.

Table 4. The results for Experiment 1-1 *.

Technique	Precision	Recall	F1-Score
PRST	44.50%	44.79%	44.64%
RBRS	50.91%	50.93%	50.92%

* We have reported the experimental results of Table 4 and Table 5 in [17]. We are allowed to use the tables in this paper by Korea Information Processing Society.

The results for Experiment 1-1 in Table 4 shows that when no duplicates relationships exist, the precision, recall, and F1 scores improve by 6.41%, 6.14%, 6.28%, respectively. In our experiments, the number of sentences in extracted summary $|S|$ is fixed at 25% of the total number of sentences in the bug report. Also, the number of golden summary sentences $|GSS|$ is about 24% of the total number of sentences in the bug report. As the values of $|S|$ and $|GSS|$, which are the denominators of precision and recall, are very similar, precision and recall in our experiments show similar values as well. F1 score, the harmonic mean of precision and recall, show a similar value. As the result, in precision, recall and F1 score, RBRS made the similar percentages of improvement.

The statistics of 44,600 bug reports described in Table 3 shows that 14,317 bug reports have no duplicates relationships but have dependency relationships, which account for approximately 32.1% of all bug reports. Therefore, we interpret that our method, RBRS, by using the dependency relationships of the bug report, can improve the summary quality by approximately 6.4% for the bug reports that account for 32.1% of the total bug report.

In addition, most bug reports that have duplicates relationships are often found to be duplicated immediately after the report, and the bug report is closed immediately. As a result, there is often a lack of content for developers to discuss or resolve the bug. In this case, certain sentences or keywords, such as the symptoms or reproducing process created during the bug report, are duplicated. It seems that the summarization result can be biased in such duplicate sentences. Because of the limitations of these duplicates relationships, this bias occurs. Using additional dependency relationships can reduce the bias, resulting in improvement in summary quality.

5.1.2. Experiment 1-2. Summary Quality Comparison of Bug Reports with Duplicates Relationships

If duplicates relationships exist while dependency relationships of blocks and depends-on exist, both PRST and RBRS can be effectively applied. To evaluate this condition, we compare the summary quality of PRST and RBRS for 36 bug reports that satisfy the above conditions. The results are shown in Table 5.

Table 5. The results for experiments 1-2.

Technique	Precision	Recall	F1-Score
PRST	49.71%	49.77%	49.74%
RBRS	51.69%	51.82%	51.76%

Experimental results 1-2 show that when duplicates relationships exist, the precision, recall, and F1 scores improve by approximately 2% compared to PRST. An improvement of 2%, which is the result of Experiment 1-1, can hardly be seen as a significant improvement in accuracy. Therefore, we interpret that the dependency relationships do not yield a significant improvement in summary quality when used with the duplicates relationships. However, based on 44,600 bug reports, our method can be expected to improve accuracy by approximately 2% for the 3% of the total bug report, which is the number shown in Table 3.

5.1.3. Experiment 1-3. The Effect of Two Dependency Relationships on Summary Quality

This experiment determines whether the dependency relationships of depends-on and blocks contribute to improving summary quality. In the experiment, we also find the optimal value of edge weighting of the dependency relationships in the weighted-PageRank algorithm. In the experiment, when creating a bug report graph, different edge weights can be assigned according to the kind of relationships between bug reports. We analyze the change in summary quality by increasing the weight of W_{dep} and the weight of W_{blo} by 0.25 from 0 to 1. If the weight is zero, the relationship is ignored. The weights of the inner edges of the sentences in a bug report and the edges of the duplicates relationship are both set to 1.0. The experimental results are shown in Table 6.

Table 6. The results for experiment 1-3.

W_{dep} / W_{blo}	$W_{dep} = 0$	$W_{dep} = 0.25$	$W_{dep} = 0.5$	$W_{dep} = 0.75$	$W_{dep} = 1.0$
$W_{blo} = 0$	44.50%	44.36%	45.31%	46.67%	47.23%
$W_{blo} = 0.25$	45.22%	45.49%	46.80%	48.23%	47.98%
$W_{blo} = 0.5$	46.75%	46.80%	48.63%	48.22%	49.03%
$W_{blo} = 0.75$	47.03%	47.82%	48.10%	49.42%	49.86%
$W_{blo} = 1.0$	47.96%	48.35%	49.29%	49.70%	50.91%

As a result of the experiment, both dependency relationships of blocks and depends-on show the highest precision when the edge weight is set to 1.0. Additionally, as W_{blo} decreases from 1 to 0, the average precision decreases by 3.4%. In the case of W_{dep}, the precision decrease is 3.3%. Both dependencies are found to have a positive effect on summary quality when applied to summary methods. There seems to be no significant difference between the two dependencies. Therefore, when applying this summary method, we suggest that the edge weights of the dependency relationships are all set to 1.0, as with the duplicates relationships. This means that three relationships of blocks, depends-on, and duplicates have the same weights in a bug report graph, which was described in Section 3.2.

5.2. RQ 2. The Effect of Number of Relationships on Summary Quality

5.2.1. Experiment 2-1. The Trend of Precision Change According to the Depth of Search in Breadth-First-Search (BFS)

In this experiment, we analyzed the change in summary quality by increasing the BFS search depth from 1 to 3 in the bug report graph formation step. The experimental results are shown in Figure 5. From depth 4 and above, as the depth increases, the size of the graph increases exponentially, and the algorithm execution time increases to a level where it is difficult to apply the algorithm to actual summarization. Therefore, depth 4 and above are excluded from the experiment.

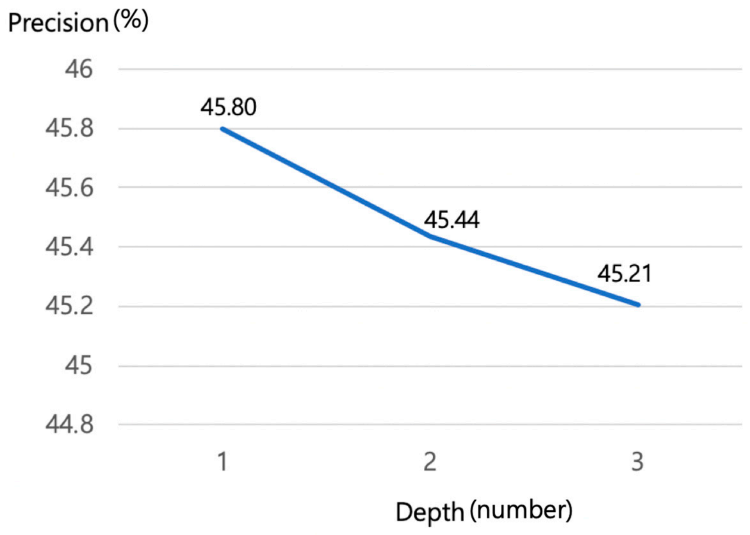


Figure 5. The results for experiment 2-1.

As a result, when the depth is 1, the graph shows the highest precision. That is, forming the graph using only the bug reports that are directly related to the bug report is effective in bug report summarization. As the depth of the search increases, the precision decreases. However, the reduction rate is very low, 0.4%. The decrease in precision is because bug reports with a distance of 2 or more from the main bug report are significantly smaller in text similarity to the main bug report than with a bug report with a distance of 1. As bug reports with low similarity are added to the graph, they cause a decrease in the importance of the most important sentences or keywords in the main bug report. However, due to the characteristics of the PageRank algorithm, only the nodes that are directly connected have the propagation of importance.

5.2.2. Experiment 2-2. The Trend in Precision Change by Limiting the Number of Relationships

In this experiment, we analyze the change in summary quality when limiting the number of relationships per bug report to a maximum of N, which means that the three types of relationships are limited to N per bug report. In this case, each bug report can have up to 3N relationships with other bug reports. For the experiment, we arbitrarily select N related bug reports. The results of analyzing the trend in precision change by increasing the maximum association limit from 1 to 6 are shown in Figure 6.

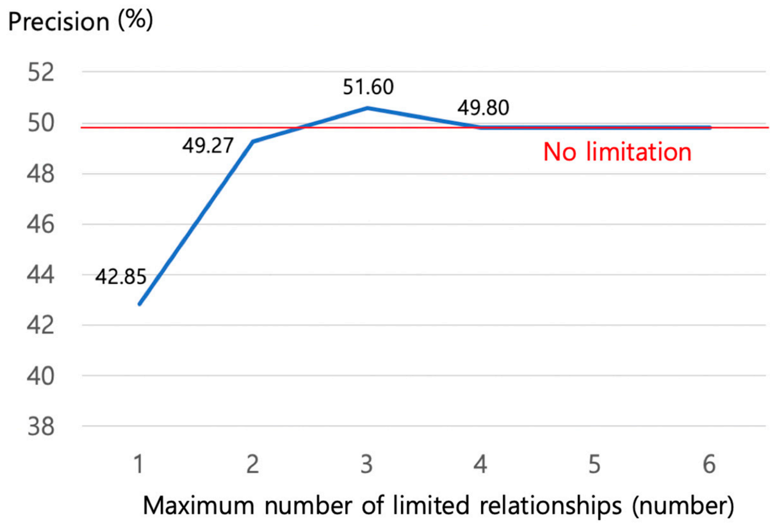


Figure 6. The results for experiment 2-2.

As a result, when the number of relationships is limited to one, the precision is very low, 42.85%. When the number of relationships is limited to three, the highest precision is 51.60%, which is higher than cases without limiting the number of relationships. If the number of relationships is limited to four or more, the same precision as the case of no limitation is shown because the number of bug reports that have more than 4 relationships among the bug reports to be tested seems to have no significant effect on precision. The low precision when the number of relationships is less than two is analyzed because there is not enough text to take advantage of the text similarity with the related bug report. When the number of relationships is three, it is expected to be the best number for applying the PageRank algorithm. Therefore, we suggest limiting the number of relationships to a maximum of three when our method is applied. In addition, for future research, we suggest a detailed experiment for change in precision as the size of text changes.

6. Additional Experiments

To increase the reliability of the experimental results in Section 5, we conduct two additional experiments. Sections 6.1 and 6.2 describe each experimental result.

6.1. Additional Experiment Based on the Data Used by Previous Work

We created our own experimental data as described in Section 4.2, because the data previously used in bug report summaries were generated without considering the dependency relationships of bug reports [2]. When we checked the previous data, only 7 out of 36 bug reports maintain the dependency relationships. Therefore, it was difficult to use the data for our experiment.

To confirm that similar results are consistently produced by our method even when different data sets are used, we ran the same experiment as Experiment 1-1 on seven bug reports that had dependency relationships among 36 bug reports in Rastkar et al. [2]. Table 7 shows the experimental result and, as shown in Table 7, RBRS yielded 55.93% precision value, while PRST yielded 50.43%. The 5.5% improvement of precision is similar to the precision improvement of 6.41% from the experimental result shown in Section 5.1.1.

Table 7. Additional results using the data used by previous work.

Technique	Precision	Recall	F1-Score
PRST	50.43%	52.06%	51.23%
RBRS	55.93%	58.07%	56.98%

6.2. Additional Experiment Using the Golden Summary Created by Three Participants

As described in Section 4.2, the first author of this paper solely created the golden summary for the experiments conducted in Section 5. Since only one person, who is not an actual project member, created the summaries, there is a possibility that the summaries may not have properly included the important sentences, which is the risk of biased data.

To mitigate this risk, we conducted an additional experiment by recruiting 3 participants who are computer science graduates of the first author's university. In the experiment, we selected 6 sample bug reports (the newly constructed golden summary corpus is available in [18]) out of 36 original bug reports. We explained to the three participants about the structure and creation process of bug reports and the procedure described in Section 4.2. They then annotated the 6 bug reports by following the procedure. Finally, the golden summary was constructed based on the sentences that were selected by two or more participants. We applied both PRST and RBRS methods on this new corpus. As shown in Table 8, the precision was improved by about 7.33%, which again shows that the proposed RBRS technique yields more accurate summary than the existing PRST technique.

Table 8. Additional results using the golden summary created by three participants.

Technique	Precision	Recall	F1-Score
PRST	42.40%	40.09%	41.21%
RBRS	49.73%	47.50%	48.60%

7. Limitations and Improvements

7.1. Limitations

The first limitation of this paper is the incompleteness of sentence separation in the text preprocessing phase. Bug reports, unlike general documents, do not consist of clearly distinguishable sentences. In some cases, bug reports simply list keywords or have a mixture of non-natural code and information. As mentioned in Section 3.3, we used NLTK, a Python-based natural language processing library. Additionally, we separated the sentences by defining special rules for bug reports. However, we observed that the separation was not complete. To solve this problem, we searched manually for sentences that were separated and then manually corrected the obvious statement separation errors. To completely remove such sentence separation errors, a sentence separation method for bug reports was required. If sentences are clearly separated, the accuracy of selecting right sentences for summarization would be improved.

The second limitation is that we created our own experimental data and the golden summaries were constructed by the first author of this paper alone. To mitigate the risk arising when we create our own experimental data, we checked the 36 bug reports in the previous data, found 7 bug reports that have dependency relationships, and ran the same experiment as Experiment 1-1 on the 7 bug reports, as described in Section 6.1. The experiment showed that, while PRST yielded 50.4% precision value, RBRS yielded 55.9%. The 5.5% improvement of precision is similar to the 6.41% improvement of precision in Experiment 1-1 (cf. Section 5.1.1). Also, to mitigate the risk arising when only one participant creates the experimental data, we conducted another experiment, where we recruited three participants to perform annotation tasks and create a golden summary, as described in Section 6.2. This additional experiment showed 7.33% improvement of precision, which is also an improvement rate similar to the precision improvement of 6.41% obtained in Experiment 1-1.

The third limitation is the improvement rate of our proposed method RBRS in comparison with the previous method PRST. The previous method showed 9.47% improvement in precision, 7.87% improvement in recall, and 8.92% improvement in F-score. On the other hand, when duplicates relationships in a baseline method are included, our method shows 1.98% improvement in precision, 2.05% improvement in recall, and 2.02% improvement in F-score. When duplicates relationships in a baseline method are excluded, our method shows 6.41% improvement in precision, 6.14% improvement in recall, and 6.28% improvement in F-score. While our experiment shows that our proposed method can significantly improve the accuracy of summarizing bug reports in the absence of the duplicates relationships, we also should seek a method to improve the summarization accuracy significantly in the presence of the duplicates relationships. One way to improve it would be the adoption of a deep learning technique for textual summarization.

7.2. Future Research

We propose three future projects. The first project is to find the most appropriate preprocessing phase for filtering bug reports, such as META reports, by detecting bug reports that can have a positive effect on the quality of bug report summaries or for separating sentences in bug reports. The second one is to extend the relationships of bug reports from explicit ones to implicit ones and study the weights of different relationships. The last one is to build larger, more accurate experiments with data from three or more experts (including over 100 bug reports).

8. Conclusions

In this paper, we proposed RBRS a method for automatically summarizing bug reports using three associations (duplicates, blocks, depends-on). To evaluate RBRS, we constructed a corpus of 36 bug reports including the correlation information of bug reports. We then compared RBRS with PRST [6]. Experimental results showed that our method RBRS yields 6.41% higher precision, 6.14% higher recall, and 6.28% higher F1 score than those of the previous method PRST.

The contributions of this paper are as follows. First, we proposed a weighted PageRank-based bug report summarization method that uses three kinds of association information for bug reports. Second, we built a corpus that can be used for bug report summary studies using the relationships of bug reports. Third, by analyzing the change in summary quality as the number and weight in relationships vary, we found a variable setting that maximizes summary quality. Fourth, we implemented the proposed method as a tool and performed a simulation to compare it with the existing method.

The bug report summary corpus constructed in this paper has a limitation that it is small and is written by persons who did not participate in building the bug reports. Previous bug report summarization research also had such limitation. So, for future research, we plan to use a larger bug report corpus to address this issue. We also found the shortcomings of the commonly used natural language processing library NLTK for preprocessing bug reports. We realized that the styles and formats of the sentences in bug reports are different from those of the normal sentences used in newspapers, novels, scientific papers, etc. Therefore, we also plan to conduct research on preprocessing techniques to enhance the quality of bug report summaries. As a final remark, although we improved the accuracy of summarizing bug reports with additional relationships of bug reports, we are not satisfied with the 50.92% F-score. For the improvement of summary accuracy, we are going to explore other machine-learning techniques, including a sequence to sequence model.

Author Contributions: Conceptualization, B.K. and S.L.; methodology, B.K.; software, B.K.; validation, B.K., S.K. and S.L.; data curation, B.K.; writing-original draft preparation, B.K.; writing-review and editing, S.K. and S.L.; supervision, S.K.

Funding: This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7066210) and by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1F1A1053262). This research was also supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2018R1D1A1A02085551).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barry, B.; Basili, V.R. Software defect reduction top 10 list. In *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*; Barry, B., Rombach, H.D., Zelkowitz, M.V., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 37, pp. 426–431.
2. Rastkar, S.; Murphy, G.C.; Murray, G. Automatic summarization of bug reports. *IEEE Trans. Softw. Eng.* **2014**, *40*, 366–380. [CrossRef]
3. Lotufo, R.; Malik, Z.; Czarnecki, K. Modelling the 'hurried' bug report reading process to summarize bug reports. *Empir. Softw. Eng.* **2015**, *20*, 516–548. [CrossRef]
4. Mani, S.; Catherine, R.; Sinha, V.S.; Dubey, A. AUSUM: Approach for unsupervised bug report summarization. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, Cary, NC, USA, 11–16 November 2012; ACM: Cary, NC, USA, 2012; pp. 1–11.
5. Huai, B.; Li, W.; Wu, Q.; Wang, M. Mining intentions to improve bug report summarization. *SEKE* **2018**, *2018*, 320–363. [CrossRef]
6. He, J.; Nazar, N.; Zhang, J.; Zhang, T.; Ren, Z. PRST: A pagerank-based summarization technique for summarizing bug reports with duplicates. *Int. J. Softw. Eng. Knowl. Eng.* **2017**, *27*, 869–896. [CrossRef]
7. Luhn, H.P. The automatic creation of literature abstracts. *IBM J. Res. Dev.* **1958**, *2*, 159–165. [CrossRef]
8. Li, X.; Jiang, H.; Liu, D.; Ren, Z.; Li, G. Unsupervised deep bug report summarization. In Proceedings of the 26th Conference on Program Comprehension, Gothenburg, Sweden, 28–29 May 2018; ACM: Gothenburg, Sweden, 2018; pp. 144–155.
9. Rastkar, S.; Murphy, G.C.; Murray, G. Summarizing software artifacts: A case study of bug reports. In Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, South Africa, 2–8 May 2010; IEEE: Cape Town, South Africa, 2010; pp. 505–514.
10. Murray, G.; Carenini, G. Summarizing spoken and written conversations. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Honolulu, HI, USA, 25–27 October 2008; Association for Computational Linguistics: Honolulu, HI, USA, 2008; pp. 773–782.
11. Sandusky, R.J.; Gasser, L.; Ripoche, G. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In Proceedings of the 1st International Workshop on Mining Software Repositories, Edinburgh, Scotland, UK, 25 May 2004; pp. 80–84.
12. Page, L.; Brin, S.; Motwani, R.; Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab, November 1999. Available online: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf (accessed on 5 October 2019).
13. Mihalcea, R.; Tarau, P. Textrank: Bringing order into texts. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, Barcelonaand, Spain, July 2004; Volume 4, pp. 404–411.
14. Xing, W.; Ghorbani, A. Weighted pagerank algorithm. In Proceedings of the Second Annual Conference on Communication Networks and Services Research, Fredericton, NB, Canada, 21 May 2004; IEEE: Fredericton, NB, Canada, 2004; pp. 305–314.
15. Xie, W.; Bindel, D.; Demers, A.; Gehrke, J. Edge-weighted personalized pagerank: Breaking a decade-old performance barrier. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, Australia, 10–13 August 2015; ACM: Sydney, Australia, 2015; pp. 1325–1334.
16. Kim, B. 36 bug reports used in GSS. Available online: http://salab.kaist.ac.kr/rbrs/rbrs_corpus.json (accessed on 10 December 2019).
17. Kim, B.; Kang, S.; Lee, S. A PageRank-based Bug Report Summarization Technique using Bug Report Relationships. In Proceedings of the 52th The KIPS Fall Conference; Korea Information Processing Society, 2019. In Press.
18. Kim, B. GSS for 6 sample bug reports. Available online: http://salab.kaist.ac.kr/rbrs/rbrs_corpus_2.json (accessed on 10 December 2019).