Al-Ahgaff University Journal of Computer Science and Mathematics
مجلة جامعة الأحقاف لعلوم الحاسوب والرياضيات

Vol. 1, Issue. 1, Jan 2023, pp. 17~24

Efficient Information Retrieval for bug localization using fewer objective functions

Ahmed Sheikh Al-Aidaroos
Information Technology Department, Al-Ahgaff University, Mukalla, Yemen
asalaidaroos@ahgaff.edu
Sameera Bin Ali Al haj
Information Technology Department, Al-Ahgaff University, Mukalla, Yemen
Seba.alhaj@gmail.com

Article Info

Article history:

Received jan 11, 2023
Revised jan 19, 2023
Accepted jan 25, 2023

Keywords:

Bug Report
Bug Localization
Information Retrieval based Bug Localization
Multi objective optimization
Automatic Query Reduction

ABSTRACT

Bug localization is one of the hardest, most costly, and most time-consuming tasks that faces software developers. Therefore, many Information Retrieval based Bug Localization (IRBL) approaches have been proposed to reduce the time and effort spent in localizing bugs. However, the quality of the queries positively affects the performance of the bug localization. ManQ is a multi-objective optimization based IRBL approach that seeks to improve query quality. It studied a set of IRBL approaches and converted them to a group of 15 objective functions. However, some researchers claimed there are a set of attributes that make a query high quality much less than ManQ's objective functions. Therefore, this study aims to adapt the ManQ approach by reducing the objective functions and keeping only the objective functions that correspond to these attributes. The adapted approach is named R-ManQ. The results show that both R-ManQ and ManQ have similar performances, but R-ManQ is much faster in terms of execution time and has a smaller number of query terms.

الخلاصة

تعد توطين الأخطاء من أصعب المهام وأكثرها تكلفة واستهلاكًا للوقت التي تواجه مطوري البرامج. لذلك، تم اقتراح العديد من مناهج توطين الأخطاء المستندة إلى استرداد المعلومات (IRBL) لتقليل الوقت والجهد المبذولين في توطين الأخطاء. ومع ذلك، تؤثر جودة الاستعلامات بشكل إيجابي على أداء توطين الخطأ. ManQ هو منهج IRBL متعدد الأهداف قائم على التحسين والذي سعى إلى تحسين جودة الاستعلام. فقد درس مجموعة من مناهج IRBL وحولتها إلى مجموعة من ١٥ وظيفة موضوعية. ومع ذلك، ادعى بعض الباحثين أن هناك مجموعة من السمات التي تجعل الاستعلام ذا جودة عالية أقل بكثير من وظائف ManQ الموضوعية. لذلك، تهدف هذه الدراسة إلى تكييف نهج ManQ عن طريق تقليل الوظائف الموضوعية والحفاظ فقط على الوظائف الموضوعية التي تتوافق مع هذه السمات. النهج المعدل سمي R-ManQ. تظهر النتائج أن كلا من R-ManQ وManQ متشابهان في الأداء، لكن R-ManQ أقل بكثير من حيث وقت التنفيذ وعدد مصطلحات الاستعلام.

1. INTRODUCTION)

All software goes through the testing and maintenance phases within the development life cycle in order to ensure that it is free from problems and errors. Despite this, many errors and issues appear, which are reported to the development teams during these phases. Those bugs are defined as an existing unexpected behavior or unexpected performance of a predefined functionality in the source code of the software [1, 2]. Those bugs are written by the end users in a text file in a natural language and then delivered to the development team in order to use it to understand the nature of those bugs and the reasons for their occurrence. This file is called a bug report [1, 3-5]. Therefore, a bug report is defined as a document created by end-users that describes this unexpected behavior and errors in the performance of the program when they use it and what the steps that lead to that behavior. A bug report contains several sections, including the bug ID, summary, and description [6].

According to [1, 6], these bugs go through various phases: unconfirmed, new, assigned, resolved, verified, closed, or reopened, as shown in Figure 1. But locating these bugs during the development phase is a difficult, costly, and time-consuming process [2, 3, 5, 7].

In addition, according to [8], it is more costly and complex to deal with bugs after the delivery of software. The process of searching and exploring source code for identifying the locations of buggy files by the developers depends on those reports, known as bug localization [5]. The localization of these bugs can be considered a somewhat simpler task for developers involved in the program development process, but a daunting task for other developers who are not involved in the development process of this software [5, 9]. In addition, manual localization of these bugs is considered difficult and expensive, especially when the software is large and complex [7, 10].

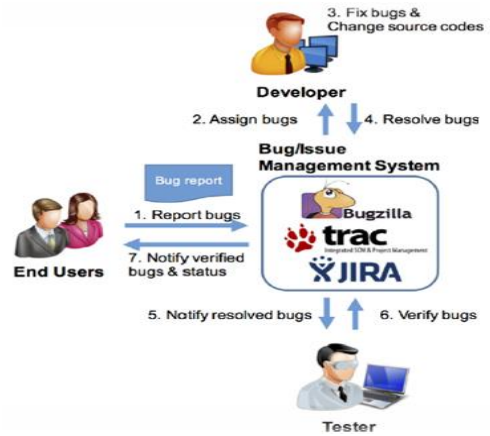

Figure. 1 Management Process of Bug [1]

In general, there are two approaches to localizing bugs: 1) dynamically locating bugs spectrum-based through program execution combined with techniques such as implementation, data monitoring, and breakpoints, and 2) statically locating bugs across various forms of analyses using bug reports along with the code, such a bug localization based on information retrieval. The first type is time-consuming and expensive, while the second type is preferred [11, 12].

Therefore, to reduce the time and effort spent in localizing these bugs, a number of IRBL approaches are proposed. The IRBL approach mechanism can be described by using a bug report as a query and source code elements as a document collection in order to find similarities between them to localize the bugs. The documents are ranked according to their relevance to the bug report, returning a ranked list of candidate source files, and then the developer checks the top-N of them, one at a time, and determines whether or not they contain the bug [7, 12-16].

According to [5], , the IRBL approaches must suggest a suitable search terms automatically from the bug report. The baseline approach applies the IR steps without any optimization. Also, it applies a three-step preprocessing step to a bug report and a buggy program (camel case splitting, stop word removal, and stemming) in order to generate the initial query and search the document corpus to retrieve relevant documents [12, 14]. However, the quality of the query is more important and it determines the performance of IRBL, whereas many IRBL approaches cannot perform well due to their use of low-quality queries [17-19].

The relevant document results in High-quality queries are at the top of the list of results, whereas low-quality queries either retrieve the requested document at the bottom of the list of results or return no result at all. Therefore, using low-quality queries makes bug localization a tedious, time-consuming, and poorly performing process [3, 11, 16, 20].

In addition to that, there are two strategies that produce a new, high-quality query based on a bug report. It involves an expansion strategy that expands the initial query with appropriate keywords and other reduction strategies that discard the noisy words from a bug report and focus on the important keywords only [20-22]. According to [11, 21] , the terms of a query selected from the bug report must be chosen carefully. Mills, et al. [16] show that bug reports alone contain enough keywords to form high-quality queries and, therefore, provide optimal performance for bug localization. Furthermore, even natural language-only bug reports might be a sufficient source of perfect terms for a query [11]. Rahman, et al. [11] define the optimal query as a query that can locate the buggy document at the top of its relevant results list.

Mills, et al. [16] proposed an approach called Query Quality Predictor (Q2P). It predicted the quality of bug report queries in concept location and traceability link recovery for Software Engineering (SE). Q2P employed a set of 28 measures of query properties (21 pre-retrieval properties and 7 post-retrieval properties) and then a machine-learning algorithm (Random Forest) in order to define a set of rules that will identify which of these queries are high-quality and which are low. Q2P was an improvement over a previous approach that relied on using only 21 pre-retrieval properties to predict the quality of the query [20]. By using Q2P, when the developer writes a low-quality query, the developer can directly reformulate it without spending time analyzing potentially useless documents retrieved by the text-relevant engine and reformatting that low-quality query.

Kim and Lee [15] proposed a new approach called ManQ based on a multi-objective genetic algorithm called Non-dominated Sorting Genetic Algorithm III (NSGA-III). ManQ focused on a set of IRBL approaches specialized in Automated Query Reduction (AQR). It investigated finding a high-quality sub-query. ManQ converted those approaches into a group of objectives and combined them into one approach. It extracted 15 objectives from them and created 15 objective functions that include eight objectives to maintain the return values of Query Quality Properties (QQP) assessment metrics, four objectives to maintain the important terms for IRB, two objective functions to maintain initial information, i.e., cosine similarity and term-occurrence in the context of the sentences, and one objective to reduce the query length. ManQ introduced a better performance in IRBL. ManQ went through three steps. 1. calculating objective functions; 2. implementing the NSGA-III algorithm and initializing the final query selection; and 3. implementing bug localization. The first step was calculating 15 objective functions that were divided into four groups, group one of which includes eight objective functions ($f_1(Q) - f_8(Q)$) to maintain the return values of Q2P assessment metrics. They were finding only the pre-retrieval property measures (coherency, similarity, term relatedness, and specificity). Group two included four objective functions ($f_9(Q)$- $f_{12}(Q)$) to maintain the important terms for IRBL. In addition, group three consisted two objective functions ($\boldsymbol{f_{13}}(Q)$and $\boldsymbol{f_{14}}(Q)$) to maintain initial information, i.e., cosine similarity and term-occurrence of a context of the sentence. The last group consisted of one objective function $f_{15}(Q)$ to reduce the query length. After completing this step, ManQ moved to the second step, which was implementing the NSGA-III algorithm, which resulted in multiple solutions from which ManQ chose a final query through the union of selective queries from each function. The last step in ManQ was implementing bug localization, where after obtaining the optimal query, ManQ submitted the query to a popular search engine for document search called Lucene in order to find the buggy files in the source code [23].

In addition, a recently published empirical study [11] reported that the bug reports that can contain terms but not software information are useful and sufficient for the bug localization process and give high results. They compared the graph-based IRBL approaches that were used to extract the important terms with a number of other IRBL approaches. They concluded that the graph-based approaches were the best approach for extracting the important terms. In addition, the authors apply a comparison between the optimal high-quality queries, which achieve high results in retrieving information, and those of lower quality, and they claimed there are a set of attributes that make a query high-quality. These attributes are: keywords of high quality queries are less frequent within a bug report, are less ambiguous (i.e., have less entropy), are more likely to be found in the description section of a bug report, are more likely to be nouns, and the optimal candidates are likely to be a small number of important keywords. That study used the Genatic Algorithm (GA), which is widely used to solve complex optimization problems in various research domains, including SE.

This paper depends on the ManQ approach [15] as a base work, and it seeks to evaluate the performance of ManQ while reducing the 15 objective functions of ManQ that meet Rahman, et al. [11] study.

2. METHOD

In this paper, ManQ approach is employed as a base work. This paper seeks to evaluate the performance of ManQ while reducing its 15 objective functions supported by Rahman, et al. [11] study and excluding other objective functions that were not supported.

2.1 Many-objective optimization-based automatic query reduction (ManQ):

ManQ studied a set of IRBL specialized AQR approaches in order to find a high-quality subquery. It combines 15 optimization objective functions into one approach to achieve all objectives without neglecting any of them, which could negatively affect the outcome. It applies preprocessing to the bug report (stop word removal and splitting of dotted terms and camel case terms); then a set of terms is produced from the initial query and encoded as binary genes [15]. As shown in Figure 2, ManQ goes through the following steps:

2.1.1 First step: calculating the Objective functions

It consists of 15 objectives that aim to improve query quality by combining the individual objectives of some studies. The objective functions are divided into four groups, as shown in Figure 2. The first group is QQP, which is a set of pre-retrieval query properties measures that are used in IR to evaluate query performance and are computed before executing the query [16, 20]. These QQP are specificity, coherency, similarity, and term-relatedness. They form eight objective functions ($f_1(Q) - f_8(Q)$). The second group maintains important keywords. It is a set of measures that include 1. determining whether the terms of a query are in the sentence describing Observed and Expected Behaviour (OEB), 2. determining whether the query contains the four keywords (words in the first, second, penultimate, or last position) of the bug report summary and the name of the source file; and 3. determining whether terms in the query are grammatically significant based on part-of-speech priority (POS). This group formed four objective functions (f_9 (Q)- f_12 (Q)). The third group is maintaining initial information, which is a set of measures that preserve the context of the sentences in the bug report based on the occurrence of the term using the PMI scale and compute the cosine similarity between subquery Q and the original query. This group formed two objective functions (f_13 (Q)and f_14 (Q)). The fourth group is minimize the query length by compare the results length of a sub query with the initial query

to evaluate how short the sub query is. It forms one objective functionf_15 (Q). All objective functions are illustrated in Figure 2.

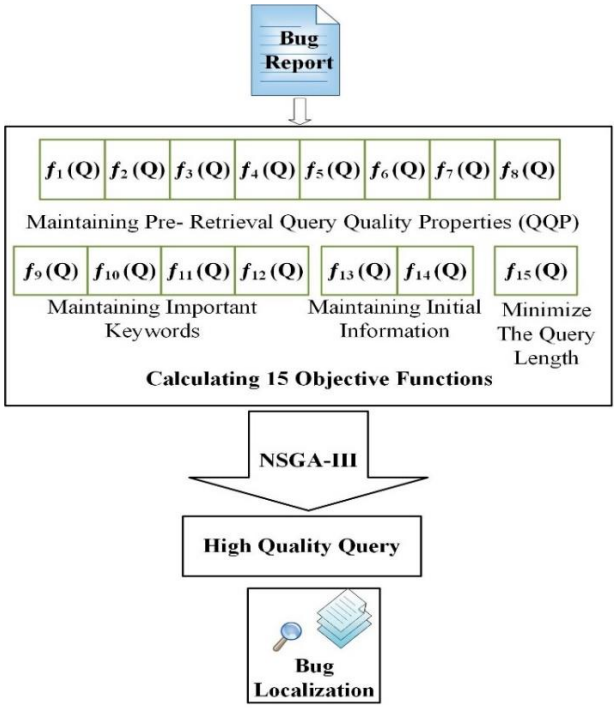

Figure. 2 An illustrated figure of ManQ Approach

2.1.2 The second step: NSGA-III and final query selection:

In this step, after NSGA III is implemented, multiple solutions are resulted where ManQ chooses a final query through the union of selective queries from each function according to next equation:

$$Q = \text{EuclideanDistance}(Q*, f) \cup \text{RankSum}(Q*, f), \quad (1)$$

$$\text{where} \quad f_i = \begin{cases} f_i = \frac{1}{|Q|} \cdot f_i & , \ if\ i<7 \\ f_i = f_i & , otherwise \end{cases} \quad (2)$$

$$\text{EuclideanDistance}(Q*, f) = \text{argmax}_Q \sqrt{\sum_{f_i \in f} f_i(Q)^2} \quad (3)$$

$$\text{RankSum}(Q*, f) = \text{argmin}_Q \sum_{f_i \in f} \text{rank}(f_i(Q)) \quad (4)$$

2.1.3 The third step: Bug Localization:

After obtaining the optimal query, the next step is to submit the query to a popular search engine for documents called Lucene in order to find the buggy files in the source code [5, 15, 16, 18].

2.2 Proposed Work: R-ManQ

As mentioned in the Rahman, et al. [11] study, it concluded there are several attributes that make high-quality queries, and the performance of IRBL approaches is highly efficient. They are: keywords of that query are less frequent within a bug report, are less ambiguous (i.e., have less entropy), are more likely to be found in the description section of a bug report, are more likely to be nouns, and the optimal candidates are likely to be a small number of important keywords. Therefore, R-ManQ aimed to reduce the objective functions that make up the ManQ by adopting the objective functions supported by Rahman's study [11] and dispensing with the rest of the other objectives. Therefore, these attributes were represented in seven objective functions and were kept within the ManQ approach while dispensing with the rest of the other objective functions, and they are:

$f_4(Q)$and $f_5(Q)$to fulfill the keywords of the optimal query are less frequent within a bug report.

$f_6(Q)$to fulfill the keywords of the optimal query are less ambiguous (i.e., have less entropy).

$f_9(Q)$ and $f_{10}(Q)$to fulfill the keywords of the optimal query are more likely to be found in the description section of a bug report.

$f_{12}(Q)$to fulfill the keywords of the optimal query are more likely to be nouns.

$f_{15}(Q)$to fulfill the optimal candidates are likely to be a small number of important keywords.

As shown in Figure 3, the blue rectangles are represented the remaining objective functions in ManQ, only seven objective functions, whereas the rest objective functions are have been removed from the original ManQ and therefore create R-ManQ.

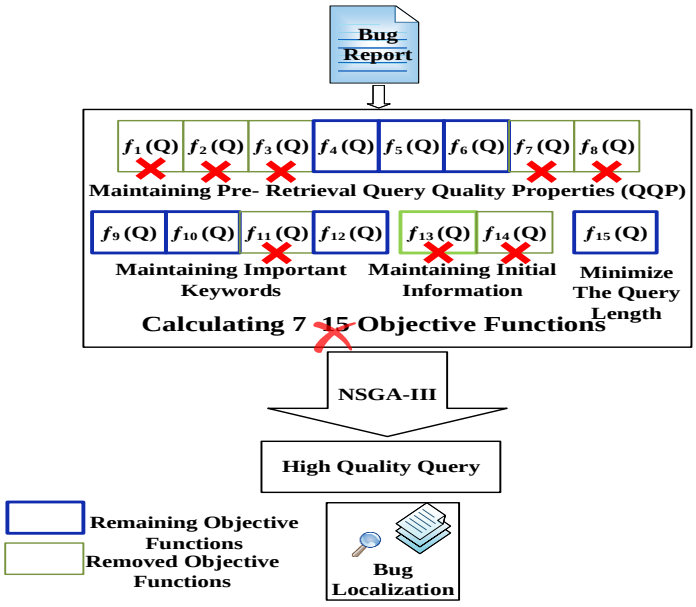

Figure. 3 proposed work R-ManQ

3. RESULTS AND DISCUSSION

This section discusses the evaluation metrics, used data set, and the obtained results from the proposed work and compares them with the base work ManQ, as well as the previous studies identified in [15] , which are, BLIZZARD, STRICT, and an initial query (INIT).

3.1. Evaluation Metrics:

The proposed work, R-ManQ, has been evaluated by a set of evaluation metrics in order to validate its effectiveness and efficiency. They are:

3.1.1 Performance Evaluation Metrics:

To evaluate the effectiveness of the proposed work, performance measures are to be used, which are used in most of the related work to evaluate the performance of IRBL. These metrics are the Top-N (N = 1, 5, 10), mean average precision at 10 ranks (MAP@10), and mean reciprocal rank at 10 ranks (MRR@10).

TOP N Rank: It returns the percentage of queries that contains at least one relevant file (buggy files) in the first N files of the resulted list. The values of N (1,5,10) will be used in this work [3, 5, 6, 12, 14, 15, 18, 24].

Mean Average Precision@10 (MAP@10) calculates the mean of the average precision of all queries, whereas Precision @ 10 calculates the precision when each related result occurs in the sorted list (in 10 ranks) [3, 5, 6, 12, 13, 15, 18, 24].

$$\text{MAP@10} = \frac{\sum_{i=1}^{10} AP_i}{N}, \text{ where } \quad \text{AP@K} = \sum_{k=1}^{D} \frac{P_k \times \text{buggy(k)}}{|S|} \tag{5}$$

Mean Reciprocal Rank@10 (MRR@10) is the mean of the reciprocal of the position of the first buggy file within the top-10 results. The reciprocal rank for a query is the inverse rank of the first relevant document found. This metric evaluates how quickly the developer finds the first buggy files [3, 5, 6, 12, 13, 15, 18, 24].

$$\text{MRR@10} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank(q)}} \tag{6}$$

3.1.2 Execution Time:

It means how long the approach took to finish its work and produce its results. it calculates the total number of the number of system clock seconds that it took.

3.1.3 Query performance changed (|Qc|):

It computed the number of queries whose performance changed (|Qc|), improved ($|Q+|$) and worsened ($|Q-|$) according to the highest rank of the buggy files [11, 15].

3.1.4 The average length of the queries ($|Q|$):

It computed the average length of the queries ($|Q|$) according to the number of terms in the final query of the approach [15].

3.2. Data set:

The used data set is the ManQ data set itself, which consists of 1546 poor queries from six open-source Java-based subject systems from two popular bug tracking systems—BugZilla and JIRA. Poor queries do not include stack traces or program entity queries [15].

3.3. Experiment

3.3.1 Experimental settings

This study used the Eclipse IDE tool to create R-ManQ, which was written in pure Java, and also used it to obtain the results. It was implemented in an AMD Ryzen 5 processor with 2.10 GHz and a RAM of 8.00 GB.

3.3.2 Experimental results and discussion

R-ManQ has reduced the objective functions that make up ManQ by adopting the objective functions supported by the empirical study [11] and dispensing with the rest of the other objectives. This section discusses the results obtained from the proposed work compared with the original ManQ, INIT, STRICT, and BLIZZARD. Table 1 shows the results of this comparison.

Tabel 1 R-ManQ, ManQ, STRICT, INIT, BLIZZARD results

Evaluation Metrics	R-ManQ	ManQ	STRICT	INIT	BLIZZARD
TOP-1	28.57%	25.71%	22.86%	25.71%	27.62%
TOP-5	42.86%	45.71%	42.86%	42.86%	45.71%
TOP-10	53.33%	56.19%	55.24%	52.38%	58.10%
MAP@10	34 %	33.25%	30.69%	32.05%	34.09%
MRR@10	36%	35.02%	31.42 %	33.49%	35%
|Q+|	33%	33%	33%	___	31.42%
|Q-|	20%	18%	30%	___	26%
|Qc|	53%	51%	63%	___	57.42%
|Q|	20	24	14	29	42
Time Execution	15	38	___	___	___

According to the obtained results, the proposed work, R-ManQ, outperformed the original ManQ in all performance evaluation metrics except for TOP-5 and TOP-10. In addition, R-ManQ outperformed INIT in all performance evaluation metrics and STRICT, except in the TOP-10. Further, R-ManQ outperformed BLIZZARD in all performance evaluation metrics, except in TOP-1 and 10.

R-ManQ achieved 28.57% in TOP-1, higher than 25.71% in ManQ, 22.86% in STRICT, 25.71% in INIT, and 27.62% in BLIZZARD. Furthermore, R-ManQ achieved 42.86% in TOP-5, the same as in STRICT and INIT. R-ManQ achieved 53.33% in the TOP-10, higher than 52.38% in the INIT. However, in the TOP-5, R-ManQ obtained less value than 45.71% in ManQ and 45.71% in BLIZZARD. In addition, R-ManQ obtained (53.33% <56.19%) less than ManQ in the TOP-10, as well as less than 55.24% in STRICT and 58.10% in BLIZZARD. All those comparisons are illustrated in Figure 4.

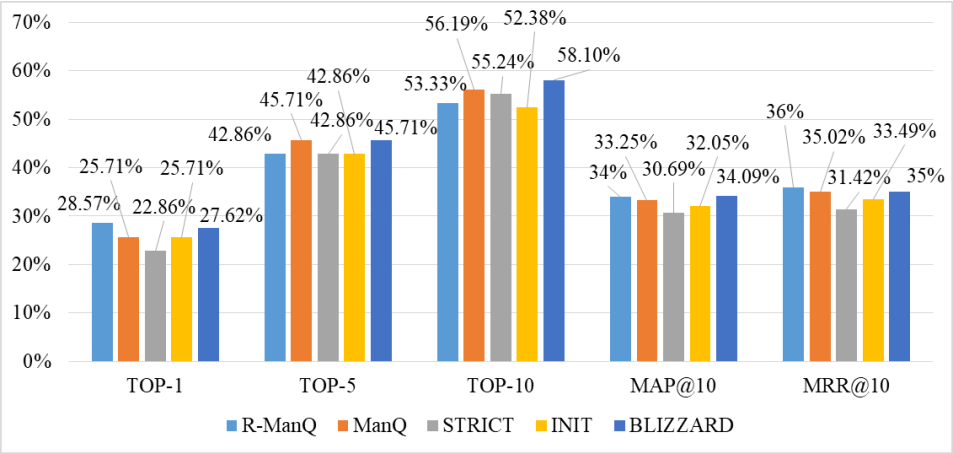

Figure. 4 The performance metrics of R-ManQ, ManQ, STRICT, INIT, BLIZZARD

However, the |*Q*+| overall queries of R-ManQ was the same as it was in ManQ and STRICT (33%), and was higher than BLIZZARD (31.42%). As shown in Table 1, the proposed work obtained better results (20%) in |Q-| than (30%) in STRICT and (26%) in BLIZZARD, while it was lower compared to ManQ (20% > 18%). Figure 5 illustrated |Qc| metrics values.

According to the number of terms, |Q|, R-ManQ decreased it to 20 compared to ManQ, INIT, and BLIZZARD (24, 29, and 42, respectively), while it was 14 in STRICT. Regarding time execution, ManQ took too long time more than the proposed work (38 > 15). Figure 6 illustrates the |Q| and execution time metrics of the proposed work compared to ManQ.

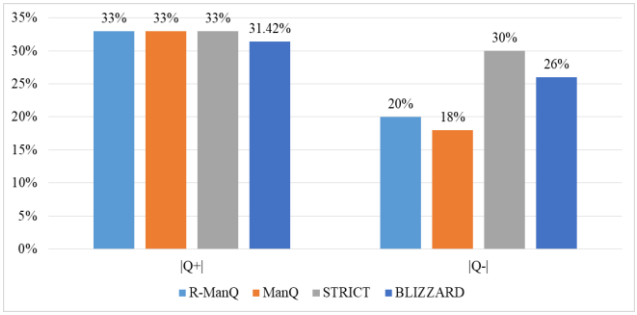

Figure. 5 |Qc| metrics R-ManQ, ManQ, STRICT, BLIZZARD

Despite reducing the number of objective functions that make up ManQ approach, the obtained results of R-ManQ were not low compared to the original ManQ, On the contrary, the proposed work has produced better results than ManQ in the three performance evaluation metrics (TOP-1, MAR@10, and MRR@10) with much less execution time than ManQ. The reason for this reduction in time is that the required processing time is reduced due to reducing the number of objective functions. However, the effect of reducing these objective functions appeared in the values of the TOP-5 and TOP-10, where their values were lower than in ManQ, which indicates that the objective functions that were excluded had an impact on the results obtained by the ManQ approach.

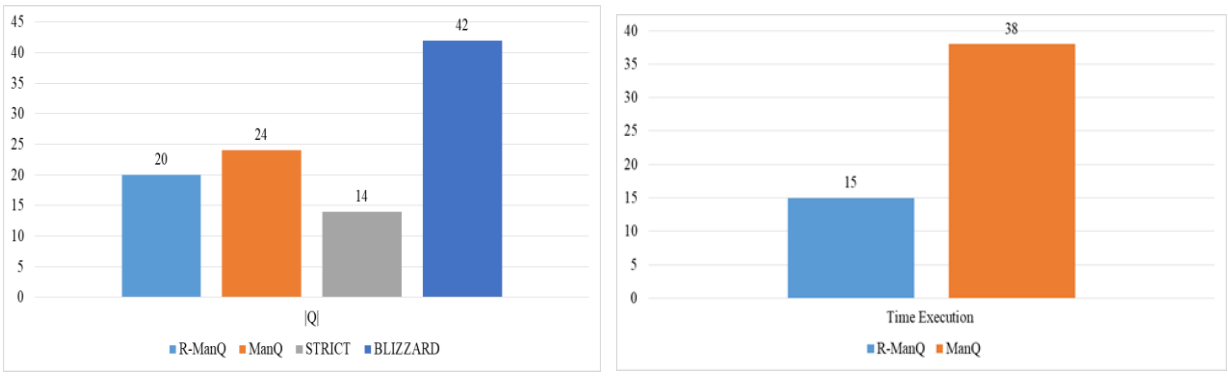

Figure. 6 Execution Time and |Q| metrics R-ManQ, ManQ, STRICT, INIT, BLIZZARD

4. CONCLUSION

Although many IRBL approaches were provided in order to make the bug localization process faster and with less effort, their performance is still not efficient. ManQ is one IRBL approach that improves the performance of IRBL by enhancing the quality of queries through a multi-objective optimization approach. Although ManQ consists of fifteen objective functions, [11] study claimed there are several attributes that make high-quality queries and the performance of IRBL approaches highly efficient. Therefore, R-ManQ has been proposed. It was evident that a similar performance could be achieved by R-ManQ as ManQ with a much lower execution time, a much lower number of objective functions (only seven objective functions), and a smaller number of terms in the query, where it obtained high results on TOP-1, MAP@10, and MRR@10 while failing on TOP-5 and TOP-10. Therefore, in the future, new objective functions related to graph-based term weighting algorithms will be added within the components of the proposed work R-ManQ as a step toward improving its performance. It was concluded that the attributes identified in [11] are the most important attributes that make the query high quality, which leads to an improvement in the process of locating the buggy files that contain the IRBL.

REFERENCES

[1] A. Goyal and N. Sardana, "Performance Assessment of Bug Fixing Process in Open Source Repositories," *Procedia Computer Science*, vol. 167, pp. 2070-2079, 2020.

[2] H. Mohsin and C. Shi, "SPBC: A self-paced learning model for bug classification from historical repositories of open-source software," *Expert Systems with Applications*, vol. 167, p. 113808, 2021.

[3] R. Almhana, M. Kessentini, and W. Mkaouer, "Method-level bug localization using hybrid multi-objective search," *Information and Software Technology*, vol. 131, p. 106474, 2021.

[4] K. Kevic and T. Fritz, "Automatic search term identification for change tasks," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 468-471.

[5] M. M. Rahman and C. K. Roy, "STRICT: Information retrieval based search term identification for concept location," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 79-90.

[6] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Information and Software Technology,* vol. 82, pp. 177-192, 2017.
[7] M. Erşahin, S. Utku, and D. Kilinç, "Bug Localization by Using Information Retrieval and Machine Learning Algorithms," *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18),,* 2018.
[8] A. Sood, S. Sharma, A. Khanna, A. Tiwari, D. Gupta, V. Madan, and S. Doss, "Bug Localization Using Multi-objective Approach and Information Retrieval," in *International Conference on Innovative Computing and Communications*, 2021, pp. 709-723.
[9] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 127-136.
[10] M. Ficco, R. Pietrantuono, and S. Russo, "Bug localization in test-driven development," *Advances in Software Engineering,* vol. 2011, 2011.
[11] M. M. Rahman, F. Khomh, S. Yeasmin, and C. K. Roy, "The forgotten role of search queries in IR-based bug localization: an empirical study," *Empirical Software Engineering,* vol. 26, pp. 1-56, 2021.
[12] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 345-355.
[13] O. Chaparro, J. M. Florez, and A. Marcus, "Using bug descriptions to reformulate queries during text-retrieval-based bug localization," *Empirical Software Engineering,* vol. 24, pp. 2947-3007, 2019.
[14] T. Dao, L. Zhang, and N. Meng, "How Does Execution Information Help with Information-Retrieval Based Bug Localization?," in *International Conference on Program Comprehension (ICPC)*, 2017.
[15] M. Kim and E. Lee, "ManQ: Many-objective optimization-based automatic query reduction for IR-based bug localization," *Information and Software Technology,* vol. 125, p. 106334, 2020.
[16] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. D. Lucia, "Predicting query quality for applications of text retrieval to software engineering tasks," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 26, pp. 1-45, 2017.
[17] S. Henninger, "Using iterative refinement to find reusable software," *IEEE software,* vol. 11, pp. 48-59, 1994.
[18] M. M. Rahman and C. K. Roy, "Improving ir-based bug localization with context-aware query reformulation," in *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 621-632.
[19] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of ir-based fault localization techniques," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 1-11.
[20] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Automatic Query Performance Assessment during the Retrieval of Software Artifacts," in *In Proceedings of the 27th IEEE/ACM international conference on Automated Software Engineering*, 2012, September, pp. 90-99.
[21] G. Kumaran and J. Allan, "Effective and efficient user interaction for long queries," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 11-18.
[22] G. Kumaran and V. R. Carvalho, "Reducing long queries using query quality predictors," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, 2009, pp. 564-571.
[23] M. Ruchika, S. Aggarwal, R. Girdhar, and R. Chugh, "Bug Localization in Software using NSGA-II," *IEEE,* 2018.
[24] G. Liu, Y. Lu, K. Shi, J. Chang, and X. Wei, "Mapping bug reports to relevant source code files based on the vector space model and word embedding," *IEEE Access,* vol. 7, pp. 78870-78881, 2019.