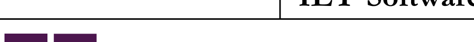
Check for updates

Received: 4 May 2022 | Revised: 3 May 2023 | Accepted: 9 May 2023

IET Software

DOI: 10.1049/sfw2.12127



ORIGINAL RESEARCH

Mining relevant solutions for programming tasks from search engine results

Adriano M. Rocha | Marcelo A. Maia

Faculty of Computer, Federal University of Uberlândia, Uberlândia, Brazil

Correspondence
Adriano M. Rocha, Federal University of Uberlândia, Campus Monte Carmelo, LMG-746 Km1 HW, Monte Carmelo, MG 38.500-000, Brazil.
Email: adriano.rocha@ufu.br

Funding information
CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico; FAPEMIG - Fundação de Amparo a Pesquisa do Estado de Minas Gerais

Abstract

Official documentation of software development technologies, for example, APIs, may not be sufficient for all developer needs, so searching on the Internet is a usual practice. Nonetheless, finding useful information may be challenging because the best solutions are not always among the first ranked pages. Developers need to read and discard irrelevant pages, that is, those without code examples or those that have content with little focus on the desired solution. This work aims at proposing an approach to mine relevant solutions for programming tasks from search engine results by removing irrelevant pages. The authors evaluated the top-20 pages returned by the Google search engine, for 10 different queries, and observed that only 31% of the evaluated pages are relevant to developers. Then, the authors proposed and evaluated three different approaches to mine the relevant pages returned by the search engine. Google's search engine has been used as a baseline, and authors' results have shown that it returns a reasonable number of irrelevant pages for developers, and the authors could establish an effective approach to remove irrelevant pages, suggesting that developers could benefit from a customised web search filter for development content.

KEYWORDS
information retrieval, software engineering, software reusability, source code (software)

1 | INTRODUCTION

Software development is a knowledge-intensive task [1], where the search for information is challenging, due to the large amount of information publicly available [2]. To obtain relevant information related to software development, developers often use the Internet, which offers a vast variety of information sources, such as, tutorials, blogs, question and answer (Q&A) services. Paradoxically, the overload of available content burdens developers, who must search, screen and analyse content from several web pages until they find those that are really relevant for the particular task [3, 4]. Among the various sources of technical content, Stack Overflow is a Q&A service specific for software developers, with a large volume of content, that also challenges the search for relevant content [5, 6].

Nonetheless, Stack Overflow does not cover all the method calls of a chosen programming technology, and the best solutions for a programming task may also not be always available on Stack Overflow [7]. In spite of that, several works have considered only Stack Overflow for searching for how to answers with code. So, mining from the "raw Web" can be a much more fruitful approach, but on the other hand, it is a different and more challenging problem because the content lacks the structure of Q&A, and most importantly, lacks the assessment by the crowd.

Search engines are one the most used services on the Internet [8]. So, developers also use general purpose search engines, such as Google or Bing, to find technical information for their task in hand [9–11]. The use of search engines to find solutions related to software development has an inherent problem that the best solutions are not necessarily always among the best ranked pages [12, 13]. Problems that typically appear among the first returned pages from a search query are lack of code examples or little relevance for the

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

© 2023 The Authors. *IET Software* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

task in hand, for example, pages with content that is not very focussed on the query performed by the developer, as shown in the results of this work. Another problem shown in the work of Cho and Roy [14] is that search engines can have an extremely worrying impact on the discovery of new web pages. These results show the importance of approaches that search pages for the content and not for its popularity because new pages are not popular, consequently they are in lower positions in the ranking, even with relevant content. In such cases, developers must screen those best ranked pages to find relevant solutions, which is a task that requires time and effort [15]. To overcome these limitations, in this work we propose an approach to mine relevant solutions for programming tasks from search engine results for a query target at a specific task. The proposed approach is aimed at filtering pages considering the content relevance and not its popularity. Figure 1 presents an overview of the approach. First, a query related to the programming task is elaborated and given as input to a search engine (Google is used as choice service). The n returned pages returned are automatically filtered in the potential relevant content. We consider the relevant solutions those that have code examples and focus on the specific searched problem because developers look for solutions that contain code examples for programming tasks consulted in search engines. Relevant solutions for developers should be focussed on the queried programming task, since solutions with little focus may not satisfy the developer due to the lack of content that solves the programming task or the excess of content that is not related to it, that is, content that implements other functionalities that do not belong to the programming task [12, 13].

We formulate the following research questions to evaluate the results.

RQ1: To what extent does Google's search engine return relevant pages to software developers?
RQ2: To what extent do filters to remove outlier pages w.r.t. Page size help to remove irrelevant pages returned by search engines?
RQ3: To what extent applying clustering algorithms w.r.t. Total method calls help to remove irrelevant pages returned by search engines?
RQ4: To what extent applying clustering algorithms w.r.t. Unique occurrences of method calls help to remove irrelevant pages returned by search engines?
RQ5: To what extent applying the filters proposed in this work improve the ranking quality of pages returned by search engines?

This article is organised as follows. Section 2 presents the background. Section 3 presents the proposed approach for mining relevant solutions for programming tasks from search engine results. Section 4 presents the study setting. Section 5 presents the results of this work. Section 6 presents the discussion of the results. Section 7 presents the threats to validity. Section 8 presents related work. Section 9 presents the conclusion and future work.

2 | MOTIVATING EXAMPLE

We present a motivating example to illustrate the problem of low quality content web searching for development solutions.

Consider a scenario where a developer wants to learn how to save data from an Android application to a web server, using the JAVA programming language and the Android API. The developer would use, for instance, the following search query, which seems to be simple and with sufficient details: "*how to* save *data from application to web server Java Android*". Inspecting the results, we observed that the top-1 page has no source code, but only a question posted by a user on how to save data from an Android application to a remote server. So after reading this page, the developer would need to check the next ones. In the top-2, there is also no source code but only a text with an overview of the data and file storage on Android. The top-3 presents an explanation and source code of how to save data locally in an Android application. However, page content is not focussed on the desired solution, and the developer would still need to continue reading the next pages. The top-4 page provides content related to how to store data in Google Drive. Note that the content on this page lacks focus on what the developer searched for. The top-5 page provides explanation and source code on how to download files in Android. Again, the content in this page lacks focus on the query entered by the developer. In the top-6 page, there is no source code, just an overview of the FCM (Firebase Cloud Messaging) architecture, which is a technology that offers messaging capabilities for different platforms. The top-7 page is a website of a software development company, where there is no content related to the search query, only the company's portfolio. The top-8 page provides a solution on how to save

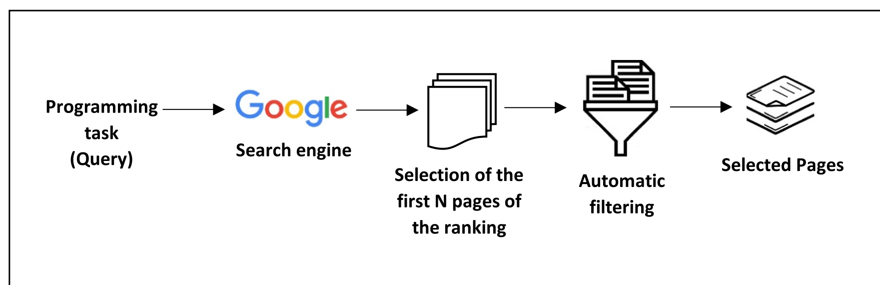


FIGURE 1 Main steps of the proposed approach.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

data on the remote server, but the solution lacks focus, as the solution also explores how to receive notifications when the stored data are changed, which was not the original goal in the query. The top-9 presents a solution on how to store data on an SD card, which is again a solution lacking focus on the intent. The top-10 provides a solution explaining how to save data to a file on Android, thus lacking focus on the intent. Finally, only in the top-11, the developer finds a solution with a high degree of focus, illustrating a scenario where the developer may need to read and analyse the content of several pages that are not related to the performed query, thus wasting time and effort.

Regarding this motivating example, one would question if the ranked results are not satisfactory because of an ill-formulated query. Although, we agree that a different query would produce a different ranking, the query proposed in the example seems to be a possible one that developers would launch. Moreover, small changes in the query formulated by the developer do not affect much the results returned by the search engine [9]. Alternative approaches for automatic reformulating queries may produce a list of pages with more adequate semantic matching with the query [16] but still does not address some issues such as lack of focus in returned pages. So, we envision that approaches that mine for relevant content in whatever list of returned pages, independently of the query quality, may play an important role for improving the developer performance.

3 | THE MINING APPROACH

We propose an approach to tackle the challenges related to filtering relevant content obtained by querying search engines, more specifically, removing top-ranked pages with undesirable features for developers, such as lack of code examples and little focus.

The input for the mining approach is a query denoting a programming-related task. This query is performed on a search engine (for example, Google), and the top-*n* pages are selected for an automatic filtering process to obtain pages with relevant solutions.

In this section, we present the steps of the proposed approach and discuss the components implemented in each one.

3.1 | Mining steps

The main steps of the approach are as follows:

1. Preparation of a query related to the programming task, for example, "how to make login with php mysql". The queries are systematically created as "*how to*" + *programming task* + *programming language* + *API*.
2. Querying the Google search engine taking the above query as input.
3. Selection of top-*n* pages returned by the search engine that have the following characteristics: textual content must be in English and must not contain only PDF, PowerPoint, Excel and Word files. To filter the content in English, the approach uses the search engine in the English version and the proposed query was formulated in English. The links returned by the search engine ending with the above file extensions are removed. To obtain the pages for each proposed approach, we obtain the links of the pages returned by the search engine manually using an anonymous non-logged browser interface, that is, we enter the query in the Google search engine and copy the returned links manually, then give these links as input to the approach.
4. Automatic filtering of the *n* pages selected in the previous step in order to 1) remove pages without source code with method calls; 2) remove outlier pages regarding to the size of the solution, that is, those that have few occurrences of method calls or a large number of occurrences of them in relation to the upper and lower limits based on the average occurrence of method calls in all the *n* pages; 3) remove pages with low focus by applying a clustering algorithm to obtain pages with solutions with more methods in common regarding all queries. The filtering mechanism is detailed in the next subsection.

3.2 | Parameter calibration

In order to define the filtering mechanisms for the mining approach, we need also to define some parameters related to criteria to evaluate the pages returned by the search engine. We will rely on a bootstrap approach to define such parameters, that is, we test the approach with an arbitrary parameter and then evaluate the pages to define if those pages adhere to the criteria, and then if the pages should or not be filtered out from the search engine results. We run this process iteratively changing the respective parameter until we find a sub-optimal value. For this calibration, we collect the search results of three queries using the chosen parameter (*01 - how to create table java swing*, *02 - how to create simple calculator in android and 03 - how to implement login php*). The top-20 ranked pages were selected for each query. Then, we manually evaluate such pages and calculate the *F-measure* to find which parameter should be chosen according the respective best *F-measure*. The manual evaluation criteria are defined in Section 4.3, which are the same used to define the ground-truth for evaluating the proposed approach. It should be noted that the developers will not need to configure the parameters for the filters of the proposed tool, they will only need to enter the query and the approach will return the relevant pages.

3.3 | Lower and upper limits for outlier pages

We hypothesize that the number of occurrences of method calls in a page should not be too small because of the possible lack of examples to explain a programming solution. Moreover

that number of occurrences of method calls should also not be too large because of the possible complexity of the solution that would hinder the developer comprehension. In order to define such lower and upper limits for this number in a page, we have used the calibration bootstrap method defined in the previous subsection. These limits will be used in the outlier filter to be presented in Subsection 3.4.

The lower limit is calculated using the expression:

$$lowerLim = \frac{averageOccurrenceMethodCalls}{8} \quad (1)$$

The upper limit is calculated using the expression:

$$upperLim = averageOccurrenceMethodCalls \times 2 \quad (2)$$

For each query, we analysed the 20 pages using the Degree of Focus criterion defined in Subsection 4.3.2. The pages evaluated with the values "High" or "Very High" were classified as "*Selected*" and the pages evaluated with the values "Low", "Very Low" and "Neutral" were classified as "*Not Selected*". For each query, considering the average total occurrence of the method calls present in the 20 pages, the divisor values of the expression lower limit varied from 2 to 10, by 0.5, and the multiplication factor of the upper limit also was varied from 1.5 to 4, by 0.5 in 0.5 Then, for each query, the respective results were manually assessed, and the metrics *precision*, *recall* and *F-Measure* were calculated. We selected the respective divisor and the multiplication factor that resulted in the highest F-Measure, that is, 8 and 2, respectively.

3.4 | Page filtering mechanism

Given a set of n web pages as input, filtering is performed through the pre-processing steps, removing outlier pages and selecting pages with common method calls, which are explained below.

Pre-processing. The first pass in this step is to obtain the method calls present in the source code of each of the n web pages given as input, using a simple Java regular expression: .*\(.*\). For example, the following Java Swing API methods are recognised by the regular expression: *getFirstRow()*, *getColumnName(column)*, *getValueAt(row, column)* and *setPreferredWidth(100)*. A filter was implemented to remove the method calls found within page scripts and style tags. We implemented 15 rules, considering the opening and closing of script and style tags. For example, when processing the page's source code, if a script opening tag is found, then the code following that tag is disregarded until the corresponding closing tag is found. For the sake of conciseness, we omit the complete rules in this paper. Nonetheless, the details of the created rules are publicly available.[1] We used regular expression to recognise method calls in the pages because it is a simple approach and can recognise method calls from many different programming languages. There are other compiler-based technologies such as Eclipse JDT that require the snippets to be compileable. As the examples included in the pages usually do not meet this requirement, we opted for not choosing this approach. Moreover, we did not choose Eclipse JDT to get the method calls because this API is specialised for the Java programming language. The purpose of the proposed approach is to work with several programming languages. We carried out tests with several programming languages other than Java and observed the effectiveness of the recognition of method calls. During the preliminary tests, some extracted method calls still had attached unwanted characters or words. To address this problem a filter was created to remove method calls that contain any of the following words or characters: function, -webkit, jQuery, gt;alert, {, }, (,), ‘, ˆ, ’,/and \. Method calls containing only one character from the alphabet have also been removed, because generally such method calls are user-defined. For example, the $x()$ method, when obtaining the method name, results in x.

We performed tests to check the effectiveness of regular expression to find method calls. We took the first 5 programming tasks shown in Table 1 and then gave them as input to the Google search engine. For each programming task, we obtained the method calls from the first 3 pages returned from the search engine, manually. Then we calculate the metrics Precision, Recall and F-measure of the algorithm. The results show that the 15 pages analysed have a median of method calls of 45, an average of 52.9, a maximum of 104, a minimum of 0 and a standard value of 29.8. Of the 15 pages, 12 had a value of 1 for precision and recall, the other pages had precision and recall, varying from 0.933 to 0.981, and from 0.959 to 0.971, respectively, demonstrating the effectiveness of the algorithm to find method calls.

Removing outlier pages regarding number of method calls. This step has as input the number of method calls returned in the previous pre-processing step. Given the number of occurrences of method calls for each page, the average of occurrences for n pages is calculated. Pages that have few method calls or a large number of method calls in relation to the average occurrence of method calls are removed. The lower and upper limits were defined in Subsection 3.3. Pages that have the number of occurrence of method calls greater than the lower limit and less than the upper limit are selected in this step. Generally, pages with few method calls in their solutions tend to not have code examples that solve the problem queried by the developer, and pages with a large number of method calls tend to have complex and extensive code examples.

Selection of pages sharing common method calls. We hypothesise that different pages that share common method calls are similar pages, and thus possibly with popular content that is frequently used by the developer community. So, we hypothesise that pages sharing common method calls are an objective proxy for pages that have high degree of focus on the user query.

[1] https://doi.org/10.5281/zenodo.6467629

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 Selected APIs, application domains and queries.

APIs	Domains	Queries
Spring	Web Development	How to upload image java spring
JPA	Persistence	How to implement crud operations java jpa
Selenium	Web testing	How to search a product item on an e-commerce Web application java selenium
JavaFX	Desktop UI	How to implement menu java javafx
Tensorflow	Machine learning	How to build a neural network java tensorflow
OpenCV	Computer vision	How to classify image java opencv
JUnit	Unit testing	How to implement matchers test java junit
Jackson	Structured Data proc.	How to process JSON data java Jackson
LibGDX	Game engine	How to implement animation java libgdx
OpenGL	Computer graphics	How to draw 2D objects java opengl

So, in order to capture those pages, we propose to apply a clustering algorithm using a proxy metric based on the sharing of common method calls between the pages. The purpose of this algorithm is to aggregate pages that share the same method calls in the same cluster. So, our hypothesis is that clusters that have pages with similar solutions are more focussed on user query. The instances given as input for this algorithm are the pages, the attributes are the method calls, and the attribute values are the number of occurrences of the respective method calls in the instance (page). We have two options for calculating the number of occurrences of method calls: 1) summing the total number of occurrences of a method call in a page; 2) returning one if the method call occurs and zero if the method does not occur. We will evaluate both approaches of clustering.

The number of clusters k generated by the algorithm needs to be defined as input, which we define as $k = numberOfInstances/d$. We calibrate the approach calculating a sub-optimal value for d, varying its value from 1.5 to 4 (0.5 in 0.5). We use the same queries defined in Section 3.2 and collected the respective result pages and apply the clustering algorithm with the above different number of clusters k. For each of them, we bootstrap the filtering of clusters with solutions more focussed on user's query and calculate the metrics Precision, Recall and F-Measure. Then, best F-Measure was obtained for $d = 2$. We chose the k-means clustering algorithm as it is one of the most used algorithms, guarantees convergence, adapts into different contexts. Because k-means are influenced by outliers, we mitigate this problem by applying the outlier removal filter mentioned in the previous paragraph before applying the clustering algorithm. We emphasise that to use the proposed filters, developers do not need to configure parameters, that is, the developer will only need to enter the query and the approach will return the relevant pages. The parameters of the filters were already defined with adjusted values.

During the calibration tests of the approach parameters shown in Section 3.2, we observed that pages containing a wide variety of method calls, result in a large vector of attributes, which can impair the quality of the generated clusters. To work around this problem, before executing the clustering algorithm, an attribute selection algorithm is first applied to obtain only relevant attributes. After the attribute selection, the vector of attributes is reduced and clusters of better quality can be generated.

To select the cluster that contains more pages with common method calls, we calculated for each cluster the number of unique occurrences of method calls existent in the different pages of the cluster. The cluster with the highest value is selected.

Next, we show an example of the clustering selection procedure. Assume the user enters the query "*how to create table java swing*" as input to the approach. To make the example simpler let us consider only the first 7 pages returned by the search engine. Table 2 shows the pages and method calls within them. Method calls that are present on at least two pages in the same cluster are highlighted in bold. We can observe that Cluster 2 has pages with solutions more focussed on the user's query, since the pages have basic method calls for creating a table using JAVA and the Swing API. Cluster 1, on the other hand, has pages with solutions that have lost their focus a little, as there are methods that could be omitted when creating a table. For example, the BorderLayout () method is related to table layout. Cluster 1 also has other API (AWT) method calls that could be replaced by Swing API methods, for example, the GridLayout (), WindowAdapter () and FlowLayout () methods. The pages present in Cluster 3 also have low focus. For example, Page 2 has methods related to database manipulation, which indicates that the solution, in addition to creating a table, the solution also addresses database-related issues that are not tied to the user's query. Page 6, on the other hand, is related to the creation of a table that can be edited by the user, in addition to having other components such as JComboBox that is not related to the user query. Table 3 shows the clusters with their respective method calls that occur on at least two pages and the sum of those occurrences, showing that Cluster two is selected as it has the largest sum of occurrences.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 Example of application of the clustering algorithm on the pages returned by the search engine.

Pages	Method calls on the page	Cluster
Page 1	**JFrame( )**, setSize(), **setLayout( )**, GridLayout(), addWindowListener(), WindowAdapter(), windowClosing(), JLabel(), **JPanel( )**, FlowLayout(), **add( )**, setVisible(), setText(), **JTable ( )**, **JScrollPane( )**	Cluster 1
Page 2	super(), forName(), getConnection(), createStatement(), executeQuery(), next(), getString(), println (), **JTable( )**, **setSize( )**, **setVisible( )**	Cluster 3
Page 3	**JFrame( )**, **JTable( )**, setBounds(), **JScrollPane( )**, **add( )**, **setSize( )**, **setVisible( )**	Cluster 2
Page 4	**JTable( )**, setFillsViewportHeight(), **JScrollPane( )**, setPreferredSize(), Dimension(), **setLayout( )**, BorderLayout(), **add( )**, **JPanel( )**, setOpaque(), **JFrame( )**, setDefaultCloseOperation(), setContentPane(), **setVisible( )**	Cluster 1
Page 5	**JFrame( )**, **setTitle( )**, **JTable( )**, **JScrollPane( )**, **add( )**, **setSize( )**, **setVisible( )**	Cluster 2
Page 6	JFrame(), Integer(), Boolean(), EditableTableModel(), **JTable( )**, createDefaultColumnsFromModel(), JComboBox(), getColumnModel(), getColumn(), setCellEditor(), DefaultCellEditor(), add(), JScrollPane(), **setSize( )**, **setVisible( )**, getRowCount(), getColumnCount(), getValueAt(), getColumnName(), getColumnClass(), getClass(), isCellEditable(), setValueAt()	Cluster 3
Page 7	**JTable( )**, **add( )**, **JScrollPane( )**, **setTitle( )**, setDefaultCloseOperation (), pack (), **setVisible ( )**, invokeLater (), Runnable (), run ()	Cluster 2

Note: Method calls that are present on at least two pages in the same cluster are highlighted in bold.

TABLE 3 Clusters with their respective method calls that occur on at least two pages and the sum of those occurrences.

Cluster	Method calls	Occ. > 1	Sum of occ.
Cluster 1	JFrame()	2	14
	setLayout()	2	
	JPanel()	2	
	add()	2	
	setVisible()	2	
	JTable()	2	
	JScrollPane()	2	
Cluster 2	JTable()	3	
	JScrollPane()	3	
	add()	3	
	setVisible()	3	**19**
	setSize()	3	
	JFrame()	2	
	setTitle()	2	
Cluster 3	JTable()	2	
	setSize()	2	6
	setVisible()	2	

Note: Bold indicates the highest value obtained in the Sum of occ. column.

4 | STUDY SETTING

4.1 | Definition of baselines

The evaluation of the previous mining mechanisms will be conducted by comparing its results with results obtained from

TABLE 4 Main filtering features of the approaches.

Acronym	Features
JG	Just Google
GOR	Google + Outliers Removal
GORCTO	Google + Outliers Removal + Clustering + Total Occurrences of Method Calls
GORCUO	Google + Outliers Removal + Clustering + Unique Occurrences of Method Calls

variations (baselines) built from queries results from the Google search engine. The main characteristics of the proposed baselines are shown in Table 4.

The JG (Just Google) approach is just a raw Google search, getting the first returned pages. The GOR (Google + Outlier Removal) approach consists of applying the JG approach and then executing the filter for removing outliers pages explained in Subsection 3.4. The GORCTO (Google + Outlier Removal + Clustering + Total Occurrence of Method Calls) approach consists of applying the GOR approach and then executing the clustering algorithm, where the attribute values are formed by the total sum of occurrences of each method call on the processed page (instance), for example, if the PHP method *isset* occurs 4 times on page X, then the value of the attribute *isset* for page X will be 4. The GORCUO (Google + Outlier Removal + Clustering + Unique Occurrence of Method Calls) approach consists of applying the GOR approach and then executing the clustering algorithm, where the values of the attributes are formed by the presence of occurrence of method calls on the

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

page, that is, if a given method call occurs at least once on the page, the value of its attribute will be 1, otherwise 0.

To evaluate the four baselines, we collect their results on the n pages returned from the queries defined in the next subsection. For each approach, precision, recall and F-Measure are calculated and compared.

Precision, recall and F-Measure are calculated according to the previously constructed ground-truth. For each query, we manually construct the ground-truth assessing the proposed three criteria (presence of code, degree of focus, and size of the solution) for evaluating the quality of the page (hit). After running the approach for each query, we evaluate the returned pages against the manually constructed ground-truth, mark which pages are hit, and then we can calculated precision, recall and F-measure according the respective definition of those metrics. Pages without code or containing irrelevant code are not hit in the ground-truth, so if the approach would return them, it is false positive, and if the approach does not return them, they are cases of true negative.

4.2 | Definition of queries for assessment

To compare the baselines, the first step is to generate queries related to the programming task as follows.

1. We analysed a Stack Overflow survey[2] to choose a popular programming language, and JAVA was the choice.
2. In order to choose the APIs that are used in the programming tasks, we adopted the criterion of choosing the most popular APIs (number of questions) of the JAVA language in Stack Overflow.[3] We checked the JAVA language API tags on Stack Overflow and ranked them by popularity. The APIs were categorised by application domain and an API was selected from each domain. When there were several APIs in each domain, we chose the most modern ones.
3. To select the programming tasks, we adopt the following criteria. For each API selected in the previous step, in the Coursera,[4] a query was made with the name of the language (JAVA) + the name of the API. For each consultation, we analysed the course summaries and selected the first programming task found that manages a product or a semi-product. Coursera was chosen because it is the largest online course platform on the Internet, with respect to the number of users [17].
4. The queries to be executed in the search engine were built as follows: "how to" + programming task + programming language + API. We add the prefix "how to" before the programming task in order to find the pages that are related to how to implement the programming task consulted, unlike pages of the type of debug-corrective. Table 1 shows the defined queries.

[2]https://insights.stackoverflow.com/survey/2020#most-popular-technologies
[3]https://stackoverflow.com/tags
[4]https://www.coursera.org

The queries are executed on the Google search engine and the first n pages are obtained for each query. We process the content of these n pages (for each query) according each baseline procedure, and then evaluate them using a ground-truth defined below. As Google considers the history of previous searches performed by the user as a criterion for selecting the pages returned,[5] we executed the queries using the browser in private mode without a logged account.

4.3 | Ground-truth

This section shows the steps to build the ground-truth. To evaluate the pages returned by search engines, we defined the criteria presented in Subsection 4.3.2, which were based on the quality indicators and dimensions, discussed in the works present in the next subsection.

4.3.1 | Evaluation of the pages returned by the search engine

Arthur and Stevens [18] defined four quality indicators (accuracy, completeness, usability and expandability) for evaluating software documentation. Accuracy is defined as the consistency between the code and all code documentation, for all requirements. A documentation is complete if all the required information is present. Usability is defined as the suitability of the documentation regarding to the ease with which one can extract needed information. Expandability is the capability of the documentation to be modified in reaction to changes in the system. Although, these quality indicators are not completely suitable for evaluating the pages related to software development returned by search engines, as they were created for evaluating software documentation, they may serve as starting point to compose evaluation criteria related to the quality of the pages returned by search engines.

Smart [19] proposed three dimensions (easy to use, easy to understand and easy to find) to assess the quality of the documentation. Easy-to-use dimension is related to the ease of users to complete tasks related to their work through documentation, which must be accurate and include all and only the essential parts. The easy-to-understand dimension is related to the documentation's ability to be unambiguous, but to have appropriate writing styles, using examples and metaphors. The easy-to-find dimension is related to the ability to organise the documentation, where it must be coherent and make sense to the user, the information must be retrieved quickly and must be visually effective.

In order to evaluate the quality of the pages returned by search engines, in the next subsection, we defined three criteria (degree of focus, code examples and solution size) based on the quality indicators of the work of Arthur and Stevens [18] and the dimensions for the evaluation of documentations of

[5]https://www.google.com/search/howsearchworks/how-search-works/ranking-results/

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the work of Smart [19], discussed previously. We consider relevant pages those with solutions that have a high or very high degree of focus, code examples and solution size not far from the average size of the solutions present in the first 20 pages returned by the search engine.

In the proposed approach, to mine relevant pages from search engine results, we capture the degree of focus of the pages through the clustering algorithm, which aims at selecting those pages that have method calls in common with other pages (similar solutions often used by the community). With this algorithm, pages that have partial solutions, which tend to be not-compilable, have a very low probability of being selected, as these will be allocated to other clusters that have pages with unusual method calls between pages. For selecting pages with source code and solution size similar to the average size of solution sizes in the 20 returned pages.

In the next subsections, we present how we defined such thresholds.

4.3.2 | Criteria for ground-truth construction

To assess the pages returned by the search engine, we use the following criteria.

Code examples. The pages have a binary value "*yes*" or "*no*" for this criterion, depending on containing or not code examples in their solutions. In this work, we are focussing on solutions where the developer can reuse source code, since most developers look for code reuse when searching in search engines. So for the purpose of this work, pages that do not have code examples are not interesting for developers who are looking for a solution to a programming task searched on search engines [9].

Degree of focus. It consists of assessing how the solutions are related to the programming task searched by the user.

1. Very low. The solution is not related to the search, that is, it contains implementations of other features that are not semantically related to the search.
2. Low. The solution has some connection with the search performed by the user, that is, it contains implementations of several features that are not related to the search.
3. High. The solution is related to the search; however, it has minor focus deviation, presenting implementations of few other features that are weakly related to the search.
4. Very high. The solution is completely related to the search carried out by the user, that is, the implementation features in the solution teach what the user wants to learn.
5. N/A. Not applicable. When the page has no implementation, evaluating the degree of focus of the implementation is not applicable. Moreover, we consider these pages irrelevant for the user.

This criterion is based on the *completeness* quality indicator of the work of Arthur and Stevens [18], since for having a high degree of focus, a page must have all the information required for the solution of the programming task expressed by the user query. This criterion is also based on the easy-to-use dimension of the work of Smart [19], since for having a high degree of focus, a page must have a precise solution, that is, *all* and *only* the essential parts.

Solution size. To evaluate this criterion, we visually analysed the size of the solutions of the 20 pages returned by the search engine, considering the number of source code lines. After that, according to that general visualisation, we evaluated the solution of each page using the possible values for the criterion.

1. Very small. Far below the observed average size.
2. Small. Marginally below the observed average size.
3. Large. Marginally above the observed average size.
4. Very large. Far above the observed average size.
5. N/A. Not applicable. When the page has no source code, evaluating the size of the code is not applicable. Moreover, we consider these pages irrelevant for the user.

4.3.3 | Ground-truth construction

The pages in the ground-truth are classified as "Selected" or "Not-selected", considering the criteria previously defined and Equation (3), where *ce*, *ss* and *dof* represent the code examples, solution size and degree of focus criteria, respectively.

$$selectedPages \equiv \begin{aligned} & ce = \text{``}yes\text{''} \wedge \\ & \text{dof} = 3 \vee \text{dof} = 4 \wedge \\ & ss = 2 \vee ss = 3 \end{aligned} \tag{3}$$

Pages belonging to the "Selected" class are considered adequate solutions for the programming task. The pages classified as "Selected" must have code examples of how to implement the programming task surveyed in the search engine, so they must have the value "yes" for the Code Examples criterion. For the Degree of Focus criterion, the pages must have a value greater than or equal to 4, indicating that features in the solution provide what the user wants to learn. Pages with values less than or equal to 2 for this criterion should not be selected because they contain code or content that is not related to the query. Regarding to the Solution Size criterion, pages with a value of two or 4 are selected, as they present solutions that are close to the average number of method calls found on the *n* pages obtained by the search engine. The pages that have the values 1 or 5 for this criterion are far from the average, so they are not selected. Pages rated with value three should not be selected because they could not be evaluated. For example, for the Solution Size criterion, pages that do not have source code are rated with three.

In the construction of the ground-truth, we evaluated the first 20 pages returned by Google, for the 10 queries shown in Table 1, totaling 200 evaluated pages. Regarding the chosen number of evaluated pages, we proceeded with a sequential evaluation of the best ranked pages returned by Google and observed that typically adequate solutions were

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

found up to the 15th position, so we decided to add a safety margin and evaluate up to the 20th position. The evaluation of the 200 pages by only one person has an internal threat that the evaluation may be biased towards the interpretation of the criteria by only this rater and different raters could have different interpretations and thus produce different ground-truth datasets. In order to mitigate this internal validity threat, we conducted an agreement analysis study where three raters evaluated 40 of the 200 pages. In a first phase, all raters evaluated blindly the pages for each one of the three criteria (presence of code examples, size of the solution, degree of focus) without knowing the evaluation of each other. Then we applied an inter-rater reliability test to assess the potential agreement level for the rest of the queries. We have chosen the Krippendorff's α coefficient, which applies for more than two raters. The results showed low potential agreement, with Krippendorff's α coefficient equals to 0.192, 0.227, 0.163, for "Code Example", "Degree of Focus", and "Solution Size", respectively, thus confirming the threat. In order to circumvent, the interpretation bias, we investigate for each criterion the reasons of disagreement. For the criterion "Code Example", two reasons were identified. One reason is when at least one of the raters considered that even if the page does not contain source code, but other pages that could be accessible from it by navigating through, then the rater considered the criterion as "True". This kind of disagreement occurred four times, and we have then defined as standard that this should be "False". Other reason is when the page has code but it is not Java code. This occurred two times and we defined that the truth is that the page contains no code. For the criteria, "Degree of Focus", when the chosen reference value points out to a low degree of focus, the misinterpretation of the criterion was because the rater considered hit pages those that contained other features beyond the focus of the solution, such as, tests, web layout, specialised deployment, manipulation of database, only a partial solution, other API, among others. When the reference value points out to a hit, the rater typically agreed that it is a incorrect rating. For the criteria "Size of Solution", the disagreements were either because of a overrating or underrating of size. The overrating of size occurred more on pages that were hit. An explanation for that is that the rater did not realise that overrating the size of the solution would imply in considering it as a non-hit page, and this was not the real intention of the rater. After clarifying, all the points of disagreement with the raters, they could update their rates. Moreover, we also decided to adjust for inter-rater analysis to transform the 5-level Likert scale results of "Size of Solution" and "Degree of Focus" to a 3-level scale, where the 3-level is more adherent to the definition of hit/non-hit. After, the adjustments we found the Krippendorff's α coefficient equals to 0.873, 0.806, 0.925, for "Code Example", "Degree of Focus", and "Solution Size", respectively, with the two latter with 3-level scale, which are >0.80, as recommended by Krippendorff [35]. It is important to note that the remaining disagreement that is still remaining in the updated versions does not influence the decision for hit/non-hit of the pages, and thus, we use those criteria in the rest of the pages. One important point that was raised out in the discussion phase of the agreement analysis is that the ground-truth construction criteria is mostly conservative for choosing hit pages, that is, the hit pages are typically unarguably hit ones, whereas non-hit pages could be still argued as useful despite non-desired characteristics. So, summing up, the proposed ground-truth will evaluate the filters with a strict notion of value of the pages, which makes sense for our general goal: *filtering pages that are indisputably relevant pages.*

All three raters have at least 15 years of experience with the Java programming language.

About the number of pages evaluated, we must argue that although we provide quantitative data analysis, the ground-truth construction is a qualitative work, so a larger scale manual construction is typically prohibitive. In total, we evaluated 200 pages (20 pages per query), as this evaluation was thorough and careful, it took months to perform and curate. The 200 assessed pages number can be considered sufficient to support our conclusions. The data obtained in this study are available at a Zenodo repository.[1]

5 | RESULTS

This section shows the results for the approach assessment using the methodology shown in Section 4. The four approaches are compared for each query against the ground-truth.

Figures 2–4 show the evaluation of the 20 pages returned by the Google search engine, for each of the 10 queries. The x-axis represents the possible values for the given criterion. The y-axis represents the number of pages.

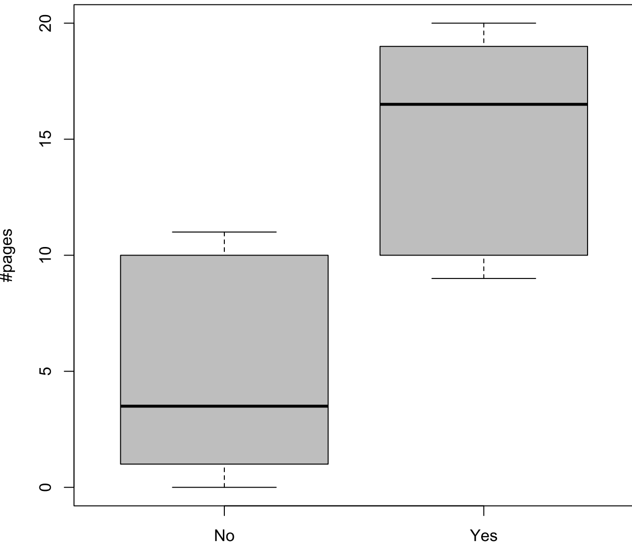


FIGURE 2 Results of the evaluation of the pages returned by the search engine for the code example criterion.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

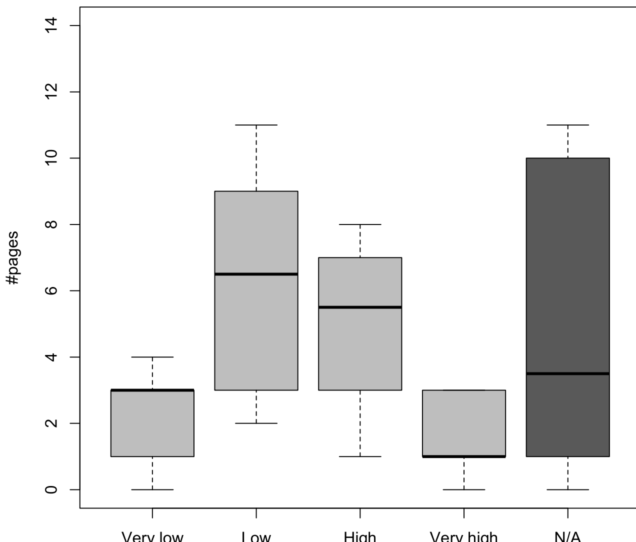


FIGURE 3 Results of the evaluation of the pages returned by the search engine for the degree of focus criterion.

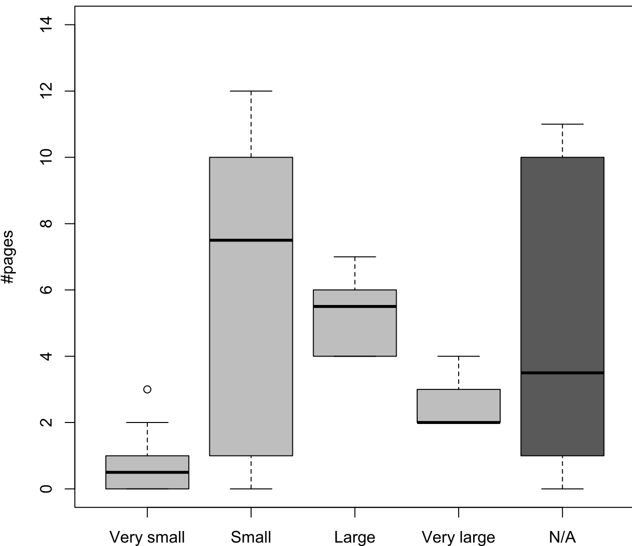


FIGURE 4 Results of the evaluation of the pages returned by the search engine for the solution size criterion.

5.1 | RQ1: To what extent does Google's search engine return relevant pages to software developers?

Figure 2 shows that for all evaluated queries, there are pages returned by the search engine that do not have code examples (25% of the total pages evaluated). This result also shows a variation from 0 to 11 pages without source code depending on the evaluated query, revealing that depending on the query, around half of the pages do not have source code among the first 20 pages returned.

We observe in Figure 3 that several pages returned by Google's search engine have no focus, that is, the solutions present in these pages are not related to the query (as shown in the 'Very low' bar) or little focus, that is, the solutions present on these pages are little related to the query carried out by the user (as shown in the 'Low' bar). These pages contain implementations of features that are not related to the programming task. The 'N/A' bar shows the pages (25% of the total pages evaluated) that have no source code, so the *Degree of Focus* of the implementation is not applicable.

Figure 4 shows that for all evaluated queries, some pages have solutions that are much larger than the average in relation to the other pages ("Very large" bar). These pages add up to 12% (24 pages) of the total pages evaluated. Pages with large solutions usually have no focus. Moreover, 91.7% of the pages with very large solutions have a low or very low degree of focus. The approach proposed in this work remove pages with solutions far above the average, improving the result. The "Very small" bar shows that a small portion of the pages evaluated (4% of the total pages evaluated) have very small solutions in relation to the average. These pages also tend to lose focus. We observed that 87.5% of the pages that have very small solutions have a low or a very low degree of focus. The proposed approach proposed also filter these pages. The "N/A" bar shows the number of pages without source code for which the criteria *Solution Size* is not applicable (25% of the total evaluated pages).

When applying the Equation (3) of Subsection 4.3 to the values of the criteria shown in Figures 2–4, only 31% of the pages returned by the search engine were selected, that is, these pages are considered good solutions for developers. The rest of the solutions (69%) are not interesting for the developer who is learning a programming task, as they have undesirable characteristics, such as no code examples, low degree of focus or/and pages with solution sizes that are far from average. This shows the importance of developing filters to remove pages with such characteristics.

RQ1 Answer: Only 31% of the total pages returned by Google's search engine have relevant solutions for software developers. The rest of the solutions have undesirable characteristics, such as no code examples, low degree of focus or/and pages with solution sizes that are far from average (outlier pages). 25% of the total pages evaluated do not have code examples. Only 33% of the analysed pages have a high or very high degree of focus on the solutions sought by the user. 16% of the total pages evaluated are outlier pages.

5.2 | RQ2, RQ3, RQ4: Effectiveness of page filtering

This subsection shows the results of the comparison of the four approaches (JG, GOR, GORCTO and GORCUO) defined in Section 4. Having the result of the selection of pages

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

obtained in the previous subsection, for each of the approaches, the metrics were calculated: Precision, Recall and F-Measure. The same number of pages was given as input for all approaches. We did this incrementally, that is, we first provided three pages as input for all approaches and we calculated the metrics (so, they are aligned for a comparison), then we provided four pages as input for all approaches and we calculated the metrics, and so on, until we reached 20 pages as input. In Figures 5–7, the number of pages given as input by the approach are the values on the x-axis.

Figure 5 shows the graph comparing the precision of the four evaluated approaches. The yellow, orange, green and blue bars represent the JG, GOR, GORCTO and GORCUO approaches, respectively. On the X axis of the graph are the number of pages given as inputs for the approaches, which varies from 3 to 20 pages, increasing by one. On the Y axis are the values of the metric precision, obtained by the given approach when being executed having as input the 10 queries evaluated in this work. The graph shows that the JG (Just Google) approach has low precision, where the medians of the yellow bars are below 40%, for most pages given as input. This reinforces the need for filters to remove irrelevant pages returned by the search engine. The GOR approach, which implements the filter for removing outliers pages, which manages to remove irrelevant pages, increasing the precision for all page numbers given as inputs, compared to JG. The graph shows that the GORCTO approach is very unstable, with a very large variation in precision, for all page numbers given as inputs. The GORCUO approach, on the other hand, has bars with higher medians compared to the other three approaches, with the best results for the precision metric, being very efficient in removing irrelevant pages. The GORCUO approach shows good results in the intervals 15–19 of page numbers given as input, with the best result with 16 pages given as input.

Figure 6 shows the results of the recall metric for the four approaches. For the JG approach, the recall will always be 100%, as this approach is used to obtain the 20 pages used in the evaluation (baseline). The problem with JG is the precision and this is the main motivation of this paper: improving the precision of the search engine results. The recall boxplot of JG

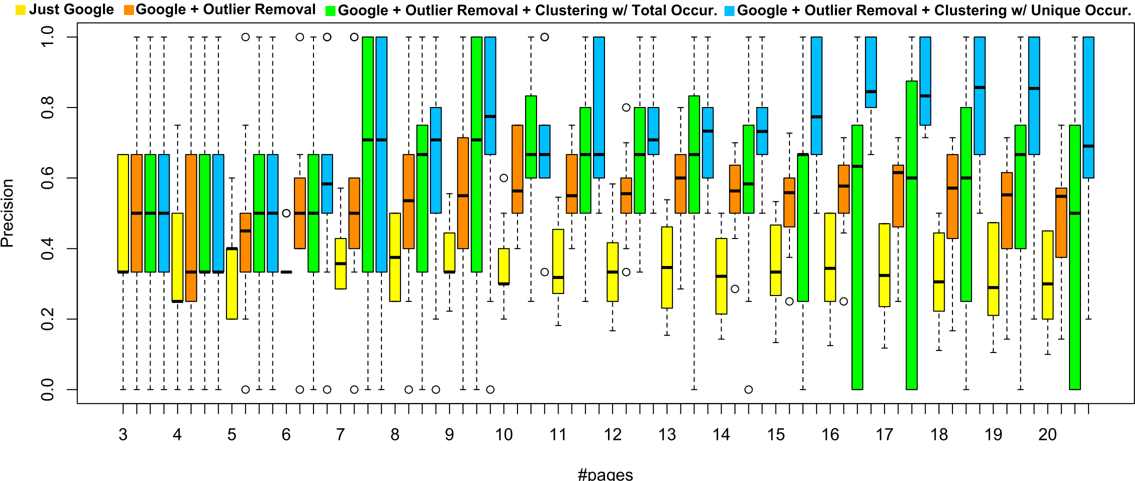


FIGURE 5 Precision of the four approaches increasing the number of pages given as input.

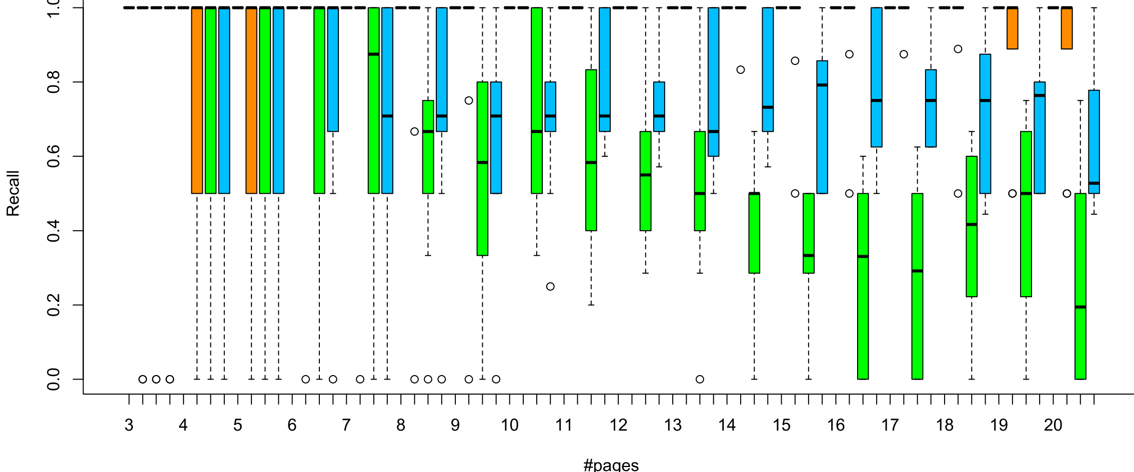


FIGURE 6 Recall of the four approaches increasing the number of pages given as input.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

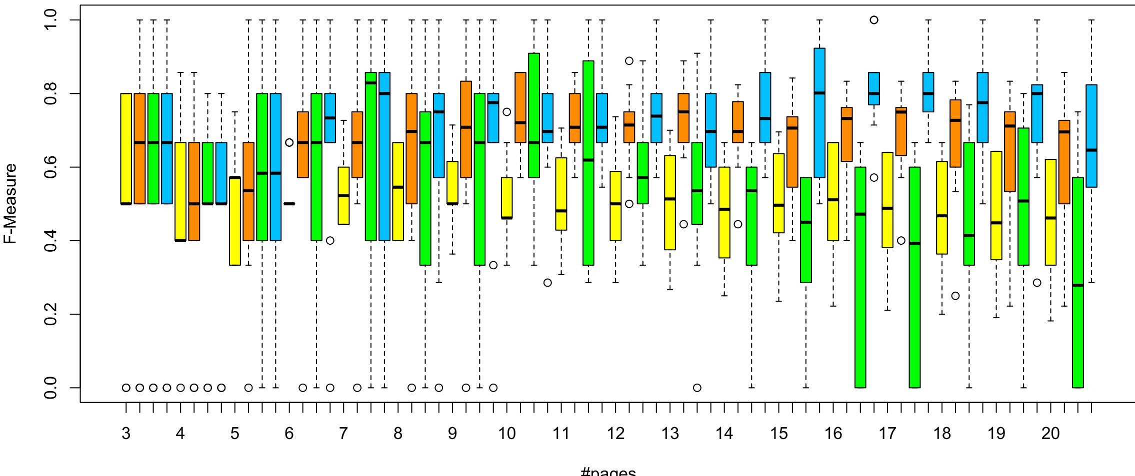


FIGURE 7 F-Measure of the four approaches increasing the number of pages given as input.

is just a mark in the top of the graph, that is, recall equals to one. The graph reveals that even after applying the filter for removing outlier pages, the GOR approach has 100% recall for most of the page numbers given as input (except for numbers 4, 5, 19 and 20). This indicates that the outlier page removal filter is effective in removing irrelevant pages. The graph shows that the GORCTO approach has the worst results for this metric, with great variability in its values and low median for most of the page numbers given as input. The GORCUO approach has a median of around 70% for this metric. Even removing a portion (around 30%) of the relevant pages, the rest of the relevant pages are already sufficient for developers to obtain the desired solutions. Since the GORCUO approach is very efficient at removing irrelevant pages, as shown in the precision graph in Figure 5, this approach is the best option for developers to obtain relevant pages.

For the F-Measure metric, Figure 7 shows that among the four approaches, GORCUO (blue bar) performed better. The GORCTO approach (green bar) varies widely for the vast majority of the landing pages, in addition to having a low overall median. The GOR approach (orange bar) shows better results than the GORCTO approach. The JG approach (yellow bar), which uses only the Google search engine, has a low overall average.

5.3 | RQ5: To what extent applying the filters proposed in this work improve the ranking quality of pages returned by search engines?

Table 5 shows the results of metrics Hit@K, Recall@K, MRR@K, MAP@K and NDCG@K, for $K = 5$, considering the 20 pages returned by the search engine as input. As the GOR, GORCOT and GORCUO approaches select pages applying their filters, the final number of pages is less or equal 20. If the respective number of pages selected by these approaches is less than 5, the number of pages is completed up to five with the first non-selected pages of the JG approach, so we can calculate the metrics for $K = 5$.

TABLE 5 Comparison of Hit, Recall@K, MRR, MAP and NDCG, for $K = 5$, considering the 20 pages returned by the search engine as input, for the 10 queries evaluated.

	Hit@K	Recall@K	MRR@K	MAP@K	NDCG@K
JG	**1**	0.347	0.595	0.558	0.663
GOR	**1**	0.472	0.658	0.624	0.739
GORCOT	0.8	0.378	0.72	0.657	0.655
GORCUO	**1**	**0.624**	**0.867**	**0.839**	**0.897**

Note: The values in bold represent the highest values obtained for each of the metrics.

Regarding the results of the Hit metric, shown in Table 5, we can observe that, for the 10 evaluated queries, the approaches JG, GOR and GORCUO manage to find at least one relevant page among the first 5 pages returned. The GORCOT approach, for two queries, could not find any relevant page among the first five returned pages.

The Recall@K metric shown in Table 5 was calculated considering the total number of relevant pages found in the first 20 pages returned by the JG approach. The table shows the results of this metric for $K = 5$, that is, for the first 5 pages returned by the approaches. Of the four evaluated approaches, GORCUO obtained the best results. This shows that after applying the filters, more relevant pages are brought to the top-5 ranking using this approach. The GOR approach had the second highest value for this metric, which indicates that after applying the filter that removes outlier pages, the top-5 ranking has more relevant pages than the JG approach, which uses only the ranking returned by Google. The GORCOT approach, which uses outlier removal as the GOR approach, had results very close to the JG approach, this indicates that after the application of the clustering from the GORCOT approach, the top-5 of the ranking worsens in relation to the top-5 of the GOR approach. This is due to filter instability of the GORCOT clustering.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 5 shows the results of the MRR, MAP and NDCG metrics, for $K = 5$. As we can see, the GORCUO approach presents better results for all metrics (MRR, MAP and NDCG). These results show that, after applying the filters proposed in this approach, we obtain a top-5 ranking of better quality.

> *RQ2*: *To what extent does filters to remove outlier pages w.r.t. Page size help to remove irrelevant pages returned by search engines?* The results shown in the F-Measure metric graph in Figure 7 reveal that the GOR filter, which removes outlier pages, helps remove irrelevant pages for software developers. For all page numbers given as input, the filter has F-Measure higher than Google Search Engines (JG), except for page number equal to 5, where the median of the JG is slightly above the median of the GOR filter.

> *RQ3*: *To what extent does applying clustering algorithms w.r.t. Total of method calls help to remove irrelevant pages returned by search engines?* The results of the Precision, Recall and F-measure metrics graphs shown in Figure 5–7 reveal that the approach (GORCTO) that uses the clustering algorithm having as input the total occurrence of method calls on the pages returned by the Google search engine has very large variability in results, for most page numbers given as input to the approach. In addition to being an approach that removes many pages relevant to software developers. So this approach is unreliable for removing irrelevant pages.

> *RQ4*: *To what extent does applying clustering algorithms w.r.t. Unique occurrences of method calls help to remove irrelevant pages returned by search engines?* The F-Measure metric results shown in the graph in Figure 7 reveal that the GORCUO filter helps remove irrelevant pages for software developers. For all page numbers given as input, the filter has higher F-Measure than Google search engines (JG). For example, for the number of pages equal to 17, the median of the GORCUO filter is around 0.8, while the Google search engine has a median around 0.5, for the same number of pages given as input.

> *RQ5*: *To what extent does applying the filters proposed in this work improve the ranking quality of pages returned by search engines?*
> As we can see in Table 5, the filters of the GOR and GORCUO approaches improve the top-5 ranking quality, for the 3 analysed metrics (MRR, MAP and NDCG). The GORCUO approach performed better than GOR. The GORCOT approach, on the other hand, improves the ranking quality for the MRR and MAP metrics, compared to the JG approach. For the NDCG metric, the GORCOT approach has a slightly lower value than the JG approach. So, the recommendation is that the GORCUO approach is the best one to improve ranking of programming solutions from search engines.

6 | DISCUSSION

The results of the evaluation carried out on the first 20 pages returned by the Google search engine, for the 10 analysed queries, reveal that only 31% of the evaluated pages have relevant solutions for software developers. The rest of the solutions have undesirable characteristics, such as no code examples, low degree of focus or/and pages with solution sizes that are far from average. These results show the importance of developing filters to remove irrelevant pages for developers.

Getting relevant pages in search engine results is a difficult task for developers, as they may have to inspect those pages manually, which obviously takes more effort to get the desired result. We observed that pages with high or very high degree of focus are scattered in the ranking returned by search engines. This may be due to the fact that search engines consider generic features that are not customised for solutions related to software development [20]. For example, pages may contain high quality text content but may lack source code examples. Or pages may have source code, may contain high quality textual content, but the solution may not be focussed on the query searched by the developer, that is, the solution may not solve directly the user's problem. Some implications and possible consequences are as follows: *1) the user stops browsing at a solution ranked in the first positions, but with low degree of focus*. A consequence is that the user may spend more time abstracting the desired solution from low-focus content, rather than looking for more focussed solutions in the lower ranking positions. *2) the user continues to look for solutions with better focus on lower ranking positions*. A consequence is that the developer has to read several pages until finding one with content focussed on the desired solution, demanding more effort. As we can see in this study, there are pages with content with a high degree of focus in the 15th position of the ranking, many developers may not examine the content of the pages in deep positions of the ranking.

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Therefore, these developers may not find a good solution that is at the bottom of the ranking.

Another feature that makes pages returned by search engines irrelevant to software developers is the absence of source code. The results show that some of the pages returned by search engines do not have source code. Probably, source code is not a major priority for search engines when ranking pages. A developer looking for code examples would spend considerable time visiting pages without source code.

The results showed that the GOR filter, which removes outlier pages, is effective in removing irrelevant pages returned by search engines. This is due to the filter removing pages that do not have source code or with small source code in relation to the average size of the solutions present in the pages returned by the search engine. The filter also removes those pages with large solution size relative to average size. As shown in the results of this work, pages with solutions that are too small or too large in relation to the average size of the solutions returned by the search engine have low or very low degree of focus, which makes the solution irrelevant for developers. For example, for the query *"how to implement menu java javafx"*, evaluated in this study, one of the pages returned by the engine just defines the SeparatorMenuItem class, without a practical example of use. Since the page has only three method calls that are related to the class's constructor method, that page is removed by the GOR filter, as the number of occurrences of methods is very low than the average occurrence of method calls for this programming task, having an average of 66.8 calls. For the same query, another page returned by the search engine has an extensive solution, with 147 method calls, which addresses features such as radio button and checkbox in the solution. The GOR filter also removed this page.

The results showed that the GORCTO filter has a great variability in the results for the different queries given as input. This filter uses a clustering algorithm where the attribute is the total method calls found in the page solutions, which makes it very unstable. Since pages that have been clustered in the same cluster may have solutions that lose focus. For example, for the query *"how to implement crud operations java jpa"* evaluated in this study, two pages (A and B) were clustered in the same cluster, even though one of the pages contained a solution that lost focus. These pages were clustered in the same cluster due to the characteristic of the GORCTO approach of using the total hits of method calls as the attribute value for the clustering algorithm. For this case, pages A and B have some methods in common with occurrence value equal or very close, these methods help these pages to be in the same cluster. For example, the *begin* method occurs 4 times on page A and 4 times on page B. The *getTransaction* method occurs 10 times on page A and 9 times on page B. The *find* method occurs 4 times on page A and 4 times on page B. The *commit* method occurs 4 times on page A and 5 times on page B. However, on page B there are several methods that do not occur on page A. For example, the *getText*, *parseInt*, *removeById* and *showInputDialog* methods. These methods are related to the graphical user interface part of the solution. In other words, page B also addresses issues related to the graphical interface, losing the main focus of the user query, which is to implement CRUD operations. As cluster selection considers the sum total of occurrences of method calls that occur on at least two pages in the clusters, pages with low focus are placed in the same cluster as pages with high focus, as a portion of the method calls on those pages occur multiple times.

The results showed that the GORCUO filter is effective in removing irrelevant pages, since the filter uses a clustering algorithm where the attribute is a single occurrence of each method call. This feature prevents interference from method calls that often occur in solutions from different pages, which reduces the quality of the generated clusters. For example, if the GORCUO approach was applied in the example of the previous paragraph, probably pages A and B would not be clustered in the same cluster, since the occurrences of the method calls would be counted only once. For example, the *getTransaction* method would have an attribute value equal to 1 for pages A and B, instead of 10 for page A and 9 for page B. In this way, this method would have no weight at the time of the agglomeration of the pages, that is, pages A and B would not be so close together because of this method call and other method calls with high occurrence numbers as attribute value.

The results showed that the GORCUO filter improves the quality of the top-5 ranking, as shown in the results of the MRR, MAP and NDCG metrics (Table 5). Even removing some relevant pages as shown in the Recall metric results (Figure 6), this filter is effective in removing irrelevant pages as shown in the Precision (Figure 5) and MAP metric results. This improvement in ranking quality is due to the use of a filter that uses a clustering algorithm, where the attribute is a unique occurrence of each method call. This algorithm manages to mine the relevant pages spread out in the ranking, bringing them to the top of the ranking, consequently, improving the ranking quality.

A point of discussion is the practical use of the proposed approach. Regarding to the legal issues to use the general search engine, one would need to acquire the licence to use the search engine as a built-in service. For instance, actually Google offers an API where one could use it to get the pages returned by the Google search engine. At the time of this manuscript publication, there is a plan to use the Google API that costs \$5 per 1000 queries (\$0.005 per query), with no daily query limits. Another possibility of application would be a partnership with the search engine provider itself, based on the knowledge that our approach helps developers finding solutions faster than using Google alone. Actually, the results of the paper are available for anyone to implement such a system using the proposed approach.

7 | THREATS TO VALIDITY

In this section, we show some threats to the validity of this work.

Internal validity. This kind of threat is about how sure we can be that the treatment actually caused the outcome. In this study, a threat to internal validity was the number of pages

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

returned by the search engine considered in the study of mining approaches. A possible larger number of pages could influence the results. To minimise this threat, we conducted a preliminary study varying the number of pages (from 3 to 20 pages) and found that from 15 pages the results begin to converge. Another threat to internal validity is the variation among raters in the interpretation of the evaluation criteria of each page during the manual construction of the ground-truth. Evaluators may apply the criteria using different assumptions, then leading to different definitions of the ground-truth. We mitigate this threat by clarifying and making the evaluation criteria as objective as possible, conducting an agreement analysis study. After clarifying the criteria among evaluators, the agreement on what is a hit or not could have a consensual marking. An important point is that the pages that are considered hit are strongly agreed among the evaluators, so pages that may raise some discussion if they are irrelevant or not, are not considered hit in the ground-truth. Our evaluation results have shown that the approach tends to effectively filter out these pages, and thus tending to return only pages that are indisputably relevant.

External validity. This kind of threat is related to whether we can generalise the results outside the scope of our study. In this study, to threaten external validity, the results of the proposed approaches are limited to the JAVA programming language. The approaches may have different results with other programming languages.

Conclusion validity. This kind of threat is about how sure we can be that the treatment is really (statistically) related to the actual outcome. In this study, we constructed a ground-truth with 10 queries, analysing 20 pages per query (200 pages in total). So, enlarging the number of queries would possibly impact the results. To mitigate this threat, we have carefully chosen queries in distinct API domains to cover a broad range of content.

8 | RELATED WORK

Research in the field of mining search engine results is a practice that has given good results in different contexts. For example, in the work of Saraswathi and Vijaya [21], they improve the quality of search engine results by identifying and removing spam links. In the work of Suneetha et al. [22], the authors aim at organising web search results into clusters facilitating quick browsing options to the browser providing interface to results precisely. On the other hand, in the work of Suo et al. [23], the authors analyse the results of the search engines using a new summary approach which calculates the sentence weight utilising the information of the distance between words.

Several other authors [24–30] have been working to improve search engine ranking results, as these search engines return many pages irrelevant to users. This shows the importance of research related to this topic. For example, appropriate solutions for developers are not always among the first pages ranked by search engines. They can be mixed with other pages with inadequate characteristics, such as without code examples and little focus on the desired solution. It is a challenge to remove pages with these inappropriate characteristics. Xia et al. [15] evaluate the quality of the pages returned by the search engines. They argue that several reviewers questioned that the quality of the content of the pages returned by search engines is low, which requires more time to find the desired content. The approach that we are proposing removes pages with inappropriate content for developers, selecting only the relevant pages.

Saraswathi and Vijaya [21] proposed a generic tool for link spam detection in search engine results using graph mining. According to the authors, Web Spam is a part of the web page that was created with the intention of increasing its ranking in search engines. The approach that we are proposing in this work removes Web Spam that is not related to the query made by the developer, through the proposed filters.

It is a challenge to obtain method calls present in the source code of solutions found on the web pages returned by the search engine because in the source code of the page, they are mixed with method calls that are part of the construction of the web page. For example, method calls within the *script* and *style* tags present on the web page are not part of the solution sought by the developer, as they are related to the construction of the web page itself. Stylos and Myers [31] obtain the method calls present in the source code of the web pages by means of a list of method calls extracted from the official documentation of the JAVA language API, with the limitation of collecting method calls only from this programming language. Zhong and Zhendong [36] propose to extract method calls by identifying typos, which are code names that do not appear in natural language dictionaries. The typo is classified as a method through the recognition of the opening and closing parenthesis characters (the approach makes a count of these characters through string processing). The approach that we are proposing also extracts method calls through the recognition of the opening and closing parenthesis characters, but we use a regular expression to obtain the method calls automatically.

Stylos and Myers [31] and Diamantopoulos et al. [32] propose approaches for obtaining pages and method calls, but these works are limited only to the JAVA programming language. The approach that we are proposing obtains pages with method calls from several different computer programming technologies.

Mandelin et al. [33] created a tool called Prospector, that given a class or an object previously known by the programmer, the tool helps to search for code examples. A disadvantage of this approach is the requirement of prior knowledge of the class or the object, this greatly limits the use of the approach for programmers who do not know the programming technology they wish to learn. Our approach does not have this limitation, that is, the developer does not need to have prior knowledge of the method calls to use the proposed approach.

Chatterjee et al. [12] have proposed a new code search technique called Sniff, which retains the flexibility to perform a

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

search in English, while obtaining small pieces of relevant code needed to perform the desired task. In Sniff, a programmer issues a query expressing the programming task in English and the tool returns a small set of relevant code snippets. Regarding the work of Mandelin et al. [33], Sniff has the advantage of eliminating the need for prior knowledge about the API to be reused.

Zheng et al. [34] mine the results of search engines to collect relevant information from a given API from an old library and then recommend candidates for the API from the new library that appear frequently in web search results returned by search engine. Our work also uses the results returned by search engines, having a different objective.

9 | CONCLUSION AND FUTURE WORK

In this work, we have shown that the Google search engine may return a non-negligible number of pages with low focus on the solution sought by the developer. The results also show that there are several pages returned with no code samples. Pages with these characteristics are not interesting for developers looking for a solution for their problem in hand, motivating our proposed approach to filter only relevant solutions for developers.

Our study compared four different approaches for filtering relevant solutions. The proposed approach that removes outlier pages and applies a clustering algorithm with unique occurrences of method calls as attribute value obtained higher precision and F-Measure than the other three evaluated approaches. These results demonstrated the effectiveness of creating filters to improve the results returned by search engines.

As future work, the main one would be the implementation of a practical tool. This tool could be a web site with a simple graphical interface, where the user would only enter the specification of the programming task, the name of the programming language and the name of the API. Given this information as input, the system would perform processing using the proposed approach, and then the system would return a web page with links to the relevant pages for the developers. This system could also be embedded into IDEs for the developer convenience. Another future work would be to use the achieved approach to create documentation focussed on the programming task queried by a developer on a search engine. That documentation could contain containing descriptive explanations, code examples and frequently asked questions about method calls found in the solutions of the pages would be returned on-the-fly. Other future work would be empirically evaluating the actual benefits of filtering web search results during a development activity. In this work, we assume that people should know which API to perform the task. As future work, one may add a procedure in the approach, in the case that the user does not inform the name of the API in the query. The approach could identify and indicate the APIs found in the N pages returned by the search service. In addition, as future work the approach could indicate the most relevant APIs, considering the filters already implemented in the current approach.

AUTHOR CONTRIBUTION

Adriano M. Rocha: Conceptualisation, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualisation, Writing – original draft, Writing – review & editing. Marcelo A. Maia: Conceptualisation, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualisation, Writing – original draft, Writing – review & editing.

ACKNOWLEDGEMENTS

We acknowledge the Brazilian funding agencies CNPq and FAPEMIG for partially supporting this work.

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no known conflicting financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY STATEMENT

The authors declare that they have made data available at https://doi.org/10.5281/zenodo.6467629.

ORCID

Adriano M. Rocha https://orcid.org/0000-0002-3797-1260
Marcelo A. Maia https://orcid.org/0000-0003-3578-1380

REFERENCES

1. Murphy, G.C., et al.: Enabling Productive Software Development by Improving Information Flow. Apress, Berkeley (2019)
2. Bajracharya, S., Ossher, J., Lopes, C.: Sourcerer: an infrastructure for large-scale collection and analysis of open-source code. Sci. Comput. Program. January 79, 241–259 (2014). https://doi.org/10.1016/j.scico.2012.04.008
3. Kataria, S., Sapra, S.: A novel approach for rank optimization using search engine transaction logs. In: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 3387–3393. March (2016)
4. Sharma, A.K., et al.: Web search result optimization by mining the search engine query logs. In: 2010 International Conference on Methods and Models in Computer Science (ICM2CS-2010), pp. 39–45. December (2010)
5. Silva, R.F.G., et al.: CROKAGE: Effective Solution Recommendations for Programming Tasks by Leveraging Crowd Knowledge, pp. 4707–4758. Empirical Software Engineering (2020)
6. Silva, R.F.G., et al.: Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In: Proc. of the 27th Intl. Conf. on Program Comprehension (ICPC'2019), pp. 358–368. IEEE Press, Montreal (2019)
7. Delfim, F.M., et al.: Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. J. Braz. Comput. Soc. 22(1), 9 (2016). https://doi.org/10.1186/s13173-016-0049-0
8. Agrahri, A.K., Manickam, D.A.T., Riedl, J.: Can people collaborate to improve the relevance of search results? In: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 283–286. October, Lausanne (2008)
9. Hora, A.: Googling for software development: what developers search for and what they find. In: IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp. 317–328 (2021)

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

10. Niu, H., Keivanloo, I., Zou, Y.: Learning to rank code examples for code search engines. Empir. Software Eng. 22(1), 259–291 (2017). https://doi.org/10.1007/s10664-015-9421-5
11. Sim, S.E., et al.: How well do search engines support code retrieval on the web? ACM Trans. Software Eng. Methodol. 21, 1–25 (2011). https://doi.org/10.1145/2063239.2063243
12. Chatterjee, S., Juvekar, S., Sen, K.: SNIFF: a search engine for java using free-form queries. In: Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, pp. 385–400. March, Berlin (2009)
13. Hora, A.: Characterizing top ranked code examples in google. J. Syst. Software 178(4–5), 37 (2021). https://doi.org/10.1016/j.jss.2021.110971
14. Cho, J., Roy, S.: Impact of search engines on PagePopularity. In: Proceedings of the 13th International Conference on World Wide Web, pp. 20–29. New York (2004)
15. Xia, X., et al.: What do developers search for on the web? Empir. Software Eng. 22(6), 3149–3185 (2017). https://doi.org/10.1007/s10664-017-9514-4
16. Rahman, M.M., Roy, C.K., Lo, D.: Automatic query reformulation for code search using crowdsourced knowledge. Empir. Software Eng. 24(4), 1869–1924 (2019). https://doi.org/10.1007/s10664-018-9671-0
17. Panth, B., Maclean, R.: Introductory overview: anticipating and preparing for emerging skills and jobs—issues, concerns, and prospects. Education in the Asia-Pacific Region: Issues, Concerns and Prospects 55 (2020)
18. Arthur, J.D., Stevens, K.T.: Document quality indicators: a framework for assessing documentation adequacy. Journal of Software Maintenance 4(3), 129–142 (1992). https://doi.org/10.1002/smr.4360040303
19. Smart, K.L.: Assessing quality documents. ACM J. Comput. Doc. 26(3), 130–140 (2002). https://doi.org/10.1145/604228.604236
20. Evans, M.: Analysing Google rankings through search engine optimization data. Internet Res. 17(1), 21–37 (2007). https://doi.org/10.1108/10662240710730470
21. Saraswathi, D., Vijaya, A.: A generic tool for link spam detection in search engine results using graph mining. In: International Conference on Pattern Recognition, Informatics and Mobile Engineering, pp. 282–286 (2013)
22. Suneetha, M., Fatima, S.S., Pervez, S.M.Z.: Clustering of web search results using Suffix tree algorithm and avoidance of repetition of same images in search results using L-Point Comparison algorithm. In: International Conference on Emerging Trends in Electrical and Computer Technology, pp. 1041–1046 (2011)
23. Suo, H., Zhang, W., Zhang, Y.: Research on automatic summarization based on search engine result. In: 2009 International Conference on Web Information Systems and Mining, pp. 74–77. November (2009)
24. Amin, G.R., Emrouznejad, A.: Optimizing search engines results using linear programming. Expert Syst. Appl. 38(9), 11534–11537 (2011). https://doi.org/10.1016/j.eswa.2011.03.030
25. Caramia, M., Felici, G., Pezzoli, A.: Improving search results with data mining in a thematic search engine. Comput. Oper. Res. 31(14), 2387–2404 (2004). https://doi.org/10.1016/s0305-0548(03)00194-1
26. Hong, Y., Vaidya, J., Lu, H.: Search engine query clustering using top-k search results. In: Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, pp. 112–119 (2011)
27. Kim, Y., Thompson, K.C., Teevan, J.: Using the crowd to improve search result ranking and the search experience. ACM Trans. Intell. Syst. Technol 7(4), 1–24 (2016). https://doi.org/10.1145/2897368
28. Lau, T., Horvitz, E.: Patterns of search: analyzing and modeling web query refinement. In: Proceedings of the Seventh International Conference on User Modeling, pp. 119–128. Banff, Canada (1999)
29. Zahera, H.M., El-Hady, G.F., El-Wahed, W.F.A.: Query recommendation for improving search engine results. In: Proceedings of the World Congress on Engineering and Computer Science, pp. 20–22. San Francisco (2010)
30. Zhuang, A., Cucerzan, S.: Re-ranking search results using query logs. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, pp. 860–861. November, Arlington (2006)
31. Stylos, J., Myers, B.A.: Mica: a web-search tool for finding API components and examples. In: Visual Languages and Human-Centric Computing, pp. 195–202. September (2006)
32. Diamantopoulos, T., Karagiannopoulos, G., Symeonidis, A.: CodeCatch: extracting source code snippets from online sources. In: 2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, pp. 21–27. June (2018)
33. Mandelin, D., et al.: Jungloid mining: helping to navigate the API jungle. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 48–61. June, Chicago (2005)
34. Zheng, W., Zhang, Q., Lyu, M.: Cross-library API recommendation using web search engines. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 480–483. September, Szeged, Hungary (2011)
35. Krippendorff, K.: Content Analysis: An Introduction to its Methodology. Sage Publications (2004)
36. Zhong, H., Zhendong, S.: Detecting API documentation errors. In: Proceedings of the 2013 ACM SIGPLAN International Conference on Object oriented Programming Systems Languages & Applications (2013)

How to cite this article: Rocha, A.M., Maia, M.A.: Mining relevant solutions for programming tasks from search engine results. IET Soft. 1–17 (2023). https://doi.org/10.1049/sfw2.12127

17518814, 0, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12127 by CAPES, Wiley Online Library on [15/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License