International Journal of Research in Advanced Engineering and Technology
ISSN: 2455-0876; Impact Factor: RJIF 5.44
www.engineeringresearchjournal.com
Volume 3; Issue 1; January 2017; Page No. 46-53

Cluster-Based search over online support resources for a visual programming language

[1] Chris Scaffidi, [2] Chris Chambers

[1] School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA
[2] National Instruments, Austin, TX, USA

Abstract
Engineers and scientists increasingly do programming for data acquisition, analysis and other activities. They often turn to online support resources to learn about programming and about specific programming tools. Unfortunately, finding resources can be difficult with existing systems, which present search results that are sorted but otherwise disorganized. Therefore, we have developed a new system that searches over forum posts, tutorials, whitepapers, and other online support resources, with an emphasis on organizing search results into clusters. This search engine, which we prototyped for one programming environment popular with engineers and scientists (LabVIEW), automatically sorts these clusters of search results based on relevance to a user query, and it automatically generates names for clusters to succinctly summarize their content. An empirical user evaluation confirmed the system successfully sorted clusters in order of relevance and selected accurate labels for naming each cluster.

Keywords: engineers; scientists; programming; online support; search engine

Introduction

Online support resources enable programmers to find help about how to use new languages. In particular, engineers and scientists increasingly do programming as part of their jobs but, often lacking training as professional programmers, they must turn to online support resources to learn about programming and about specific programming tools [1, 2, 3].

Unfortunately, finding resources can be difficult with existing systems, which present search results that are sorted but otherwise disorganized. The online search system for LabVIEW programmers, which we have studied in prior research [4], illustrates this challenge. LabVIEW includes a visual programming language with built-in hardware integration and support for programming tasks common to science and engineering, such as data acquisition, analysis and graphing [5]. For example, a programmer who searches the support site for help on socket input would retrieve code samples for how to retrieve data all at once, how to retrieve it incrementally within a loop, and how to register an event-listener for receiving data values in bursts as they arrive. But the search engine would mix these search results together, with no organization. The engineer or scientist might need to delve through numerous search results, recognize that the search query needs adjustment, type a new search, look through the new search results, see the need for additional adjustment of the query, and so on.

In prior work, we took a first step toward solving this problem by building a cluster-based search engine that demonstrated how to cluster code samples from the LabVIEW forum using unsupervised machine learning [6]. Rather than returning a disorganized mass of search results, that search engine organized results into clusters by topical association, and then presented the clusters to users. The current paper presents an improved search engine that addresses key limitations of that prior work.

The first limitation now addressed is the absence of support for organizing other kinds of online resources (other than forum threads) into sorted clusters. Specifically, the new system described in this paper can organize forum threads, whitepapers, tutorials, and any other form of programming-related document; we say programming-related because, in particular, the new search engine clusters resources using not only code samples attached to resources but also screenshots of visual code attached to resources (as well as other textual attachments associated with resources).

In addition, the second key limitation addressed in the current work is the old system's lack of any labeling for clusters of search results. The enhanced search engine now automatically names each cluster with a human-readable label that describes the content of the cluster.

The remainder of this paper is organized as follows. Section 2 summarizes related work, including the prior version of our search engine in more detail. Section 3 describes our new search engine, including its support for indexing resources, clustering resources, naming clusters, retrieving clusters, and sorting clusters. Sections 4 and 5 present the methodology and results of our evaluation, respectively, which confirmed the effectiveness of our new approach. Finally, Sections 6 and 7 present a discussion of our particular results as well as conclusions and implications for future work.

Related Work

The current work builds upon a large and growing body of work at the intersection of knowledge management and retrieval. This area, for example, includes research exploring how to train supervised machine learning models to detect undesirable user-posted content deserving of removal [7, 8]. Knowledge management and retrieval, as a general research area, is of great potential value for helping programmers because they often use search engines to look for example code that they can use for solving problems that they encounter while programming [9, 10]. Thus, search engines are a reasonable kind of system to create when trying to help programmers with such problems.

Different systems have been provided with the goal of helping programmers to find example code. For instance, some work has delivered tools that cluster pieces of example code to provide users with features for recommendation, search and navigation [11, 12]. Other research, for example, has induced machine-learning features from source code [13] and automatically categorized code in order support navigation [14, 15]. To illustrate such approaches, Strathcona indexes a code repository by parsing the code, extracting names of classes, methods and fields, and issuing recommendations to each programmer by finding code that references classes, methods and fields with similar names to those the programmer is currently editing [12].

Unfortunately, tools like Strathcona typically rely on the object-oriented nature of the code involved. So these systems are beneficial to programmers when using such languages, but engineers and scientists also very frequently use domain-specific languages that lack object-oriented structure [1]. Examples include Excel and LabVIEW, both of which at times present programmers with perplexing problems [4, 16].

Programmers have fewer choices for search engines tailored to help programmers find code written in domain-specific languages. For example, several systems exist to help programmers obtain example code for the Yahoo! Pipes programming environment [17], which supports creation of visual scripts for processing newsfeeds and other lists of online documents and articles. Tools exist to help programmers generate useful keywords when searching for example Pipes code [18] and to help them retrieve example code based on specifications (i.e., mathematical statements indicating what kind of code the programmer wants to find) [19].

Yet even these search engines are of limited value for the interesting case of visual programming languages, the most popular of which among engineers and scientists is that of LabVIEW [5, 20]. In a LabVIEW program, each computation is represented as a node in a graph, and wires connecting nodes show data flow. Nodes wait to execute until needed data arrive. LabVIEW is popular because learners can become minimally productive after only 4-6 hours of instruction, while subsequent training over several weeks quickly builds additional specialized skills [21, 22].

Even programmers of visual languages need online help from time to time, though, and search engines like those described above do not take any advantage of the visual icons that comprise example code linked to online resources. The closest-related system is one that we developed in prior work, which displayed example code in visual groupings based on their mutual similarity [6]: specifically, the system (1) performed unsupervised machine learning offline to identify which code examples were similar to one another, (2) could receive a textual query from a user and retrieve a list of relevant online forum threads that included example code, and (3) display links to those search results by iterating through them and displaying links in visual groupings that corresponded to the machine-learned relationships [6]. In subsequent work, we showed how to filter search results with another machine learning algorithm that inferred which forum resources were more valuable than others [23].

Unfortunately, this prior work only utilized code examples in forum threads, ignoring all other kinds of online support resources (such as whitepapers and tutorials). The earlier version only could utilize source code of examples, which is problematic for LabVIEW specifically because these users commonly attach screenshots of code to their forum posts, rather than the actual source code itself [4]. And, finally, the prior version gave no names to visual groups of code, nor any indication to users about how clusters of search results differed from one another, or even of what topic each cluster addressed. The current paper presents a new search engine that addresses these limitations.

System Overview

The goal of our system is to provide an organized set of answers to questions that engineers, scientists and other users may have when they search through online resources. These resources include forum threads (consisting of a web page where users typically discuss a problem), tutorials, whitepapers, and other pages provided by National Instruments about the LabVIEW programming environment. In addition to its actual textual body, each page can have one or more attachments. Our system recognizes three types of attachments. First, it handles PDF attachments that (from the system's standpoint) essentially consist of more text. Second, it handles code attachments, which in LabVIEW are called "Virtual Instruments" (VIs). Each code attachment consists of one piece of code written in LabVIEW's visual programming language. Finally, each online support resource may have one or more images containing a screenshot of code.

Fig. 1 shows our systems' output for a typical use. A user has provided a query asking how to configure an XY scatter plot. In response, the search engine returned a series of search results (blue links) organized into clusters. Each cluster has an automatically-generated name consisting of 3 words that succinctly describe the cluster's contents. For instance, the first cluster in this example contains search results generally related to configuration of a plot legend. The second cluster contains results related to configuring a plot's axes. Other clusters address other topics, such as how to configure multiple plots on the same graph, how to configure a surface plot, and how to configure a waveform plot.

The system supports the search functionality shown in Fig. 1 through a four-stage process, as described below.

Phase 1: Offline indexing

Prior to receiving any queries from users, all online support resources are copied from the National Instruments support website into a database. This is accomplished with a program that reads the lists of all forum threads, whitepapers, manuals, tutorials, and other kinds of resources available on the website. The program iterates over these lists, retrieves linked pages, parses those pages, and extracts key information from each page (Table 1).

For each resource, the program extracts the title and body text from the HTML of the resource's web page. It follows links in the page to retrieves the binary file for each attachment. The three types of attachments are analyzed as follows:

- For each PDF, it extracts the text therein, which it appends to the main text of the webpage in all analyses that follow.
- For each VI, it invokes the LabVIEW runtime environment to identify what programming primitives appear in the VI. These can include, for example, references to the LabVIEW primitives for loops,

subtraction, string concatenation, opening network sockets, writing to serial ports, reading files, and displaying user interface widgets. Primitives also can be references to other VIs, much like how functions in textual programming languages can be called by other code and thus serve as new primitives extending the core language; consequently, the set of potential primitives in a VI is essentially open-ended.

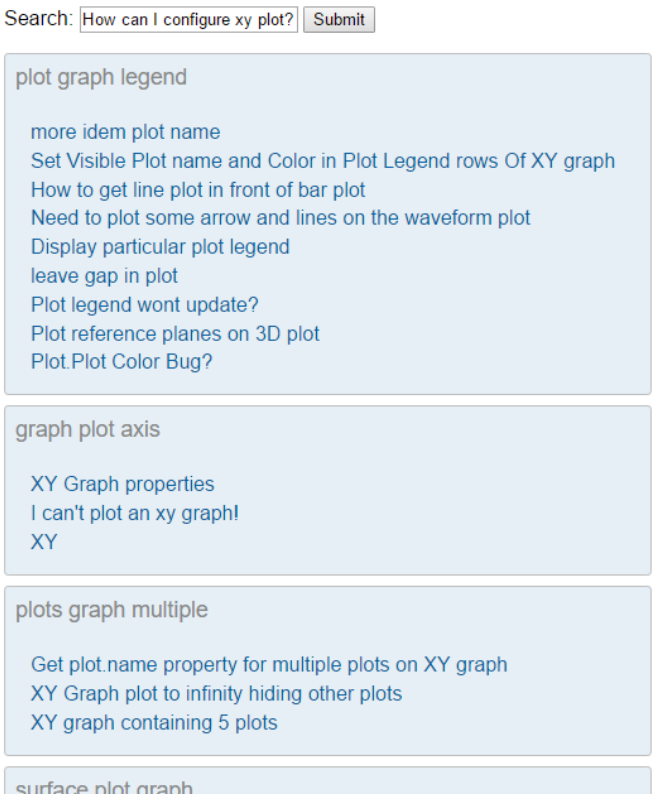

Fig 1: Search engine returns a list of clusters, each of which has an automatically-generated name and contains links to one or more online support resources

Table 1: Information obtained for each online support resource

Item	Description
Title	Retrieved from the HTML title tag of the resource page
Text	Contains the text of the resource's web page, not including header and footer information
Attachments	Includes PDFs, VIs, and images
TFIDF entries	A representation of the resource as a vector in a multi-dimensional space, with one dimension per distinct word or programming primitive
EstValue	An estimate of the likelihood that the resource contains a solution; equals 1 for National Instruments resources and an algorithmically-generated estimate for forum posts

- For each image, which might be a screenshot of a LabVIEW program, our system scans the image to identify what programming primitives appear. It accomplishes this with a brute-force pixel-by-pixel comparison of every contiguous region within the image to every single programming primitive (which always has the same visual appearance in all LabVIEW programs).

Having obtained the data above, the system then constructs a TFIDF vector representing the resource [24, 24]. TFIDF is a very common model for representing search resources. The vector for a resource has one dimension for each distinct token that the resource can contain; in our case, these tokens consist of (a) human-readable words appearing in the resource's title and text, and (b) programming primitives appearing in the resource's VIs and screenshots. Words are treated on a case-insensitive basis, and highly common "stop words" (such as "the") are discarded [24]. That said, the 180,881 resources in our copy of the National Instruments website nonetheless together contain a total of 272,307 distinct tokens, including 214,205 distinct words and 58,102 distinct programming primitives. For a given token w and a given resource r, a TFIDF vector entry $V_r(w)$ takes into account what proportion of r consists of w (i.e., how often w appears relative to all other tokens in r) and how often w appears in all the other resources. Refer to [24] for details on the standard TFIDF model and [6] for details on the representation of LabVIEW resources as TFIDF vectors.

For each resource, our system also computes the estimated "value" or "worth" of each resource. If the resource is a forum thread, the system computes the estimated value of the resource based on a formula that was constructed in previous work using supervised machine learning [23]. Specifically, a forum thread consists of multiple messages, and in our prior work, we used the words within forum thread messages, combined with the sequencing of messages within the thread, to train a model that assigned a number between 0 and 1 for each message in the thread, indicating the likelihood that the message solved whatever problem prompted a user to start the thread. Our system assigns an EstValue to the thread by looking at the estimated value of each message within the thread and then taking the maximum of those values. Other than forum threads, all other resources on the support website were created by National Instruments; for these, the system simply assigns an EstValue of 1.

Phase 2: Clustering

Having represented each resource as a TFIDF vector, the system then uses unsupervised machine learning as in the prior version of our system to organize resources into clusters [6]. It uses the standard kmeans clustering algorithm [24], configured to use 1000 distinct clusters; we configured the system to use this number of clusters because when we previously used the same number of clusters for analyzing forum threads, the resulting 1000 clusters each had an easily-identifiable topic [4], and the threads within clusters tended to have a high mutual overlap of words [6]. The output of this unsupervised learning is thus a set of clusters. Each the centroid of a given cluster c exists in the same multi-dimensional space as the resources' TFIDF vectors; as with individual resources, $V_c(w)$ reflects the extent to which token w appears in the resources of cluster c as well as the extent to which w appears in other clusters.

Phase 3: Cluster naming

For each cluster, the system generates a list of words to name that cluster. The essential insight for picking informative cluster labels is that *good names are distinctive*. A word that appears infrequently in a cluster is unlikely to be accurate. Yet appearing frequently within a cluster is not sufficient, alone, for a word to be useful. One reason is that a word that appears

frequently in a cluster, but that also appears frequently in other clusters, is unlikely to be informative because it says nothing about what makes that cluster distinctive. For example, the word "data" appears in many clusters' resources and is thus a poor label.

We therefore conceptualized the identification of naming words as a statistical problem. For each word w, the cluster-centroid TFIDF entries $V_c(w)$ together constitute a statistical distribution: some clusters will contain a high level of that word, while others will contain relatively little. For a given cluster, we seek words that occur unusually frequently within that cluster, relative to all the other clusters. These are the words that capture what makes this cluster distinctive.

We operationalized this insight with a heuristic based on the t-statistic (which commonly appears in the widely-used t-test) computed as in Eq. 1 below.

$$t_{wc} = \frac{V_c(w) - [\text{mean } V_c(w) \text{ over all clusters}]}{[\text{stddev } V_c(w) \text{ over all clusters}]} \quad (1)$$

Thus, for clusters that have an unusually high value for $V_c(w)$, relative to other clusters, the corresponding t_{wc} is also relatively high. Our system computes t_{wc} for every combination of word and cluster. It then, for each cluster, sorts words by t_{wc} in decreasing order. Note that although $V_c(w)$ is defined over all tokens, regardless of whether w is a human-readable word (drawn from the text of a resource) or a programming primitive (drawn from a VI or a screenshot), all tokens except words are discarded when generating a human-readable name for each cluster.

Phase 4: Retrieving clusters and names

Fig. 2 summarizes the four steps of the final phase, for retrieving clusters and names, which the following few paragraphs describe in detail.

(4A) When a user enters a query into the search engine, the system responds by retrieving a list of clusters and the names of each cluster. A sample query, such as "How can I configure xy plot?" serves to illustrate the process. First, the query is broken into lowercase words, with stop words discarded, leaving a set of query words like {"configure," "xy," "plot"}.

(4B) Clusters containing one or more of these words (a disjunction) are retrieved, and the TFIDF vector for each cluster centroid is used to assess how well the cluster matches the query. For example, cluster x might have V_x("configure") of 0.2 and V_x("xy") of 0.4, giving it a score of 0.3, while another cluster y might have V_y("plot") of 0.2 and V_y("xy") of 0.1, giving it a score of 0.15, so x would appear higher than y in the search results. TFIDF-based sorting such as this is very typical in search engines, including the prior version of our search engine (Section 2); what differs here is the application of this heuristic for the purpose of sorting clusters according to their centroids, rather than solely to individual resources.

(4C) Resources within each cluster are retrieved and sorted based on their individual TFIDF vectors. In this case, rather than sorting solely on a disjunction of query words (as above for clusters), the resources are sorted based on a combination of the disjunction of query words and the conjunction of query words. (Specifically, the average over non-zero TFIDF entries is used in combination with the sum over all TFIDF values, including those that are non-zero.) The rationale for this is pragmatic: Because the number of resources vastly exceeds the number of clusters (by two orders of magnitude), it is feasible to be more selective about which resources are displayed. The TFIDF-based score of each resource is multiplied by the Est Value score for that resource, thereby accounting for both the relevance of the resource to this query and the likelihood that the resource contains valuable information.

(4D) Finally, the names assigned to each cluster are retrieved, in order of decreasing t-statistic. These are static in the sense that they do not depend at all on the specific user query at hand. The system iterates over the clusters and, for each, displays the words that describe it as well as the links within it.

Fig 2: Algorithm for retrieving clusters and their names.

A.	Tokenize the query into words, discarding stop words.
B.	Retrieve an ordered list of clusters:
a.	For each word w in the query, retrieve the list of clusters {c} for which $V_c(w) > 0$, using a case-insensitive match.
b.	For each cluster c in this set, compute $Score_c$ defined as the average of $V_c(w)$ over words for which $V_c(w) > 0$.
	Sort clusters in this set from highest to lowest $Score_c$.
C	For each cluster c, retrieve the list of relevant resources:
a.	For each word w in the query, retrieve the list of resources {r} in c for which $V_r(w) > 0$, using a case-insensitive match.
b.	For each resource r in this set, compute $Subscore_r$ defined as $Subscore_r$ = [mean $V_r(w)$ over words in query for which $V_r(w)>0$] + [sum of $V_r(w)$ over all words in the query] Sort resources from highest to lowest $EstValue_r$ *$Subscore_r$.
c.	Iterate over the list of resources and discard any whose $EstValue_r$*$Subscore_r$ is less than 75% of the highest $EstValue_r$*$Subscore_r$ within this cluster.
D	For each cluster c, retrieve the list of words that describe the cluster:
a.	Retrieve the cluster's naming tokens, sorted in order of decreasing t-statistic.
b.	Retain the top 3 naming tokens that are words with a t-statistic of at least 2.0.

Summary of key points

In summary, the system incorporates an overall approach and specific algorithmic details explored in prior work. These include the representation of resources as TFIDF vectors, the use of unsupervised machine learning to cluster these resources, and the use of a supervised machine learning model to estimate the likelihood that a forum post contains a solution. In addition, the current system goes beyond the previous system in several ways:

1. It supports a wide range of resources beyond forum threads, including diverse National Instruments resources.
2. It can analyze visual code in both source and screenshot form, unlike the earlier version that only supported source code.
3. Instead of generating search results simply by searching over resources and then visually gathering these resources into clusters, as in the old system, the new version truly searches over clusters, which it sorts using a TFIDF-based heuristic.
4. Finally, it uses a heuristic based on the t-statistic to automatically identify and sort words that help to describe the content of each cluster.

Materials and Methods

Our evaluation examined how well the search engine sorted and named clusters of search results. This was accomplished by having a human rate the quality of each output generated by the system. We then compared between different outputs to assess how well the system met our design goals.

Our evaluation was framed by the following scenario. Suppose that a programmer ran into several problems when learning to use LabVIEW. For each problem, the programmer entered a query into the search engine and reviewed the resulting clusters. We were interested in the following research questions:

- **RQ1:** For what percentage of queries does the search engine's top-ranked cluster adequately address the corresponding problem encountered by the programmer?
- **RQ2:** To what extent does the search engine correctly sort clusters, in the sense that more informative clusters appear earlier in the search results?
- **RQ3:** For what percentage of words assigned to cluster names is the search engine's top-ranked word necessary for describing the corresponding cluster?
- **RQ4:** To what extent does the search engine correct sort words, such that more informative words appear earlier in the cluster names?

We answered these research questions using the materials and methods described below.

Materials

In a prior empirical study of the LabVIEW forums, we had identified 14 topics of questions that users frequently had asked and, within these topics, 100 common sub-topics [4]. To select test queries for evaluating our search engine, we selected the 2 most common sub-topics within each of the 2 most common topics of questions. For each of these sub-topics, we selected one post whose text contained a clearly expressed question. This procedure yielded the following 4 test queries shown in Table 2.

Table 2: Results of testing the search engine with sample queries

Topic	Sub-topic	Test query
Hardware I/O	proportional-integral-derivative (PID) controller	"How to create PID controller?"
	reading and writing serial port data	"How to save a datas [sic] from serial port?"
User interface	fine-tune the display of data on XY scatter graphs	"Is it possible to use only the graph palette of the first graph to control the zoom, cursors, etc... on both graphs?"
	manipulate user interface panels dynamically	"Can I embed a Front-Panel Vi inside of a tab control?"

We copy-pasted each of the 4 test queries into the search engine. For each query, we accepted the first 3 clusters from the search results and, for each cluster, filtered out all except the first 3 online support resources. (Thus, the results below are conservative in the sense that clusters might generally contain more information in practice than what our analysis with these restricted clusters suggests.) For these 3*4=12 clusters, we obtained 3 (sorted) words to name each cluster.

Methods

One of our co-authors then evaluated the quality of the system outputs. To minimize the risk of confirmation bias, this co-author was entirely uninvolved in the generation of search results. For each test query, the clusters were randomly sorted so that this human evaluator could not tell in what order the search engine had sorted clusters for the corresponding test query; likewise, the words for each cluster were randomly re-sorted to conceal the order in which the search engine had sorted them. To assure face validity, this co-author has several years of experience with LabVIEW programming, in addition to many years of experience with other programming languages.

This rater assigned each cluster a rating, indicating the extent to which the cluster contained information answering the corresponding test query, as follows:

4 = The cluster fully answers the question (test query) on the stated sub-topic & topic.

3 = The cluster partially answers the question.

2 = The cluster is related to the question and topic but does not answer the question.

1 = The cluster is related to the topic in general but is not relevant to this question.

0 = The cluster is related neither to the topic nor to the question.

To answer RQ1, we then computed the proportion of clusters in the first search position that were rated as answers (i.e., rating ≥ 3). We likewise computed the proportion of clusters in the second search position rated as answers, and the proportion of those in the third search position.

In addition to assigning each cluster a rating, the evaluator assigned each cluster a rank, essentially indicating a "correct" ordering of the 3 clusters for each test query (where a rank of 1 indicates the best, 2 is in between, and 3 is lowest). The correlation was then computed, using a linear regression model, between this human-assigned ranking and the automatically-assigned cluster position in the search results, thereby answering RQ2.

Evaluation of the word-level data proceeded in an analogous fashion. The human rater assigned to each automatically-generated word a rating, indicating how well the word described the corresponding cluster, as follows:

4 = Complete: The word, by itself, fully describes this cluster.

3 = Essential: The word should appear in any description of this cluster.

2 = Informative: The word provides useful information but could be omitted.

1 = Relevant: The word is topically related but does not describe this specific cluster.

0 = Irrelevant: The word is entirely unrelated to this cluster.

To answer RQ3, we computed the proportion of top-ranked words that were rated as essential (i.e., rating ≥ 3), and we likewise computed the proportion among the second-ranked words and the third-ranked words.

Finally, to answer RQ4, the evaluator assigned each word a rank, indicating the "correct" ordering of words for each corresponding cluster, and we used linear regression to compute the correlation between this human-assigned ranking and the position automatically assigned to words by the search engine.

Results

Sorting of clusters (RQ1 and RQ2)

The search engine generally did an effective job of sorting clusters in terms of relatedness to the test queries (Fig. 3). Averaged across the 4 test queries, the 4 clusters in the first position of search results had the highest proportion rated as answers (75%), the 4 in the middle position had a lower proportion rated as answers (25%), and the 4 in the lowest position had the lowest proportion rated as answers (0%). The fact that none of the clusters in the final position were rated as answers suggests that clustering more than one or two clusters of search results may be unnecessary in practice; this contrasts with a typical search engine, which may need to return one or more pages full of search results to meet users' needs.

The search engine's sorting of clusters was compared to the ranking assigned by the human evaluator. The linear regression of the human-assigned cluster ranking versus automatically-assigned cluster position was significant overall ($P<0.05$, $F(1,10)=6.41$, $N=12$, $R^2=0.39$), as was the coefficient of 0.63±0.25 for the slope of the fit ($P<0.05$, $t=2.53$). Thus, on average, the human-assigned cluster ranking changed by 0.63 for every step of cluster position over the search results. The R^2 value indicated that more than a third of the variance in human-assigned rankings was captured by the cluster position.

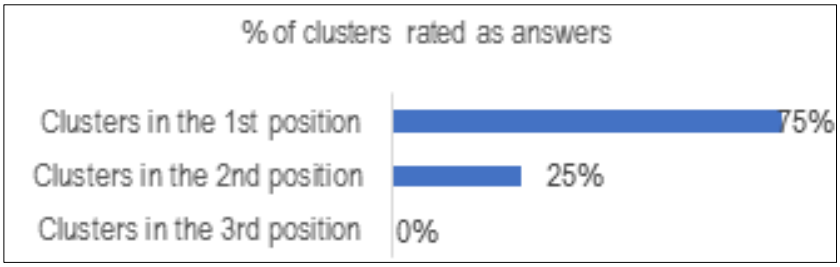

Fig 3: Clusters appearing earlier in search results had a higher probability of being rated as an answer (i.e., rating ≥ 3).

Sorting of words to name clusters (RQ3 and RQ4)

The system selected and sorted words that accurately described the content of clusters (Fig. 4). Averaged across the 12 clusters returned for the 4 test queries, the 12 words in the first position of cluster names had the highest proportion rated as essential for naming their respective clusters (67%), a slightly lower proportion of the 12 words in the middle position of cluster names were rated as essential (58%), and almost no words in the last position of cluster names were rated as essential (8%).

The search engine's sorting of words for naming clusters was compared to the word ranking assigned by the evaluator. The linear regression of human-assigned word ranking versus automatically-assigned word position was significant overall ($P<0.01$, $F(1,34)=14.12$, $N=36$, $R^2=0.27$), as was the coefficient of 0.54±0.14 for the slope of the fit ($P<0.01$, $t=3.76$). In other words, the human-assigned ranking of words on average changed approximately half as quickly as did the automatically-assigned word position within cluster names, and, as with cluster-level results, approximately of one-third of the variance in human-assigned rankings was captured by the word position.

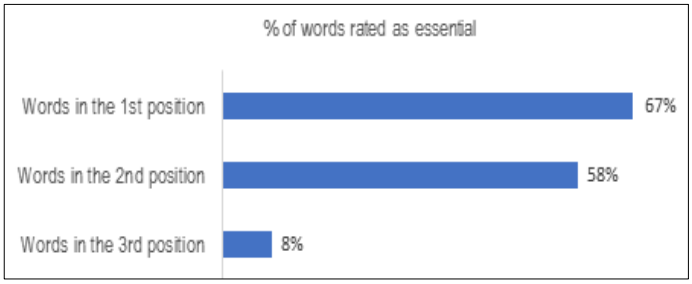

Fig 4: Words assigned a higher position by the system had a higher probability of being rated as essential for naming clusters (i.e., rating ≥ 3).

Discussion

The results provide answers to the research questions that drove the evaluation.

(RQ1) Clusters in the top position usually contained an answer to the test query. Approximately 75% of the time, the search engine could have returned just one cluster of results and thereby provided a partial or full answer to the programmer's query, particularly given that the cluster in the second slot rarely provided even a partial answer (25%).

(RQ2) Clusters' human-assigned rankings generally changed in correspondence to clusters' position in search results. This indicated that the search engine's heuristics for sorting clusters were generally consistent with the evaluator's perception of how well the clusters answered the test queries.
(RQ3) Words in the top position within automatically-assigned cluster names were usually rated as essential. Note, however, that words in the second position were also rated essentially approximately half of the time. Therefore, returning two names per cluster could provide an effective means of naming clusters.
(RQ4) Words' human-assigned rankings typically changed in correspondence to words' position in cluster names. Thus, the system's statistical heuristics for sorting names were generally consistent with the evaluator's perception of how well words described clusters.

Conclusions

These results indicate the search engine succeeded in sorting and naming clusters of online resources for a visual programming language commonly used by scientists and engineers. These resources can include forum posts, tutorials, whitepapers, and a wide range of other documents. These results highlight the potential opportunity for integrating our approach into existing search engines aimed at supporting programmers as they learn to use languages like LabVIEW. Future work could build on this research in several areas.
First, deployment of the search engine would facilitate the impact of the search engine on programmers' success. For example, observing scientists and engineers as they use the search engine in a laboratory setting would reveal the extent to which the search engine enables them to complete specified tasks more quickly or more successfully. Deployment in the field would uncover strengths and weaknesses of the system in practice, relative to other available kinds of search engines and support systems.
Second, adaptation of the search engine could aid in meeting the needs of targeted user populations beyond the typical engineer, scientist or other end user. For instance, technical support personnel charged with helping others adopt LabVIEW might value a form of the search engine that provides additional features for curating search results. These users could include technical support personnel who work for National Instruments (who may wish to collect certain clusters of resources frequently needed when helping LabVIEW customers), and they could include external partners such as university faculty (who may wish to collect certain clusters of resources frequently needed for students in their courses).
Finally, the fact that it is possible to generate many topically related clusters suggests the existence of relationships among these clusters that could potentially support personalized learning. For instance, a system could track, for each user, the extent to which that person has already read about (or potentially used) resources from each cluster. In response to subsequent queries from that user, the system could then preferentially highlight clusters that are topically-related to the ones that the user already has learned from during prior interaction with the system. In this way, perhaps aided by some sort of implicit dependency graph among the clusters, the system could gradually aid engineers and scientists in incrementally mastering a succession of online support resources to build proficiency and success.

Acknowledgements

National Instruments funded this research and gave permission to download the contents of the online forums. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of National Instruments.

References

1. Jones M, Scaffidi C. Obstacles and opportunities with using visual and domain-specific languages in scientific programming. IEEE Symposium on Visual Languages and Human-Centric Computing. 2011, 9-16.
2. Ko A, Abraham R, Beckwith L, Blackwell A, Burnett M, Erwig M. The state of the art in end-user software engineering. ACM Computing Surveys. 2011; 43(3):1-44.
3. Ko A, Myers B, Aung H. Six learning barriers in end-user programming systems. IEEE Symposium on Visual Languages and Human-Centric Computing. 2004, 199-206.
4. Scaffidi C. What training is needed by practicing engineers who create cyberphysical systems? Euromicro Conference on Software Engineering and Advanced Applications, 2015, 298-305.
5. Travis J, Kring J. LabVIEW for Everyone: Graphical Programming Made Easy and Fun, Prentice Hall Publishers, 2006.
6. Scaffidi C, Chambers C, Surisetty S. A code-centric cluster-based approach for searching online support forums for programmers. IEEE International Conference on Machine Learning and Applications. 2015, 1032-1037.
7. Almeida T, Alberto T. Learning to block undesired comments in the blogosphere. IEEE International Conference on Machine Learning and Applications. 2013, 261-266.
8. Reynolds K, Kontostathis A, Edwards L. Using machine learning to detect cyberbullying. IEEE International Conference on Machine Learning and Applications. 2011, 241-244.
9. Brandt J, Guo P, Lewenstein J, Klemmer S. Opportunistic programming: How rapid ideation and prototyping occur in practice. 4th International Workshop on End-User Software Engineering. 2008, 1-5.
10. Brandt J, Guo P, Lewenstein J, Dontcheva M, Klemmer S. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. ACM SIGCHI Conference on Human Factors in Computing Systems. 2009, 1589-1598.
11. Chatterjee S, Juvekar S, Sen K. Sniff: A search engine for Java using free-form queries. Fundamental Approaches to Software Engineering. 2009, 385-400.
12. Holmes R, Murphy G. Using structural context to recommend source code examples. IEEE Conference on Software Engineering. 2005, 117-125.
13. Bontcheva K, Sabou M. Learning ontologies from software artifacts: Exploring and combining multiple sources. Workshop on Semantic Web Enabled Software Engineering, 2006.
14. Asuncion H, Asuncion A, Taylor R. Software traceability with topic modeling. IEEE Conference on Software Engineering, 2010, 95-104.
15. Landauer T, Dutnais S. A solution to Plato's problem: The latent semantic analysis theory of acquisition,

induction, and representation of knowledge. Psychological review. 1997; 104(2):211-240.
16. Chambers C, Scaffidi C. Struggling to excel: A field study of challenges faced by spreadsheet users. IEEE Symposium on Visual Languages and Human-Centric Computing. 2010, 187-194.
17. Pruett M. *Yahoo! Pipes*, O'Reilly, 2007.
18. Riabov A, Boillet E, Feblowitz M, Liu Z, Ranganathan A. Wishful search: Interactive composition of data mashups. ACM International Conference on the World Wide Web. 2008, 775-784.
19. Stolee KT, Elbaum S. Toward semantic search via SMT solver. ACM International Symposium on the Foundations of Software Engineering. 2012; 25:1-4.
20. Kodosky J, MacCrisken J, Rymar G. Visual programming using structured data flow. IEEE Workshop on Visual Languages. 1991, 34-39.
21. Garcia-Cerezo A, *et al.* Using LEGO robots with LabVIEW for a summer school on mechatronics. IEEE International Conference on Mechatronics, 2009, 1-6.
22. Sherry R, Lord S. LabVIEW as an effective enhancement to an optoelectronics laboratory experiment. Frontiers in Education Conference, 1997, 897-900.
23. Scaffidi C. Mining online forums for valuable contributions. Iberian Conference on Information Systems and Technologies, 2016, 1-6.
24. Chakrabarti S. Mining the Web: Discovering Knowledge from Hypertext Data, Elsevier, 2002.
25. Wu H, Luk R, Wong K, Kwok K. Interpreting TF-IDF term weights as making relevance decisions. ACM Transactions on Information Systems. 2008; 26(3):13.