LemonAid: Selection-Based Crowdsourced Contextual Help for Web Applications

Parmit K. Chilana, Amy J. Ko, Jacob O. Wobbrock
The Information School | DUB Group
University of Washington
Seattle, WA 98195 USA
{pchilana, ajko, wobbrock}@uw.edu

ABSTRACT

Web-based technical support such as discussion forums and social networking sites have been successful at ensuring that most technical support questions eventually receive helpful answers. Unfortunately, *finding* these answers is still quite difficult, since users' textual queries are often incomplete, imprecise, or use different vocabularies to describe the same problem. We present *LemonAid*, a new approach to help that allows users to find help by instead selecting a label, widget, link, image or other user interface (UI) element that they believe is relevant to their problem. LemonAid uses this selection to retrieve previously asked questions and their corresponding answers. The key insight that makes LemonAid work is that users tend to make similar selections in the interface for similar help needs and different selections for different help needs. Our initial evaluation shows that across a corpus of dozens of tasks and thousands of requests, LemonAid retrieved a result for 90% of help requests based on UI selections and, of those, over half had relevant matches in the top 2 results.

ACM Classification: H.5.2 [**Information interfaces and presentation**]: User Interfaces. *Graphical user interfaces*.

General terms: Design, Human Factors.

Keywords: contextual help; crowdsourced help; software support

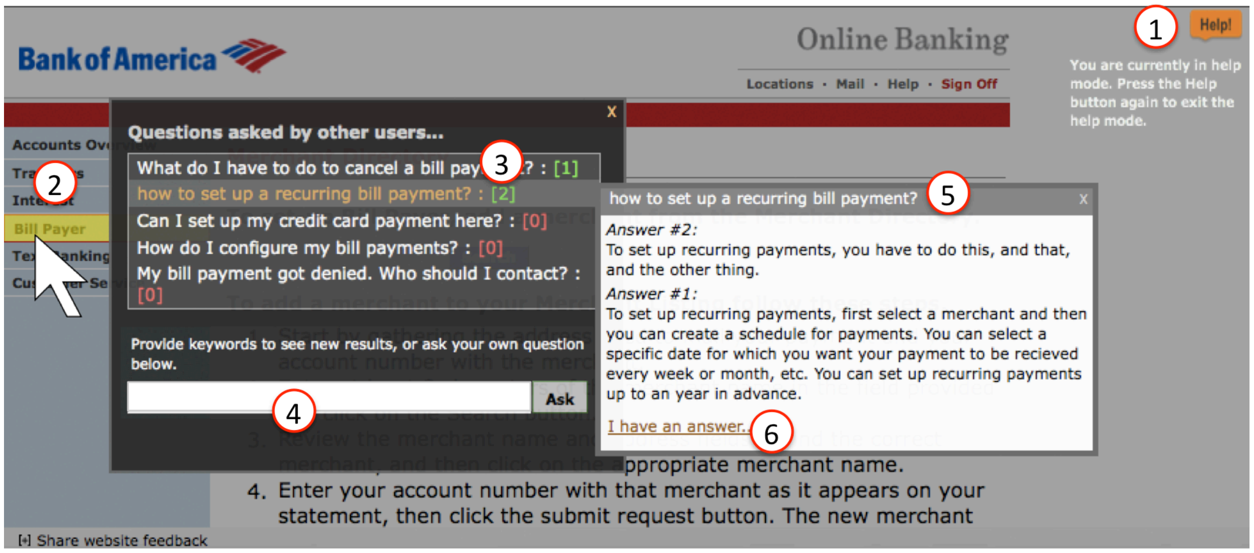
INTRODUCTION

Millions of users on the web struggle to learn how to use and configure applications to meet their needs. For example, customers must decipher cryptic error messages after failed e-banking transactions, office workers wrestle with adding attachments to their company wikis, and new users may have to interpret complex privacy settings on social networking sites. As today's web applications become more dynamic, feature-rich, and customizable, the need for application help increases, but it is not always feasible or economical for companies to provide custom one-on-one support [34].

Among the long history of approaches to software help, perhaps the most powerful approach is *crowdsourced* help. With crowdsourced help (*e.g.*, [19,22,29]), users can help each other answer questions in discussion forums, mailing lists, or within their online social networks. Such resources reinforce the social nature of technical support that users tend to prefer [29,33] and companies also benefit, as they have to expend fewer resources on support.

While crowdsourced help is powerful at *generating* answers to help questions, *locating* useful answers from past discussions can be difficult. First, questions and answers are scattered across different resources: a user may post a technical help question on her social network, unaware that a similar question had already been answered on the application's forum site. Second, even if a user finds a discussion that potentially has the answer, the answer may be buried deep within long conversation threads that span multiple pages. Even though recent Q&A sites have incorporated strategies for promoting the best responses to the top of the list, users often cannot find these threads in the first place because users' queries tend to be not only incomplete and imprecise [3], but also plagued by the classic *vocabulary problem* [11], where different users provide different words to describe the same goal (*i.e.*, "add a photo" *vs.* "insert an image"). While search engine algorithms can be used to mitigate some of the challenges in natural language retrieval, the onus is still on users to translate their help needs and problem contexts into keywords that result in an effective search.

We present *LemonAid*, a new approach to technical help retrieval that allows users to ask for help by selecting a label, widget, link, image or other user interface (UI) element, rather than choosing keywords. With LemonAid, help is integrated directly into the UI (as in Figure 1) and users can ask questions, provide answers, and search for help without ever leaving their application. The key insight that makes LemonAid work, one supported by our formative studies, is that users tend to make similar selections in the interface for similar help needs and different selections for different help needs. This tight coupling of user needs to UI elements is central to LemonAid's effectiveness, reducing unnecessary variation in users' queries. In fact, our initial evaluation of LemonAid's retrieval, based on a corpus we created consisting of over 2,700 UI selections from over 500 users



You are currently in help mode. Press the Help button again to exit the help mode.

Questions asked by other users...

What do I have to do to cancel a bill pay...? : [1]
how to set up a recurring bill payment? : [2]
Can I set up my credit card payment here? : [0]
How do I configure my bill payments? : [0]
My bill payment got denied. Who should I contact? : [0]

Provide keywords to see new results, or ask your own question below.

how to set up a recurring bill payment?

Answer #2:
To set up recurring payments, you have to do this, and that, and the other thing.

Answer #1:
To set up recurring payments, first select a merchant and then you can create a schedule for payments. You can select a specific date for which you want your payment to be recieved every week or month, etc. You can set up recurring payments up to an year in advance.

I have an answer.

appropriate merchant name.
4. Enter your account number with that merchant as it appears on your statement, then click the submit request button. The new merchant

Figure 1. The LemonAid user interface: **(1)** the help button, which invokes the help mode; **(2)** a user's selection; and **(3)** questions relevant to the user's selection (the brackets contain the number of answers available for each question). **(4)** A search box where users can provide keywords to filter the results or ask a new question if they do not see a relevant question. **(5)** Answers linked to the selected question and **(6)** a link that allows a user to submit an answer for the selected question.

on Mechanical Turk, showed that on average, LemonAid retrieved a relevant result for 90% of help requests based on UI selections and, of those, over half had relevant matches in the top 2 results.

LemonAid works exclusively with standard DOMs, makes no assumptions about how an application's back or front end is implemented, does not require the modification of application source code, and does not require the use of a specific UI toolkit. It simply operates as a layer above an application's UI. The only work that developers must do to integrate LemonAid into their site is extract text labels appearing in the UI from a web application's code and include the framework in their client-side deployment. We have tested the LemonAid framework with a few web applications and found that the integration work is minimal.

While LemonAid's approach is influenced by recent work on helping programmers debug code in-context (*e.g.*, [5,14,18]), the novelty lies in providing a contextual help framework for *end-users* to resolve issues through crowdsourcing. Our larger vision is that software teams will be able to use this repository of contextually reported issues for better understanding potential usability problems and user impact. In this paper, we contribute:

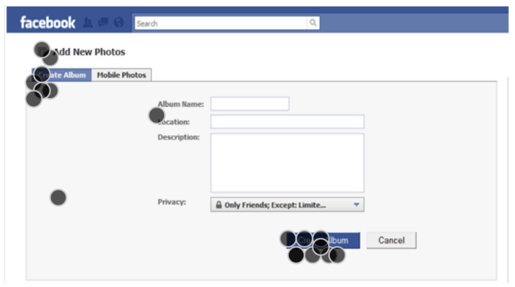
- A new selection-based querying interface and a question and answer authoring interface allows users to generate help content and find help without leaving the application.
- A new help retrieval algorithm that leverages contextual information and user interface selections to retrieve relevant help content.
- A framework that allows web developers to easily integrate LemonAid within a web application.
- An initial evaluation that demonstrates the feasibility of LemonAid in retrieving relevant matches for a crowdsourced corpus of selections.

USING LEMONAID TO FIND HELP

LemonAid allows users to find application help in-context by selecting UI elements, including labels, widgets, links, and images that they believe are relevant to their help request, question, or problem. For example, consider Figure 1, which shows an example scenario in which Bob, a teacher who occasionally does online banking, wants to pay his electricity bill by setting up a monthly online payment through his bank. Bob has checked his account balance a few times using the bank's online application, but he is not familiar with all of the other application features. He clicks on the "Bill Payer" option and it opens up a new page with more information, but now Bob does not know what to do. Normally, Bob would call his tech-savvy friend for help, but since it is late, Bob needs to find a solution on his own.

Bob clicks on the LemonAid "Help" button at the upper right of the page (Figure 1.1) and the screen dims, indicating that the application has entered help mode. LemonAid fades the user interface, indicating to Bob that the meaning of a "click" has changed. As Bob moves his mouse cursor over the page, he notices that words and objects under his cursor are highlighted in yellow, indicating that they are clickable. Bob selects the "Bill Payer" label, as he thinks it is most relevant to his problem (Figure 1.2). LemonAid displays five questions that it believes are relevant to his selection (Figure 1.3), all which have been asked by other users who had previously selected the same or similar labels. Bob immediately notices the 2nd question, "how to set up recurring payments," and sees it has 2 answers (indicated by the number in brackets). Bob clicks on the question and sees that the first answer is what he needs (Figure 1.5).

While he is logged in, Bob also wants to update his phone number in the e-banking system. He again goes into LemonAid's help mode and this time, he clicks on a tab



Scenario. You are trying to add photos from a recent trip to your Facebook page for the first time. You heard from a friend that it's easy to upload multiple photos from your hard drive, but when you arrived at this page, you did not see any such option. You are wondering if you came to the right place and what you should be doing next to upload your photos.

Figure 2. Aggregate results from a task in our formative study and its corresponding scenario.

labeled "Account Profile." The help system now shows a much longer list of relevant questions (not shown), as there are many features relevant to this label. Bob does not want to read all of them, so he starts typing into the search box (Figure 1.4) and notices that the question list updates. Bob now only sees two questions (not shown), the first of which is explicitly about adding phone numbers.

LEMONAID DESIGN AND ARCHITECTURE

The main component of LemonAid is a retrieval engine that produces a ranked list of relevant questions in response to a user's selection on a page. Selections are captured as a DOM element and other context (described next) and then matched against the existing set of questions stored in LemonAid's repository of application-specific questions and answers. In this section, we (1) describe the contextual data that LemonAid captures, (2) explain how LemonAid represents questions, answers, and users' selections, and (3) explain LemonAid's retrieval algorithm.

Capture of Contextual Data

When a user makes a selection, LemonAid captures information about the target DOM object and underlying HTML, which we will call *contextual data*. There was a variety of contextual information that LemonAid could gather (as explored in other recent help approaches [9]). But, we focused on identifying contextual data that would be useful in discriminating between different help problems within the UI from the user's perspective.

We designed a formative study that presented 20 participants with a series of 12 screen shots from popular web applications. Each screen shot conveyed a problem scenario consisting of a textual description and a printout of where a problem was encountered, as shown in Figure 2. All of the problem scenarios were taken from real questions from help forums for these web applications. During the study, we asked participants to pretend that they had a "magic wand" that they could use to point anywhere on the interface to get help and to indicate their selection with a physical sticker. An example scenario and its results are displayed in Figure 2.

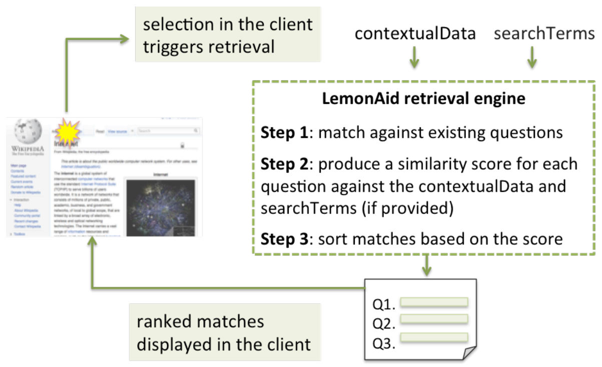
There were two major findings from the study. First, participants tended to select labels in the UI that they believed were *conceptually relevant* to the help problem. Most of these keywords were application-specific labels or headings (*e.g.*, 15 of 20 participants selected the "*Create Album*" keywords in the scenario in Figure 2). Second, when no label appeared relevant, participants selected UI elements that were similar in terms of their visual appearance and location on the screen, with a bias towards the top-left. These findings suggested that LemonAid could determine similarity between selections largely based on the text on UI labels, and leverage additional attributes of the selected DOM object such its layout position and appearance.

Based on these results, we designed LemonAid to capture the three contextual details listed in Table 1. When a user clicks on a region of a page, LemonAid first determines the topmost DOM node (based on *z-order*) under the user's cursor. From this, it extracts the tag name of the selected node (`nodeType`). It also extracts the XPath string representing the sequence of tag names and child indices that indicate the path from the root of the DOM tree to the selected node (`nodeXPath`). Finally, it extracts all of the text node descendants of the selected node, concatenated into one string, using the standard *innerText* or *textContent* property of the selected DOM node, depending on the browser (`nodeText`). If the selected node is an image and includes an *alt* attribute, this text is also concatenated to `nodeText`. While we also considered using HTML *id*s or *class*es, since they are also related to appearance and layout, they are often dynamically generated between user sessions and thus not useful for representing questions shared by users.

Since the text labels on DOM elements could potentially be user-generated and privacy-sensitive, LemonAid only stores the `nodeText` if it is a known *UI literal*. UI literals include any string that is explicitly defined in the application source code or any whitespace-delimited string that appears in application resource files, such as localization files. A UI literal may represent labels on UI widgets, headings, or error messages, among other application-specific strings. Every time a user makes a selection, LemonAid compares the `nodeText` of the selected node against a whitelist of known application UI literals to determine whether or not to store the captured `nodeText`. For example, for a heading element where the text is "Account Settings," the `nodeText` would only be stored as part of the contextual data if "Account Settings" was found in the list of application UI literals. In contrast, if the selected text were a container that included a

Attribute	Description	Example
nodeText	Visible text on the selected DOM node	"Bill Payer"
nodeXPath	The XPath uniquely identifying the selected DOM node in the page	/HTML/BODY/TABLE/TBODY/TR[5]/TD
nodeType	The HTML tag of the selected DOM node (i.e., DIV, TABLE, BUTTON, etc.)	TD

Table 1: Contextual data captured in a user selection with example from the bill payer scenario in Figure 1.



selection in the client triggers retrieval
contextualData
searchTerms
LemonAid retrieval engine
Step 1: match against existing questions
Step 2: produce a similarity score for each question against the contextualData and searchTerms (if provided)
Step 3: sort matches based on the score
ranked matches displayed in the client
Q1.
Q2.
Q3.

Figure 3. The retrieval engine uses contextual data from the user's selection and any search terms provided by the user to produce a ranked list of relevant questions.

user's username, this would likely *not* be in the whitelist and would therefore not be included in `nodeText`. (We describe LemonAid's built-in functionality for extracting UI literals in a later section discussing LemonAid's integration steps.)

Questions, Answers, and Users' Selections

Once the user makes a selection and LemonAid captures the data in Table 1 to represent it, the data is used as a query to LemonAid's repository of questions. To explain this retrieval, we first define how questions and answers are stored. Each *Question* submitted by a user through LemonAid's question box (Figure 1.4) includes a unique id for each question in the set (*id*); the `nodeXPath`, `nodeType`, and `nodeText` described in Table 1 (*contextualData*), and the optional question text provided by the user in the text field in Figure 1.4 (*questionString*). Since the focus of this paper is on help retrieval, and not help authoring, LemonAid's *Answers* are basic, mimicking the kind of answers found in discussion forums: each *Answer* has a unique *id*, an *answerString*, which stores the text provided in the field in Figure 1.6, and a *questionID*, linking it to the question for which the answer was provided. *Questions* may have multiple *Answers*.

A *UserSelection* in LemonAid consists of the contextual data captured from the user's selection (as described in Table 1) and optional `searchTerms`, which store the text provided by the user in the search input field (Figure 1.4). A *UserSelection* initially generates a query consisting only of the contextual data. Each keystroke in the search input field generates a new *UserSelection* and updates the retrieved results using an *auto-suggest* interaction.

Ranked Retrieval of Matching Questions

To retrieve help, LemonAid utilizes a relevance ranking approach leveraging *contextualData* and optional *searchTerms* using the process shown in Figure 3. The retrieval algorithm takes a *UserSelection* and compares it to each previously asked *Question* in the application's repository, producing a *similarity score* between 0 and 1 for each *Question*, with 1 being the best match. LemonAid then presents the matching questions in descending order of score. The score, which is computed as in Figure 4, is a combination of factors, including a `contextScore` based on a weighted sum of the three scores in Table 2 and a `textSimilarityScore` if the user provided `searchTerms`. The next two sections describe these scores in detail.

```
findMatchingResults(UILiteral, nodeType, nodeXPath, searchTerms)
returnValue: a sorted resultSet containing matching questions from set of existing questions.

for each question Qi in the set of existing questions Q
    nodeTextScore = Qi.nodeText contains UILiteral ? 1 : 0
    xPathScore = a percentage computed by comparing the overlap in
    Qi.XPath and nodeXPath of the current selection;
    typeScore = Qi.nodeType string equals nodeType ? 1 : 0
    contextScore = .7 nodeTextScore + .2 xPathScore + .1 typeScore
    if contextScore > 0.25
        add Qi to the resultSet
    if searchTerms is non-empty
        compute textSimilarityScore with full text tf–idf weighting
        if textSimilarityScore > 0
            add Qi to resultSet
if searchTerms is non-empty
    sort resultSet by textSimilarityScore, then contextScore
else
    sort resultSet by contextScore
return resultSet
```

Figure 4. Pseudocode for retrieval algorithm.

Similarity based on context

As discussed above, our formative study suggested that similarity between selections could largely be based on the text on UI literals. Therefore, the primary factor in the `contextScore` is the `nodeTextScore`, which is 1 if the `nodeText` of a selection contains (ignoring case) the `nodeText` of the *Question* being compared and 0 otherwise. With this approach, LemonAid is able retrieve a match related to a specific item in the container (*i.e.*, a navigation menu item) even if the user's selection was the container itself. This factor is given a weight of 0.7, since it was the most important factor in our formative studies.

The 2nd factor in the `contextScore` is the `XPathScore`, which captures similarity in layout and position identified in our formative study. Although XPaths can change as UI layouts evolve over time [2,4], many menus and header items on a page stay relatively the same or have only slight layout differences over time. Therefore, this score is a measure of the percent node overlap between the `nodeXPath` of the query and a *Question*'s `nodeXPath`. We compute this by starting from the root and doing a node-by-node string comparison from root to leaf, incrementing the score by 1 every time there is a match, and stopping when there is no match or the last node of the shorter path is reached. We divide the final sum by the length of the longer XPath to get a percentage. (For example, the overlap between `HTML/BODY/DIV/` and `HTML/BODY/DIV/DIV/P/` is 3/5 or 60%). Because location and position were only a factor in a

Constituent	Similarity Score	Weight
nodeTextScore	string contains (1 or 0)	0.7
XPathScore	% node overlap between XPaths [0,1]	0.2
nodeTypeScore	string equals (1 or 0)	0.1

Table 2. Weights for contextual data.

minority of the scenarios in our formative study, the `nodeXpath` score has a weight of only 0.2.

The 3rd and final factor in the `contextScore` compares the `nodeType` of the selected node to that of the potentially matching *Question*. This factor accounts for both appearance similarities, while also helping with situations where multiple UI elements share the same text label, such as a button and a heading with the same words. The `nodeTypeScore` is 1 if the labels of the selection and the Question are equivalent and 0 if not. Because ties were rare, and appearance was only rarely a factor in our formative study, we only give `nodeTypeScore` a small weight of 0.1.

After `contextScore` is computed, the algorithm in Figure 4 includes a *Question* in the results set if its score is above 0.25. This threshold was selected because it implies that there is no `nodeText` match, but there is a strong match between the `nodeXPath` and `nodeType`. Even though this type of match is weaker than one based on `nodeText`, it is useful for cases where a question may be attached to a container or non-UI literal text (*e.g.*, user-generated content). Since `nodeText` similarity is not relevant in such cases, the `nodeXPath` and `nodeType` similarity can still be used as (weak) indicators of relevant questions.

Similarity based on search keywords

If the user provides `searchTerms` in the field in Figure 1.4, the algorithm in Figure 4 also computes a `textSimilarityScore`. It does this by comparing a query's `searchTerms` with the whitespace-delimited words in each existing *Question*'s *questionString*. To compare the similarity between these keywords, we created a search index on each *questionString* and used a standard full-text search feature based on the vector space model [27]. The similarity score is computed using the term frequency–inverse document frequency (*tf–idf)* weighting approach in information retrieval. The main idea of *tf-idf* is that terms that occur frequently in the target document (in our case, a question in the repository of previously asked questions), but less frequently in the whole document collection are the useful terms. The weight of these terms is a combination of term frequency within the target document and its frequency across all documents. Each *Question* that matches the user's *searchTerms* is included in the result and sorted in descending order based on the `textSimilarityScore`. This result set is then sorted by `contextScore` of each question against the *UserSelection*.

INTEGRATING LEMONAID INTO WEB APPLICATIONS

One of the strengths of LemonAid's simplicity is that it can be easily integrated into an existing website with minimal modification to the site itself.

First, site administrators choose an ID to uniquely identify their application-specific help information in the third party server. Next, administrators can either provide a URL to their main source directory or run a script provided by LemonAid to extract UI literals from a site's code and localization files. From this, LemonAid generates a CSV file containing the list of literals and stores it alongside the question repository. LemonAid uses a simple algorithm for finding string literals in commonly used web programming languages, looking for sequences of characters delimited by single ('') and double ASCII quotes (""). While this approach does not account for UI literals that may be dynamically generated, it covers a large range of UI literals defined at design time. While this extraction approach may generate false positives (extracting strings that do not appear in the user interface), these non-UI literals are not visible to the user and hence not selectable anyway. Furthermore, site administrators have full control in editing the CSV file containing string literals from their source code.

Finally, site administrators include a few lines of JavaScript on all of their web application's pages, just as with analytics services such as *Google Analytics*. Doing so links the LemonAid source code to the web application, and makes LemonAid functional on that page. The interface shown in Figure 1 is an example implementation of LemonAid on the static version of *Bank of America*'s Bill Payer site. The UI literals were obtained by manual screen scraping since we did not have access to Bank of America's source.

The current implementation of LemonAid sets up a basic infrastructure through which anyone can anchor questions and answers on the underlying application's UI literals. Site administrators may have different needs in terms of managing the Q&A and the related community of users. For example, some web applications are already controlled by user authentication and it may be just a matter of integrating LemonAid with the existing user accounts on the site. Another approach may be the use of social networking plugins to facilitate communication among users within their social network. In other cases, administrators may want to restrict answer authoring to the company's support personnel and may want to store the help data locally.

EVALUATION

At the core of LemonAid is a retrieval engine that produces a ranked list of questions relevant to a user's selection and optional search terms. As explained above, although users' natural language descriptions of the same problem may differ, users tend to make the same selections in the UI for a given problem. Thus, to assess the effectiveness of LemonAid's retrieval algorithm, we focused our evaluation on answering the following question: *across a corpus of help problem scenarios, how effective is LemonAid at retrieving a relevant question asked by another user using only the current user's selection*? To operationalize this, we measured the rank of the first retrieved *Question* that regarded an identical help problem (described next), using only the contextual data from the *UserSelection*.

Developing a Crowdsourced Corpus

To perform this assessment, we first needed a large corpus of LemonAid help selections. Since we did not have a site with a large number of users to which LemonAid could be deployed (as website owners viewed the adoption of the tool without evidence of its efficacy as risky), we developed a corpus using a *simulated* community of users through Amazon's Mechanical Turk (mTurk) platform [36]. mTurk is an online marketplace where workers receive micro payments for performing small tasks, termed Human Intelligence Tasks (HITs). Recently, mTurk has become a popular way for researchers to recruit a large number of participants for small tasks [17,25,30]. We used mTurk to have hundreds of web users read a detailed help scenario and perform a LemonAid help request by selecting a UI element and providing a question relevant to the scenario.

To ensure that our corpus of help scenarios was realistic, we began by selecting the first 100 questions tagged as *popular* or *recently asked* in the *How Do I* category of *Google Calendar*'s help forum [38]. We chose *Google Calendar* because it is a popular application used by millions of people and offers not only basic functionality, but also a range of advanced functions that people have trouble finding and using. From our sample of 100 popular or recently asked questions, we eliminated questions that appeared to be duplicates and created a random sample of 50 questions that we could use in our evaluation. Although there were many more than 50 questions in Google's help forums, by analyzing the 10 "related discussions" that Google lists alongside each thread, we found that many of these discussions concerned the same issue and believe that 50 questions represented a substantial proportion of the common problems. This is reinforced by previous studies that have shown that there often are a large number of duplicate discussions on forums [28] and other forms of issue reports [18].

To convert the help discussions into scenarios, we identified the expected or desired behavior identified by the help requester and wrote a textual scenario to represent it. We also included a motivation for the task in the scenario and details about Google Calendar to help a user unfamiliar with the application understand the specified goal. Figure 5 shows an example scenario involving a calendar-sharing question.

In addition to scenario text, we also created a Google Calendar page representing an application state in which a user might encounter the problem, as in Figure 5. We created the HTML pages for each scenario by manually recreating the chosen Google Calendar state and scraping the application's corresponding HTML for that state. We then augmented each scenario page with LemonAid's question-asking functionality. Since LemonAid requires a list of UI literals corresponding to the application and we did not have access to Google Calendar's source code, we manually extracted a set of UI literals by scraping each visible text label (and ALT text of images) from all UI elements for each scenario page. Finally, because the focus of our study was retrieval performance on users' *first* selections and not on expert use of LemonAid, we disabled LemonAid's help retrieval and answer authoring functions for the study, so that after a participant selected a UI element and wrote a query, their task

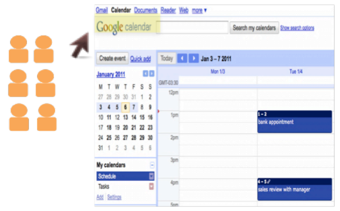
An mTurk *HIT* consisted of:

1) Reading a problem scenario related to Google Calendar.

You share your calendar with all team members at your company. Recently, one of the members left the company and now you would like to stop sharing the calendar with her, but cannot remember how to change the sharing details.

2) Answering 2 comprehension questions related to the scenario.

3*)* Making a selection relevant to the problem on a corresponding Google Calendar page instrumented with LemonAid.



4) Formulating a help-seeking question based on the scenario.

```
How to remove a person from a shared calendar?|
```

Figure 5: Illustration of mTurk use in developing a corpus

was complete. This reduced the possibility that participants would change the type of selections they made after completing multiple HITs based on the type of results returned by LemonAid's retrieval algorithm.

Of the 50 help problems, 8 were likely to be encountered in different contexts (for example on the main calendar view, or in a configuration dialog); for these, we created two scenarios, each with a different application state, resulting in a total of 58 scenarios overall.

Our mTurk HIT presented one of these 58 help-seeking scenarios (example in Figure 5), including the scenario text and the static HTML page with interactive LemonAid features. Users were asked to (1) read the scenario, (2) answer two multiple choice comprehension questions (described next), (3) enter the help mode, (4) select one of the highlighted words or elements on the screen that they felt were most relevant to the problem, and (5) provide a question in their own words that they would ask to get help in the given scenario.

The comprehension questions were included in order to gain some confidence that participants understood the scenario and were not selecting UI elements randomly (a common problem in many mTurk studies [8,17]). Each comprehension question had 5 items; the scenarios and questions were carefully edited by two of the authors for clarity. If users answered one of the questions incorrectly, they were given another explanation of the scenario to help them understand the scenario better.

Each mTurk HIT was launched with 55 assignments per HIT, with the intent of gathering 50 selections per scenario. We used 5 of the 58 HITs to pilot the mTurk protocol and our data collection strategy, resulting in a final data set of 53 unique HITs. We paid users $0.15 per HIT. (Each HIT took an average of 3.5 minutes to complete.) The study (including the pilot tests) ran for about 5 weeks. To prevent duplicate responses and other mischief, we asked mTurk users to

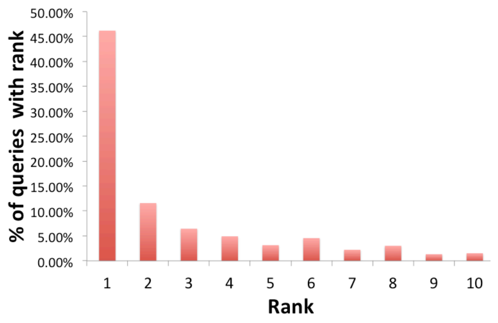


Figure 6. Distribution of ranks in the corpus.

provide a unique 6-digit passcode that was generated after they made an on-screen selection for a particular HIT. We also asked mTurk users had to write a brief explanation for why they made a particular selection.

After obtaining the data, we computed the time that an mTurk user spent on the task and compared it to the average completion time (3.5 minutes). If this time was below the 20% of the average *(i.e.,* less than 45 seconds), we automatically eliminated the response. For responses that fell between 45 seconds and 3.5 minutes, we manually checked the written explanation of why a particular selection was made. If the explanation was not intelligible, we excluded that response. Finally, we also checked the passcode that mTurk users provided against the passcodes generated by our system and eliminated responses that had incorrect passcodes. These three data points together allowed us to detect UI selections that appeared to be hastily selected with no apparent comprehension of the scenario. We were able to use between 47-52 selections for each HIT (about 10% of the data contained invalid selections as per the above criteria). Our final corpus included 2,748 help selections from 533 different mTurk accounts.

Results

As explained above, LemonAid uses a ranked retrieval approach where the retrieved results (in the form of *Questions)* are presented in an ordered list. Since our study solicited multiple selections corresponding to each scenario, multiple relevant *Questions* could potentially be retrieved for a given selection. To assess the performance of the retrieval, we focused on computing the rank of the 1st relevant *Question* for a given selection of all retrieved results. We defined ground truth in the retrieval by denoting, for each captured selection, which one of the 50 scenarios the selection corresponded to.

We computed ranks for all 2,748 selections in the corpus, retrieving relevant results from all other selections in the corpus using only the contextual data in the selections (excluding participants' question text). LemonAid retrieved 1 or more results for 90.3% of the selections. Figure 6 shows the proportion of queries resulting in median ranks of 1 through 10. The median rank of the results across the whole corpus was 2, thus the relevant result was likely to be in the top 2 results for at least half of the queries (about 57.8% in this case).

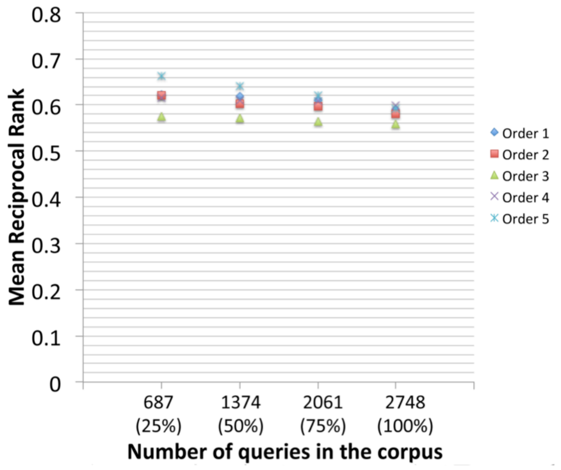


Figure 7. MRR for different corpora sizes and orderings.

To assess performance across the whole corpus more systematically, we computed the Mean Reciprocal Rank (MRR). MRR values are bounded between 0 and 1 and are sensitive to the rank position

$$\mathrm{MRR} = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{1}{rank_i}$$

(*e.g.*, from rank 1 to 2, MRR falls from 1.0 to 0.5). The reciprocal rank of a result is equivalent to the multiplicative inverse of the rank of the first relevant result. The MRR is computed as the average of the reciprocal ranks of results for a set of queries in corpus C where $1/rank_i$ is the inverse rank of the i^{th} query in C, and $|C|$ is the size of the corpus. The resulting MRR was 0.5844, meaning that the average rank of the result across the repository (taking into account all the best and worst-case ranks) was between 1 and 2.

To understand why LemonAid failed for 9.7% of the queries, we inspected the selections made by users (based on the `nodeText`, `nodeXPath`, and `nodeType`) and the corresponding task scenario. We found that the failed queries mainly represented idiosyncratic selections; in other words, a few of the users made selections that did not match any selection made by other users. When we further looked at the corresponding question text provided with the selections, we found that such users (despite meeting our initial filtering criteria) either misunderstood the scenario description, were confused about the selection task, or were simply guessing.

While the overall performance of the retrieval algorithm is important, its performance over time, as users ask more questions, is also important. To investigate the effect of corpus size, we randomly selected 5 subsets of queries of four different corpus sizes (25%, 50%, 75%, and 100 % of the 2,748 queries). Figure 7 displays the MRR for these 20 corpus subsets, showing that while MRR degrades as the number of selections increase, it degrades quite slowly. A live deployment of LemonAid would obviously introduce other factors that would affect these outcomes; for example, there might be many more help problems. However, these results show that users would also be more likely to find an existing question about a problem rather than ask a new one.

DISCUSSION

While our initial evaluation represented data from a simulated community of users on mTurk, the main finding is promising: LemonAid retrieved a relevant match in the top 2 results for over half of the queries based on the UI selection. Thus, in most cases, users would only have to make a UI selection that they think is relevant and they would see a relevant question (and answer, if available). This is a dramatic improvement over traditional text-based queries for help on the web, which require substantially more effort. The key phenomenon that facilitates the retrieval of high-ranking relevant results is that users' queries are restricted to a smaller and more focused set of UI selections instead of natural language text and that users tend to select similar labels for similar problems and different labels for different problems.

These results, and the larger vision underlying LemonAid's approach to crowdsourced contextual help, raises some issues around scope, scalability, robustness, and privacy.

Problem Scope. For a given application, users may have a range of feature-related or account-specific troubleshooting help needs. Since LemonAid is integrated within the UI of the application, its primary strength is likely to be in providing user interface related help. For other types of issues that reach beyond the user interface, such as a problem with a blocked account or an issue with a credit card transaction, LemonAid would be able to inform a user that it is necessary to contact support, but it will not be able to help the user address their problem directly. Help needs that require the intervention of support personnel are less a limitation of LemonAid and more a limitation of crowdsourced help approaches in general.

Scalability. As shown in Figure 7, we have some initial indication that the retrieval algorithm is relatively stable as a help corpus increases in size. However, another important question is how LemonAid's retrieval scales for applications that vary from a narrow to a wide range of features and corresponding UI literals. For instance, in our study we observed was different users consistently made the same selection in the UI for the same problem, but made different selections for different types of problems. Thus, for an application that has a large number of features (and more possibilities for selections), the spread of questions could be sparse. For the case of an application with only a few features, there will likely be similarly few possible selections. We predict that LemonAid's performance will still degrade slowly as there would possibly be fewer questions about applications that have more limited functionality.

Another case we observed in our evaluation was the same label being used as an anchor for many different problems. For example, the "settings" label of *Google Calendar* was a particularly common choice when users perceived no better label in some of the scenarios. The retrieval algorithm was not able to retrieve a relevant answer in the top few results based on the selection alone. In this situation, the user would need to provide keywords to pare down the results. Still, in the worst case, LemonAid only degrades to the performance of a full-text search, but within the limited scope of questions generated through the LemonAid interface, rather than everything on the web.

Robustness. One concern about the utility of LemonAid in practice might be that web applications are constantly changing; anchoring help to rapidly changing labels and layouts may not be robust to such change. The UI labels that LemonAid relies on, however, are likely to change less often than the average website content, since changing functionality labels often requires costly user re-education. Moreover, when functionality labels and locations do change, it would actually make sense for the help associated with those UI literals to be deprecated. With LemonAid, this would be automatic, since questions attached to labels that have been removed would no longer be matched. The only process required to keep help content current would be to refresh LemonAid's list of application-specific UI literals, which is a simple matter of re-extracting string literals from their source)

Privacy. By using text on web pages, much of which may be privacy-sensitive, LemonAid also raises some privacy concerns. However, since we are only extracting UI literals from source code, and users can only select labels that match these static labels, user-generated content is never captured as part of a help request. There is a possibility that there may be some overlap between a UI literal and user-generated text. Future versions of LemonAid could allow users to redact details from their selections before submission.

Bootstrapping. While we have shown LemonAid performs well on a large corpus of queries, the approach still requires someone to provide help in order for the system to be useful and the help must actually be helpful. These challenges are not unique to LemonAid, however; they are more general challenges with crowdsourced help, and evidence has shown that they are they are easily surmountable with the right types of incentives and community features [34]. In future work, we will explore these community aspects further.

Evaluation Limitations. Our evaluation has some limitations that that should be considered when interpreting our results. For example, our results might only hold for the type of users represented by mTurk workers [17]. Although we tried to filter out invalid selections in our mTurk data (based on our criteria discussed above), it could be that a few users genuinely misunderstood scenario descriptions or the purpose of the task and ended up selecting something not relevant to the scenario. Moreover, our evaluation did not explore the effect of LemonAid users *interactively* exploring LemonAid search results, which may also affect LemonAid's utility. We hope to explore these issues further in a live deployment where users would be driven by their actual help needs.

RELATED WORK

Although LemonAid's approach to retrieving help is novel, it builds upon a variety of prior work in help systems research.

Context-Sensitive Help. Context-sensitive help to date has largely been about attaching help to specific UI controls. Researchers have explored a variety of ways to invoke this help, including tooltips, special modes as in the "?" icon in some Windows dialog boxes, Balloon Help [10] in early Apple systems, pressing F1 over UI elements, and even choosing a command to see animated steps [32] or videos [12]. Other recent approaches have explored the use of screenshots and visual search in creating contextual help [35]. Despite the utility of these forms of context-sensitive help, one drawback is that designers must *anticipate* where users might seek help, so that they can author it at design-time and attach it to UI controls. Also, the help presented is static and often limited to explaining the functionality of a widget. LemonAid addresses this issue by letting *users* decide where help should be embedded, authoring that help at run-time.

Adaptive Help. Adaptive help attempts to overcome context-insensitivity by monitoring user behavior for opportunities to help [7,24]. These systems make an explicit effort to model users' tasks, often employing AI techniques to predict and classify user behavior, some even using speech recognition [15]. Perhaps the most well known is "clippy" in Microsoft Word (which was a simplified version of a more successful intelligent agent [16]). Although powerful, these systems are limited by their ability to model and infer users' intent, meaning that the static help that they provide can often be irrelevant. Moreover, these systems may interrupt at inappropriate times and are often perceived as being intrusive. In contrast, an ambient and unobtrusive approach is feature recommendation based on monitoring of application usage [21]. Still, in all of these cases, help is tied to functionality rather than user's intentions and application use.

Automatic Help. Another class of help tools manifest as automatic help tools. Rather than inferring users' intent, such tools enable users to explicitly state their problems to obtain customized help. For example, SmartAidè [26] allows users to choose particular application task, and AI planning algorithms generate step-by-step instructions based on the current application state. The Crystal system [23] allows users to ask "why?" questions about unexpected output by simply clicking on the output itself. While such help techniques are powerful in generating customized solutions to users' help requests, they can only answer a limited class of questions amenable to automatic analysis. They also often require significant adaptations to an applications' code to provide useful answers.

Crowdsourced Help. Crowdsourced help is the most recent approach to software help. The essential idea is that the user community can generate solutions to help requests more quickly than any tool or in-house support team [13]. Early research examples of this approach, such as AnswerGarden [1], focused on organizational support and exchange of expertise; similar ideas emerged in the open source community in technical support forums [28]. Some research has explored the role of contextual help in content authoring. For example, the CHIC framework [31] for the Eclipse IDE adds links from each Eclipse UI control to a wiki where users can author help. Similar approaches that try to link community discussions in the interface have appeared recently in the IP-QAT system [20] and in commercial contexts [37] as well. LemonAid goes further by letting users decide which aspect of the interface matters for particular problems and allows users to author and discover help there. Furthermore, LemonAid is not specifically tied to any application structure and can be applied to any site implemented with web standards.

FUTURE WORK AND CONCLUSION

In this paper, we have introduced LemonAid, a new framework for integrating crowdsourced contextual help in web applications. We have shown that LemonAid's approach to selection-based query and retrieval is effective, providing a relevant answer in the top 2 results for over half of the queries in a corpus developed by a simulated community. We also have initial evidence that as a LemonAid help corpus grows in size, its ability to retrieve relevant results degrades slowly.

In our future work, we will explore a number of enhancements. For example, we will incorporate community feedback features for improving the ranking of search results and indicating which questions require useful answers, enabling users to vote on relevant questions and solutions that are tied to specific selections. We will include multimedia options, such as screen shots and videos, for enhancing solution authoring and diagnosis of reported issues [6].

In addition to helping users find help content, LemonAid may also provide other benefits. For example, a user could browse the goals of other users who have used the site by simply clicking on different labels in the UI. Other users' goals expressed in the context of the application could lead to the serendipitous discovery of new application features, shortcuts, and customizations. Software teams could also use their product's LemonAid help repository as a dataset of user expectations and potential usability issues. This would be a significant improvement over the status quo, where feedback about an application is scattered in discussion forums and social networking sites all over the web, with no simple way to monitor them. With LemonAid, user selections and queries can be easily aggregated and placed in the exact context in which users experienced a problem. Ultimately, software teams can use this information to better understand users and provide a more seamless user experience.

ACKNOWLEDGEMENTS

We thank Tovi Grossman, Jeff Huang, and Michael Lee. This work was funded in part by fellowships awarded to the first author from Facebook Inc., and Canada's Social Sciences and Humanities Research Council (SSHRC).

REFERENCES

1. Ackerman, M.S. and Malone, T.W. Answer Garden: a tool for growing organizational memory. *Proc ACM SIGOIS* (1990), 31-39.
2. Adar, E., Dontcheva, M., Fogarty, J., and Weld, D.S. Zoetrope: interacting with the ephemeral web. *Proc ACM UIST* (2008), 239-248.
3. Belkin, N.J. Helping people find what they don't know. *Commun. ACM 43*, 8 (2000), 58-61.

4. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R.C. Automation and customization of rendered web pages. *Proc ACM UIST* (2005), 163-172.

5. Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S.R. Example-centric programming: integrating web search into the development environment. *Proc ACM CHI* (2010), 513-522.

6. Chilana, P.K., Grossman, T., and Fitzmaurice, G. Modern software product support processes and the usage of multimedia formats. *Proc ACM CHI* (2011), 3093-3102.

7. Delisle, S. and Moulin, B. User interfaces and help systems: from helplessness to intelligent assistance. *Artificial Intelligence Review 18*, 2 (2002), 117-157.

8. Downs, J.S., Holbrook, M.B., Sheng, S., and Cranor, L.F. Are your participants gaming the system?: screening mechanical turk workers. *Proc ACM CHI,* (2010), 2399-2402.

9. Ekstrand, M., Li, W., Grossman, T., Matejka, J., and Fitzmaurice, G. Searching for software learning resources using application context. *Proc ACM UIST* (2011), 195-204.

10. Farkas, D.K. The role of balloon help. *ACM SIGDOC 17*, 2 (1993), 3-19.

11. Furnas, G.W., Landauer, T.K., Gomez, L.M., and Dumais, S.T. The vocabulary problem in human-system communication. *Commun. ACM 30*, 11 (1987), 964-971.

12. Grossman, T. and Fitzmaurice, G. Toolclips: An investigation of contextual video assistance for functionality understanding. *Proc ACM CHI* (2010), 1515-1524.

13. Harper, F.M., Raban, D., Rafaeli, S., and Konstan, J.A. Predictors of answer quality in online Q&A sites. *Proc ACM CHI* (2008), 865-874.

14. Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S.R. What would other programmers do: suggesting solutions to error messages. *Proc ACM CHI* (2010), 1019-1028.

15. Hastie, H.W., Johnston, M., and Ehlen, P. Context-Sensitive Help for Multimodal Dialogue. *Proc IEEE ICMI* (2002), 93.

16. Horvitz, E. Principles of mixed-initiative user interfaces. *Proc ACM CHI* (1999), 159-166.

17. Kittur, A., Chi, H., and Suh, B. Crowdsourcing user studies with Mechanical Turk. *Proc ACM CHI* (2008), 453-456.

18. Ko, A.J. and Myers, B.A. Designing the whyline: a debugging interface for asking questions about program behavior. *Proc ACM CHI* (2004), 151-158.

19. Lakhani, K.R. and Von Hippel, E. How open source software works:"free" user-to-user assistance. *Research policy 32*, 6 (2003), 923-943.

20. Matejka, J., Grossman, T., and Fitzmaurice, G. IP-QAT: In-Product Questions, Answers & Tips. *Proc ACM UIST* (2011), 175-184.

21. Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. CommunityCommands: command recommendations for software applications. *Proc ACM UIST* (2009), 193-202.

22. Morris, M.R., Teevan, J., and Panovich, K. What do people ask their social networks, and why?: a survey study of status message q&a behavior. *Proc ACM CHI* (2010), 1739-1748.

23. Myers, B.A., Weitzman, D.A., Ko, A.J., and Chau, D.H. Answering why and why not questions in user interfaces. *Proc ACM CHI* (2006), 397-406.

24. Pangoli, S. and Paternó, F. Automatic generation of task-oriented help. *Proc ACM UIST* (1995), 181-187.

25. Paolacci, G., Chandler, J., and Ipeirotis, P.G. Running experiments on amazon mechanical turk. *Judgment and Decision Making 5*, 5 (2010), 411-419.

26. Ramachandran, A. and Young, R.M. Providing intelligent help across applications in dynamic user and environment contexts. *Proc ACM IUI* (2005), 269-271.

27. Salton, G., Wong, A., and Yang, C.S. A vector space model for automatic indexing. *Commun. ACM 18*, 11 (1975), 613-620.

28. Singh, V., Twidale, M.B., and Nichols, D.M. Users of Open Source Software - How Do They Get Help? *Proc HICSS* (2009), 1-10.

29. Singh, V., Twidale, M.B., and Rathi, D. Open Source Technical Support: A Look at Peer Help-Giving. *Proc HICSS* (2006), 118.3.

30. Snow, R., O'Connor, B., Jurafsky, D., and Ng, A.Y. Cheap and fast---but is it good?: evaluating non-expert annotations for natural language tasks. *Proc Empirical Methods in NLP* (2008), 254-263.

31. Stevens, G. and Wiedenhöfer, T. CHIC - a pluggable solution for community help in context. *Proc ACM NordiCHI* (2006), 212-221.

32. Sukaviriya, P. and Foley, J.D. Coupling a UI framework with automatic generation of context-sensitive animated help. *Proc ACM SIGGRAPH* (1990), 152-166.

33. Twidale, M.B. Over the shoulder learning: supporting brief informal learning. *J CSCW 14*, 6 (2005), 505-547.

34. Tynan-Wood, C. The (Better) Future of Tech Support. *InfoWorld*, 2010.

35. Yeh, T., Chang, T.H., Xie, B., Walsh, G., Watkins, I., Wongsuphasawat, K., Huang, M., Davis, L., and Bederson, B. Creating contextual help for GUIs using screenshots. *Proc UIST* (2011), 145-154.

36. Amazon Mechanical Turk. http://www.mturk.com/.

37. TurboTax Support. http://turbotax.intuit.com/support/.