Query Suggestion for Struggling Search by Struggling Flow Graph

Zebang Chen, Takehiro Yamamoto, Katsumi Tanaka
Department of Social Informatics, Graduate School of Informatics
Kyoto University, Japan
{chenzb, tyamamot, tanaka}@dl.kuis.kyoto-u.ac.jp

***Abstract*—We propose a method to generate effective query suggestions aiming to help struggling search, where users experience difficulty in locating information that is relevant to their information need in the search session. The core is identifying struggling component of an on-going struggling session and mining the effective representations of it. The struggling component is the semantic component of information need for which the user struggled to find an effective representation during the struggling session. The proposed method identifies the struggling component of given on-going struggling session and mines the sessions containing the identified struggling component from a query log to build a struggling flow graph. The struggling flow graph records users' reformulation behaviors for the terms of the struggling component, through struggling flow graph we can mine effective representations of the struggling component. The experimental results demonstrate that the proposed method outperforms the baseline methods when it can use two or more queries in a struggling session.**

I. Introduction

Search engines have been widely used to find information and solutions to problems. However, many users sometimes struggle to locate relevant information to their problem. This search process is referred as *struggling search* [1]. In struggling search, the user struggles to reformulate query many times for the information need. For example, Table I(a) shows a struggling session in which a user attempted to find information about *where did donut come from*. Unfortunately, although the user attempted to reformulate his/her query many times, he/she could not locate the required information and gave up. Hassan *et al.* reported that, among long search sessions, 36% are struggling sessions and many users fail to locate the required information [1]. Therefore, it is important for search engines to help users in struggling search effectively.

Query suggestion is an effective way to help a user reformulate query. Many query suggestion methods have been proposed and proven very useful. However, existing query suggestion methods have limitations when addressing struggling search. One problem is that most queries in struggling sessions are failed queries with few clicks, which means that existing methods that rely on query terms and clicked documents are likely to generate ineffective suggestions and may lead to another failed trial. Another problem is that the information needs of many struggling sessions are long-tail information need, which means that no user or few users have searched for the information previously, thus, there are no effective queries in the query log.

TABLE I: Examples of struggling sessions (terms in italics are the struggling phrase of the query).

session (a)	
Query	
the donut	<failed>
the donut *came from* where	<failed>
history of the donut	<failed>

session (b)	
Query	
history of the leprechaun	<failed>
leprechaun where did the name *come from*	<failed>
history of the leprechaun's name *origin*	<successful>

In this paper, we propose a query suggestion method aiming to help struggling search. The core idea of our method is identifying the *struggling component* of an on-going struggling session and mining effective representations of the struggling component to generate query suggestions. The *struggling component* is the semantic component of information need for which the user struggled to find an effective representation during the session (See Section III). For example, there are two semantic components in the information need of the session shown in Table I(a): *came from*, and *donut*. Among these components, the user struggled to find an effective representation for *came from*. The same struggling component may occur in search sessions for different information needs. As shown in Table I(b), another user attempted to find out *where did the name of leprechaun come from*. This user also struggled to find an effective representation for *came from*, similar to the user in Table I(a). However, this user finally found out an effective representation for the struggling component: *history origin*, which can also enable the user in Table I(a) to locate relevant information using query *"the donut history origin"*.

Given an on-going struggling session, we first identify its struggling component and retrieve sessions with the same struggling component from a query log. We then build a *struggling flow graph* by aggregating the struggling reformulation behaviors for terms of the struggling component. Further we can mine effective representations of the struggling component to generate effective query suggestions for the on-going struggling session.

The contributions of this paper can be summarized as follows.

- We propose a query suggestion method to help struggling search.
- We introduce the concept of struggling component and a method to identify the struggling component of a

struggling session, which enables us to understand which part that the user struggles at.
- We introduce the *struggling flow graph*, which is a graph that records multiple users' struggling reformulation behaviors for the terms of the struggling component.

The remainder of this paper is organized as follows. In Section II, we introduce related work including query suggestion, struggling search, and long-tail search. In Section III, we introduce and define several key concepts used in this paper. In Sections IV and V, we describe how we identify the struggling component and generate effective query suggestions, respectively. In Section VI, we explain our experiment and analyze the results. The paper is concluded in Section VII.

II. Related Work

Query Suggestion. One main approach to query suggestion is mining click-through data. Beeferman and Berger [2] built a bipartite graph based on click-through data and clustered queries with the same clicked documents because these queries are likely to have the same search intent. Baeza-Yates *et al.* [3] clustered queries based on their term-weight vector calculated from queries as well as their clicked documents. The queries in the same cluster are ranked and considered possible query suggestions. Mei *et al.* [4] ranked queries using the hitting time on the bipartite graph to ensure semantic consistency between the original query and query suggestions. Cao *et al.* [5] summarized queries and mapped sessions to concepts based on click-through data to build a concept sequence suffix tree, which is used to obtain the context of the current search in order to generate context-aware query suggestions. Another main approach to query suggestion is mining query reformulations of search sessions. Boldi *et al.* [6] used the query flow graph [7] to generate query suggestions. Each node in a query flow graph represents a query and each edge between two nodes means that they are consecutive in at least one session. A short random walk is applied to the graph to generate query suggestions. Jones *et al.* [8] proposed a model to generate a set of query substitutions by replacing the whole query or parts of the query with related phrases.

Struggling Search. Struggling search was first formally introduced by Hassan *et al.* [1]. They analyzed the characteristics of struggling sessions and proposed a model to distinguish struggling sessions and exploratory sessions [9]. Task difficulty is the main factor that leads to struggling search. Carmel *et al.* [10] investigated what makes a query difficult, by capturing and analyzing the relationships among the main components of a topic: the textual expression (the query or queries), the set of documents relevant to the topic, and the entire collection of documents. Aula *et al.* [11] studied behavioral signals when a user has difficulty with a search task. They found that, when finding relevant information is difficult, users attempt to formulate more diverse queries, use advanced operators more frequently, and spend more time on viewing search results pages. Liu *et al.* [12] explored the effect of task difficulty on search behavior for users with different levels of domain knowledge. Although many studies have been done on understanding task difficulty and how users behave when experiencing difficulty, few studies have attempted to find ways to assist struggling search. Liu *et al.* [13] used a learning-to-rank approach to rank query suggestions generated by previously proposed method [5] for difficult search, which is limited by ineffective click-through information when facing struggling search because most queries in struggling search are failed and without effective click. Odijk *et al.* [14] studied the characteristics of search tasks where users struggled and proposed method to predict user's query reformulation types.

Long-tail Search. Yao *et al.* [15] conducted an empirical study of user behavior with rare queries using a large-scale query log and proved significant differences among many features, including query length and search results. To generate query suggestions for long-tail queries, Bonchi *et al.* [16] [17] built a center-piece sub-graph containing term nodes and query nodes with two types of connections: term-query connections and query-query connections. A random walker starts with the terms in a given query to generate query suggestions even if the query has not occurred previously. To extend the reach of query suggestion for long-tail queries, Szpektor *et al.* [18] introduced the concept of *query template* and proposed an enhanced query flow graph [7] with query templates, through which query suggestions are available even for the long-tail queries that do not have succeeding query. Although both the method proposed by Bonchi *et al.* [16] [17] and the method proposed by Szpektor *et al.* [18] are effective on generating query suggestions for long-tail queries, they do not focus on the effectiveness of query suggestions to help locate relevant information, which is the key to helping struggling search.

III. Preliminaries

In this section, we first explain the concepts of query log, session, and struggling session used in this paper. We then introduce the key concepts for modeling struggling search in order to clarify our approach to generating query suggestions.

Query log: A *query log* is a log that records user search queries and document clicks with their timestamps. A typical format of a record in a query log is *<user_id, query, clicked document, timestamp>*. In this paper, we use the AOL query log [21], which contains the search behaviors of approximately 650,000 users over 3 months, as our primary dataset. Note that the proposed method is not specific to the AOL query log, but is applicable to other query logs.

Session: A *session* represents a topically coherent session during which the user's information need does not change. One well-known way to extract sessions from a query log is to use a predefined time threshold, *e.g.*, 30 min of inactivity [19], but a session extracted in this way may contain search queries and document clicks for different information needs [20]. To ensure that sessions are topically coherent, we used the settings in [1] with a small adjustment: two consecutive queries are topically coherent if they are no longer than 10 min apart and they must share at least one non-stopword term. In the rest parts of this paper, we simply use *session* to represent a *topically-coherent session*.

TABLE II: Example components.

Component	Representations
come from	come from, originate from, history, invent, *etc*.
troubleshoot	not work, stop work, fix, repair, stuck, problem, troubleshoot, *etc*.
gas mileage	gas mileage, mpg, fuel mileage, miles per gallon, kilometer per liter, *etc*.

Struggling session: A *struggling session* is a session in which the user experiences difficulty in locating information that is relevant to their information need. Typical user search behaviors in a struggling session are reformulating queries many times and spending significant time on the search process [1]. Examples are shown in Table I.

Here, we introduce the key concepts for modeling struggling search.

Information need: An *information need* is the need to locate information in order to satisfy a user's requirement. Example information needs are as follows: *where did donut come from*, *where did the name of leprechaun come from* and *what to do when your car fan stops working*, *etc.*

Component: A *component* is a semantic unit that corresponds to things in the real world, such as concepts, objects, actions, and relations. In this work, we model that an information need comprises a set of components. For example, the information need *where did donut come from* comprises two components, *donut*, *came from*.

Each component has a set of representations, which are used by users to describe the component in their queries. For example, to describe the component *came from*, users may use *came from, history, origin, etc*. More examples are shown in Table II.

Struggling component: In a struggling session, we define the *struggling component* as the component of information need for which user has difficulty in finding an effective representation. For example, for the session shown in Table I(a), the information need of the session is *where did donut come from*, which has two components, *donut*, *came from*. During the session, the user frequently reformulated the representations of the *came from* component, which may mean that the user had difficulty in finding an effective representation of this component. Thus, *came from* is the struggling component in this session.

Struggling phrase: Given a query in a struggling session, the *struggling phrase* of the query is defined as the set of terms in the query that represent the struggling component of the session. Note that the struggling phrase of a query may be empty. Table I shows examples of struggling phrases in struggling sessions. For example, the struggling phrases of the queries in Table I(a) are ϕ, {came from}, {history}, respectively.

IV. Identifying Struggling Phrase

In this section, we explain how we identify the struggling phrases of a struggling session.

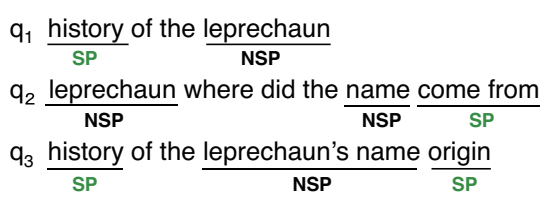

Fig. 1: Example of struggling phrase classifier.

A. Problem statement

Given a struggling session with a sequence of queries, our objective is to identify the struggling phrase of each query. To this end, we take a machine learning approach and build a struggling phrase classifier. Specifically, for each non-stopword term in a query of a struggling session, we classify it into one of two classes, *struggling phrase* (SP) and *non-struggling phrase* (NSP), as shown in Figure 1.

B. Features

We have designed several features for the struggling phrase classifier, as shown in Table III.

- **Term-level features:** These features describe the characteristics of a term itself. For example, a term with many synonyms has a higher probability to be struggling phrase because users may struggle to find an effective representation from these synonyms.
- **Query-level features:** These features describe a term's characteristics in a query. A term that did not occur in preceding or succeeding queries but had similar terms in the preceding and succeeding queries has a higher probability to be struggling phrase, as the term may be replacement of a term in preceding query, or was replaced by another term in succeeding query.
- **Session-level features:** These features describe a term's characteristics in a session. A term that the user gave up quickly has a higher probability to be struggling phrase, and a term that the user kept in every query of the session has a higher probability to be a non-struggling phrase.
- **Global-level features:** These features describe term's characteristics in the whole query log. If the struggling sessions ratio among all sessions the term occurred is high, then the term has a higher probability to make users struggle. If a term's occurrence ratio is low in many sessions it occurred, it may mean that many users replaced that term with other terms in their sessions.

C. Experimental setup

Data. We manually extracted 128 struggling sessions from the AOL query log, which contained 622 queries. For each query, we manually labeled each non-stopword term as struggling phrase or non-struggling phrase by checking the term, the query, the search results, and the context in the session, assuming we were the searcher of the search session. We trained the classifier with these labeled data to identify struggling phrase and non-struggling phrase of each query in a struggling session.

Baseline. For the baseline, we used the frequency-based rule to

TABLE III: Struggling phrase classifier features.

Name	Description
Term level features	
isEntity	whether the term refers to an entity
POS	part-of-speech of the term
synonymNum	number of synonyms in WordNet [22]
Query level features	
inQueryPos	the position order in query
maxNeighborSim	the maximum similarity between terms in preceding and succeeding queries
Session level features	
sessionOccurrenceRatio	ratio of queries in which the term occurred in session
sessionFirstOccurrence	order number of the query in which the term first occurred in session
Global level features	
strugglingSessionRatio	ratio of struggling sessions among all sessions the term occurred
globalOccurrenceRatio	average of occurrence ratio among all sessions the term occurred

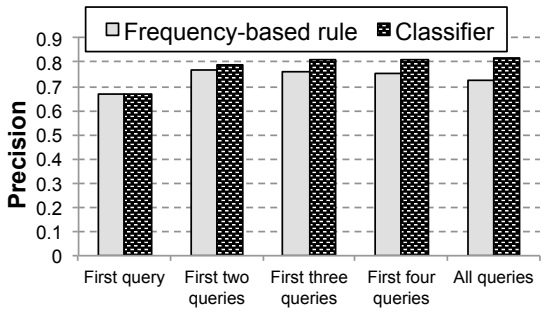

Fig. 2: Precision of struggling phrase classifier.

identify the struggling phrase of a query. This method classifies a term as a non-struggling phrase only when all queries in the session contain the term. Otherwise, the term is classified as a struggling phrase.

Setting. We used the support vector machine with a linear kernel to train the classifier. 10-fold cross-validation was used for this experiment. Note that the number of queries in a session influences the performance of both the classifier and the frequency-based rule. To investigate the effect of the number of available queries of a session on the classification performance, we experimented with several cases by changing the number of queries in a session (e.g., the first query, the first two queries, the first three queries ..., all queries).

D. Experimental results

Figure 2 shows the precision of classification by the classifier and the frequency-based rule. The x-axis is the number of queries of sessions used by each method.

As can be seen, the classifier outperformed the frequency-based rule in terms of precision. The precision of the classifier increased with an increased number of queries. The precision reached 0.82 when the classifier used all the queries in sessions. In contrast, the precision of the frequency-based rule increased when available queries increased from one query to two queries, but decreased with further increased number

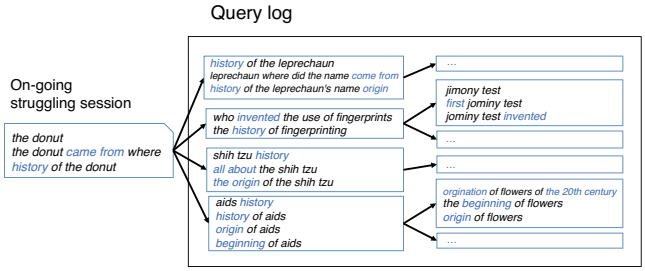

Fig. 3: Mining sessions with the same struggling component.

of queries. One reason for this is that users tended to slightly modify the terms of non-struggling phrases later in the session after several times of failure.

V. Generating Effective Query Suggestion

In this section, we first explain how we mine sessions that have the same struggling component as the on-going struggling session. We then introduce the struggling flow graph, how to build the graph, and mine effective representations of the struggling component to generate effective query suggestions.

A. Mining sessions with the same struggling component

The same struggling component may occur in sessions with different information needs. We have hypothesized that sessions with the same struggling component have similar struggling phrases, and share the same effective representations for the struggling component. Two examples have been shown in Table I(a) and Table I(b), in which the first user searched for *where did donut come from* and the second user searched for *where did the name of leprechaun come from*, both struggled to find an effective representation of the struggling component *came from*.

Given an on-going struggling session, we want to find sessions with same struggling component as the on-going session. Figure 3 illustrates how we mine sessions with same struggling component as the on-going session from a query log. Specifically, given an on-going struggling session, we first identify its struggling phrases and retrieve sessions that contain at least two queries and at least one identified struggling phrase in their queries from a query log. Then, we again extract the struggling phrases of each retrieved session to obtain more sessions with same struggling phrases in a breadth-first-search manner until no session for retrieving or with more than 1,000 retrieved sessions.

B. Struggling flow graph

Having obtained sessions with possibly the same struggling component as the on-going session, we then build a struggling flow graph with these sessions, which is a graph extending the *query flow graph* [6]. The purpose of the struggling flow graph is to aggregate multiple users' reformulation behaviors for the terms of a struggling component and mine the effective representations for the struggling component.

Here, s_0 denote an on-going struggling session for which we want to provide query suggestions, $S = \{s_1, \ldots, s_n\}$

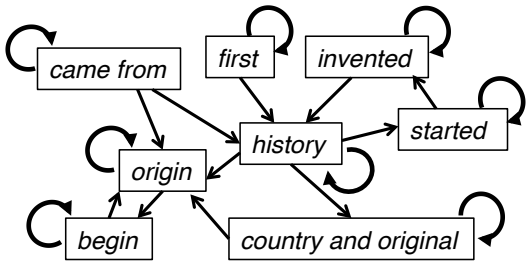

Fig. 4: Example of struggling flow graph.

denote the set of sessions mined by the method described in Section V-A , and $V = \{v_1, \ldots, v_m\}$ denote the unique set of struggling phrases obtained from S by using the classifier described in Section IV.

Formally, the struggling flow graph for sessions S is represented as weighted directed graph $G_S = (V, E, \omega)$, where:

- V is the set of nodes in the graph. Each node corresponds to a struggling phrase.
- $E \subseteq V \times V$ is the set of directed edges. If there is at least one session where two struggling phrases v_i and v_j appear consecutively in S, there is a directed edge from v_i to v_j. In addition, for every $v_i \in V$ there is a self-transitive edge from v_i to v_i.
- $\omega : E \rightarrow (0..1]$ is a weighting function that assigns a weight $\omega(v_i, v_j)$ to each pair of struggling phrases $(v_i, v_j) \in E$.

C. Finding effective representations by struggling flow graph

Here, we describe how we find the effective representations of the struggling component using the struggling flow graph. As explained previously, the struggling flow graph is an extension of the query flow graph. In the query flow graph, each node represents a query q and the weight of an edge between two queries q_i and q_j is determined by the probability that the user who issued query q_i and reformulated it as query q_j in sessions. To generate query suggestions for a given query q, a short random walk starting from query q is applied to the query flow graph. The query suggestions are ranked based on their value of stationary distribution.

We follow the idea of the query flow graph, except for the following differences. Differing from the query flow graph, which aggregates query reformulation of all sessions in the query log, a struggling flow graph aggregates only the reformulation of *terms* for a struggling component from a set of sessions with same struggling component. This allows us to track the term-level reformulation for single struggling component, which may address the problem of long-tail queries and keep query suggestions consistent. In addition, the high failure rate of struggling search increases the probability of generating failed query suggestions by the random walk in the graph. To address this problem, we incorporate information about whether a phrase appears to be successful or a failure into the graph. Specifically, we set a self-transition probability for each node based on our assumption on effective struggling phrases, a possibly effective struggling phrase based on our assumption will be set with higher self-transition probability and is likely to obtain a higher random walk value in the stationary distribution.

D. Preparing transition probabilities

In this subsection, we describe how we compute the transition probabilities among the nodes in the struggling flow graph. The assumption behind mining effective struggling phrases is that if a struggling phrase is effective, more users who input a query containing it will stop searching as it is effective in helping the users find relevant pages with the information they require. Similarly, if a struggling phrase is ineffective, more users who input a query containing it will continue searching as it will fail to help the users find the relevant pages. Based on this assumption, we first set the self-transition probability of each node (struggling phrase) as the ratio of sessions that end with the struggling phrase with effective clicks among all the sessions it occurred. Here, we regard that a query had an *effective click* if a user clicked at least one document from its search result page and spent more than 30 seconds at the document. We define the self-transition probability of struggling phrase v_i as follows:

$$\omega_{\text{original}}(v_i, v_i) = \frac{e(v_i)}{\sum_{v_k \in V} N(v_k, v_i)} \ , \tag{1}$$

where $e(v_i)$ represents the number of sessions that end with a query containing struggling phrase v_i with effective clicks in sessions S and $N(v_k, v_i)$ represents the number of sessions where v_k and v_i occur consecutively in sessions set S.

The problem of the above equation is that the value of $\omega_{\text{original}}(v_i, v_i)$ is not trustworthy if the struggling phrase v_i occurred only a few times. For example, suppose that a struggling phrase was input only once in a query and the user who input the query stopped searching after inputting it. Although 100% of users who input a query containing it stopped searching, it is not guaranteed to be effective as that the user might stop just because he/she gave up. To handle this problem, we define the *trustworthiness degree* of a struggling phrase to evaluate how trustworthy the information of the struggling phrase is. The trustworthiness degree $\text{td}(v_i)$ of a struggling phrase v_i is defined as:

$$\text{td}(v_i) = \min\left(1.0, \frac{e(v_i)}{\theta}\right) \ ,$$

where θ is a threshold, which we empirically set to 16 in this work. As a struggling phrase v_i occurs more times, $\text{td}(v_i)$ approaches 1.0.

Based on the trustworthiness degree, we modify the self-transition probability of a struggling phrase. If a struggling phrase is trustworthy enough, we use the same self-transition probability as described in Equation (1). If the trustworthiness degree of a struggling phrase is low, then its self-transition probability will be cut based on its trustworthiness degree. The modified self-transition probability $\omega(v_i, v_i)$ of struggling phrase v_i is defined as:

$$\omega(v_i, v_i) = \text{td}(v_i)\omega_{\text{original}}(v_i, v_i) \ .$$

For the probability cut from the original self-transition probability, we averagely distribute it to all other struggling phrases in the struggling flow graph as a random jump probability from v_i, which is denoted as $\mathrm{rj}(v_i)$:

$$\mathrm{rj}(v_i) = \frac{\omega_{\mathrm{original}}(v_i, v_i) - \omega(v_i, v_i)}{|V| - 1} \ .$$

Finally, we set the transition probability between two different struggling phrases v_i and v_j $(i \neq j)$ as follows:

$$\omega(v_i, v_j) = (1 - \omega_{\mathrm{original}}(v_i, v_i)) \frac{N(v_i, v_j)}{\sum_{v_j \in V} N(v_i, v_j)} + \mathrm{rj}(v_i) \ .$$

Note that the both weighting functions $w(v_i, v_i)$ and $w(v_i, v_j)$ are already normalized such that their values represent the transition probability among nodes.

E. Generating query suggestions

In this subsection, we describe the overall method to generate query suggestions for a given on-going session. Given an on-going session $s_0 = \{q_1, \ldots, q_l\}$, where l represents the number of queries in the session, we first identify the struggling phrases of the session using the method described in Section IV. We then mine the set of sessions S using the method described in Section V-A and build the struggling flow graph G_S. Then, we apply a random walk to the graph and extract the top k struggling phrases according to their random walk values in the stationary distribution. Finally, we generate the query suggestions by replacing the struggling phrase of the last query in the on-going sessions with each of the top k struggling phrases.

VI. Experiment

In this section, we introduce an experiment conducted to evaluate the effectiveness of the query suggestions generated by the proposed method. For the baseline, we used the query flow graph proposed by Boldi *et al.* [7] and the similarity-based method. The query flow graph is a directed graph that aggregates a query log and has been proven very useful for query suggestions and user behavior analysis. For the similarity-based method, given an on-going struggling session, we first compute the similarity between the last query of the session and queries in the query log. We then rank the queries based on their similarity and suggests the top k similar queries as query suggestions. We followed the method used in [1] to compute the similarity between two queries q_i and q_j as: $\frac{|q_i \cap q_j|}{|q_i| + |q_j| - |q_i \cap q_j|}$, where $|q|$ represents the number of terms in the query. To calculate the number of matches in query q_i and q_j, two terms were considered as matched if any of following criteria are met: (1) exact match, (2) approximate match, (3) lemma match, (4) semantic match. For the proposed method of the struggling flow graph, we also conducted experiments with different struggling phrase identification methods in order to investigate their influence on query suggestion.

Thus, we compared the following four methods in the evaluation: (1) query suggestions using the query flow graph, which is denoted as *QFG*, (2) query suggestions based on the similarity-based method, which is denoted as *SIM*, (3) query suggestions using the struggling flow graph with the frequency-based rule to identify struggling phrases, which is denoted as *SFG-frequency*, and (4) query suggestions using the struggling flow graph with the classifier to identify struggling phrases, which is denoted as *SFG-classifier*.

TABLE IV: Examples sessions used in experiments.

Struggling component	**Example session**
gas mileage	q_1: lincoln mkz 2007 q_2: lincoln 2007 mkz mileage q_3: 2007 mkz lincoln epa
origin	q_1: where does the name bonnaroo come from q_2: bonnaroo hisotry q_3: bonnaroo history
troubleshoot	q_1: my serial mouse does not work q_2: parrell mouse port not recognized q_3: how to fix mouse in registry

A. Evaluated sessions

To evaluate the effectiveness of the proposed method with diverse struggling sessions, we first manually prepared ten types of struggling components by scanning the struggling sessions sampled from the AOL query log. For each struggling component, we manually extracted as many struggling sessions with the struggling component as possible from the AOL query log. Finally, for each struggling component, we randomly sampled 10 struggling sessions from the extracted sessions and used them in the experiment (successful queries were removed for the successful struggling sessions). In total, 100 sessions were used for evaluation and removed from the query log. Table IV shows examples of struggling components and their corresponding session.

B. Ground truth

We defined a query suggestion that enables a user to locate relevant pages as an *effective* suggestion. To create ground truth query data, for each session described in the previous subsection, we first simulated the session user and attempted to understand the user's search intent as best as possible. For each of the top 10 query suggestions generated by a method, we issued the suggestion to a commercial search engine and examined whether there were relevant pages for the user's information need in the first page of the search results. If there was at least one relevant document in the search results page, the suggestion was considered *effective*; otherwise, the suggestion was considered *ineffective*.

C. Evaluation metrics

The following metrics were used to measure the performance of query suggestion methods.

- **Effective support rate@10**: The ratio of sessions in which one or more effective query suggestions were offered in the top 10 query suggestions.
- **MRR@10**: Mean reciprocal rank (MRR) of the top 10 query suggestions.

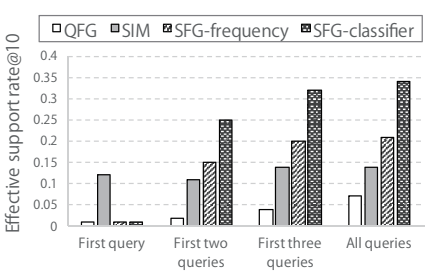

Fig. 5: Effective support rate@10.

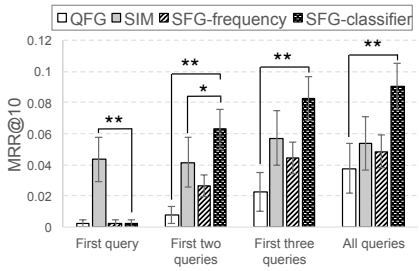

Fig. 6: MRR@10.

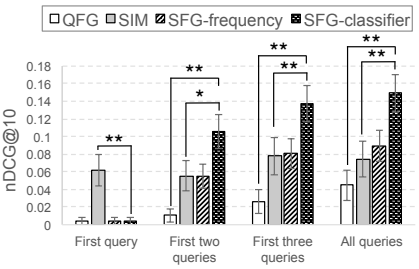

Fig. 7: nDCG@10.

- **nDCG@10**: Normalized discounted cumulative gain (nDCG) of the top 10 query suggestions. Here, we used binary relevance for the evaluation. An effective query suggestion had relevance of 1, while an ineffective query suggestion had relevance of 0.

As for MRR@10 and nDCG@10, we regarded these metrics as 0 when a method failed to generate any query suggestion.

D. Setting

We used all the labeled struggling sessions prepared in Section IV-C to train the struggling phrase classifier, which was used to detect struggling phrases of a session. In addition, as with the experiment explained in Section IV-C, we experimented with several cases by changing the number of available queries in a session to investigate its influence on the query suggestion performance. Note that there is no overlap between the sessions prepared in Section IV-C and those prepared in Section VI-A.

E. Results

Figure 5 shows the results of effective support rate@10. It can be observed that, when we can only use the first query in the session, SIM outperformed the other methods in terms of generating effective query suggestions (i.e. effective support rate@10). One possible reason why SIM outperformed SFG-classifier and SFG-frequency is that the proposed method relies on the struggling phrases in the session. The struggling phrases identified only from a single query might be too ambiguous to mine the appropriate representation of the struggling component.

When we can use two or more available queries in the session, both SFG-frequency and SFG-classifier outperformed QFG and SIM in terms of generating effective query suggestions. One main reason why SFG-frequency and SFG-classifier outperformed QFG is many queries in struggling search are long-tail queries, which may have not occurred in the query log. Since QFG relies on the occurrence of the query and query transition, it often failed when generating query suggestions for struggling session. SIM does not rely on the occurrence of the query so it is able to generate query suggestions for most struggling sessions. However, most queries during struggling sessions were failed queries. A query that is similar to a failed query is likely be another failed query, so many query suggestions generated by SIM were ineffective. We also can see that SFG-classifier outperformed SFG-frequency, surprisingly the query suggestion performance difference is much bigger than that of the precision on identifying struggling phrases between the classifier and the frequency-based rule shown in Figure 2. Two possible reasons of this are: (1) we recursively applied the struggling phrase identification method to the retrieved sessions, which made the influence of the precision difference on identifying struggling phrases become bigger, and (2) the wrong identification of struggling phrase sometimes introduced many noisy sessions and greatly worsened the query suggestion performance.

Figures 6 and 7 summarize the results of MRR@10 and nDCG@10, respectively. The error bars in Figures 6 and 7 represent the standard error of the mean (SEM). Significant differences between SFG-classifier and the baselines found by the two-sided Randomized Tukey's HSD test [23] at significant level $\alpha = 0.1$ and $\alpha = 0.05$ are marked with "*" and "**", respectively. From Figures 6 and 7, we can see that SFG-classifier outperformed QFG and SIM when we can use two or more queries in the session. When using all the queries in the session, QFG achieved MRR@10 of 0.038 and nDCG@10 of 0.045, SIM achieved MRR@10 of 0.054 and nDCG@10 of 0.075, SFG-frequency achieved MRR@10 of 0.049 and nDCG@10 of 0.090, and SFG-classifier achieved the best performance with MRR@10 of 0.090 and nDCG@10 of 0.149. While SFG-classifier achieved the best performance, we also found its performance was still not high. This indicates that the query suggestion for struggling search is still difficult and we need to further tackle the problem.

An example of query suggestion result is shown in Table V, which was generated for the first struggling session in Table IV when the number of available queries was two, where the user struggled to find *the gas mileage of lincoln 2007 mkz*. The query suggestions generated by QFG are shown in Table V(a), and those generated by the SFG-classifier are shown in Table V(b). As can be seen, no query suggestion was generated by the QFG because the queries of the first session in Table IV never occurred in other sessions in the query log. While the SFG-classifier method first identified struggling component of the struggling session, and then mined effective representations of it using struggling flow graph to generate query suggestions, which successfully generated several effective query suggestions for the user's information need.

TABLE V: Query suggestions for the first struggling session in Table IV.

Suggestions
Not available.

(a) Suggestion by QFG.

Suggestions
1st: lincoln 2007 mkz gas mileage
2nd: lincoln 2007 mkz distance
3rd: lincoln 2007 mkz drive
4th: lincoln 2007 mkz calculator
5th: lincoln 2007 mkz best mileage
6th: lincoln 2007 mkz mile
7th: lincoln 2007 mkz fuel mileage

(b) Suggestion by SFG-classifier.

F. Limitations

As the experimental results show, SFG-classifier achieved the best performance when it could use two or more queries in the session. However, the performance of SFG-classifier was still not high due to the several limitations. The first limitation is that the current SFG-classifier method can only support the struggling sessions in which users mainly struggle at a single component. If the user struggles at multiple components during the session, SFG-classifier may fail to distinguish these multiple struggling components and treat them as a single component, which will lead to generate ineffective query suggestions or even no query suggestion. For example, SFG-classifier failed to generate any query suggestions for the third struggling session shown in Table IV, because the user struggled at two components (i.e. *serial mouse* and *does not work*). SFG-classifier wrongly treated these multiple struggling components as a single struggling component and failed to match any past sessions with the same struggling component in a query log. To solve the problem, we need to improve our model so that it can identify multiple struggling components and find the effective representations for each component. The second limitation is that SFG-classifier introduces many noisy sessions that do not share the same struggling component of the on-going struggling session during mining sessions with the same struggling component, and we are considering an effective method to remove these noisy sessions. The third limitation is that SFG-classifier relies on the existence of past successful sessions which share same struggling component with on-going struggling session in the query log. There are struggling sessions that have unique struggling component and no user struggled at that component before, so the SFG-classifier will fail to generate effective query suggestions for these struggling sessions.

VII. Conclusion

In this paper, we proposed a query suggestion method for struggling search based on identifying the struggling phrases of a struggling session and mine effective representations for them using struggling flow graph. We conducted an experiment to investigate the effectiveness of the proposed method compared to baselines. The experimental results show that the proposed method outperformed the baselines when the method can use two or more queries of struggling session.

From the experiment, we found that the precision on identifying struggling phrases greatly affected the query suggestion performance. In the future, we plan to improve the model for identifying struggling phrases and improve the prediction of the user's information need by analyzing term-level reformulation behaviors.

Acknowledgment

This work was supported in part by JSPS Grants-in-Aid for Scientific Research (Nos. 15H01718 and 16K16156).

References

[1] Ahmed Hassan, Ryen W White, Susan T Dumais, and Yi-Min Wang. Struggling or exploring?: disambiguating long search sessions. In *WSDM'14*, pages 53–62, 2014.

[2] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *KDD'00*, pages 407–416, 2000.

[3] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 588–596, 2005.

[4] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *CIKM'08*, pages 469–478, 2008.

[5] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In *KDD'08*, pages 875–883, 2008.

[6] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Workshop on Web Search Click Data*, pages 56–63, 2009.

[7] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *CIKM'08*, pages 609–618, 2008.

[8] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *WWW'06*, pages 387–396, 2006.

[9] Gary Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[10] David Carmel, Elad Yom-Tov, Adam Darlow, and Dan Pelleg. What makes a query difficult? In *SIGIR'06*, pages 390–397, 2006.

[11] Anne Aula, Rehan M Khan, and Zhiwei Guan. How does search behavior change as search becomes more difficult? In *CHI'10*, pages 35–44, 2010.

[12] Chang Liu, Jingjing Liu, Michael Cole, Nicholas J Belkin, and Xiangmin Zhang. Task difficulty and domain knowledge effects on information search behaviors. In *ASIST'12*, 49(1):1–10, 2012.

[13] Yang Liu, Ruihua Song, Yu Chen, Jian-Yun Nie, and Ji-Rong Wen. Adaptive query suggestion for difficult queries. In *SIGIR'12*, pages 15–24, 2012.

[14] Daan Odijk, Ryen W. White, Ahmed Hassan Awadallah, and Susan T. Dumais. Struggling and success in web search. In *CIKM'15*, pages 1551–1560, 2015.

[15] Ting Yao, Min Zhang, Yiqun Liu, Shaoping Ma, and Liyun Ru. Empirical study on rare query characteristics. In *WI'11*, pages 7–14, 2011.

[16] Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini. Recommendations for the long tail by term-query graph. In *WWW'11*, pages 15–16, 2011.

[17] Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini. Efficient query recommendations in the long tail via center-piece subgraphs. In *SIGIR'12*, pages 345–354, 2012.

[18] Idan Szpektor, Aristides Gionis, and Yoelle Maarek. Improving recommendation for long-tail queries via templates. In *WWW'11*, pages 47–56, 2011.

[19] Ryen W White and Susan T Dumais. Characterizing and predicting search engine switching behavior. In *CIKM'09*, pages 87–96, 2009.

[20] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM'08*, pages 699–708, 2008.

[21] Greg Pass, Abdur Chowdhury, Cayley Torgeson. A Picture of Search. In *InfoScale'06*, page 1, 2006.

[22] Christiane Fellbaum. WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press, 1998.

[23] Tetsuya Sakai. Metrics, statistics, tests. In *PROMISE Winter School 2013: Bridging between Information Retrieval and Databases*, 2014.