Structured Query Suggestion for Specialization and Parallel Movement: Effect on Search Behaviors

Makoto P. Kato†‡[*] Tetsuya Sakai† Katsumi Tanaka‡

†Microsoft Research Asia, China
tetsuyasakai@acm.org

‡Kyoto University, Japan
{kato,tanaka}@dl.kuis.kyoto-u.ac.jp

ABSTRACT

Query suggestion, which enables the user to revise a query with a single click, has become one of the most fundamental features of Web search engines. However, it is often difficult for the user to choose from a list of query suggestions, and to understand the relation between an input query and suggested ones. In this paper, we propose a new method to present query suggestions to the user, which has been designed to help two popular query reformulation actions, namely, specialization (e.g. from "nikon" to "nikon camera") and parallel movement (e.g. from "nikon camera" to "canon camera"). Using a query log collected from a popular commercial Web search engine, our prototype called *SParQS* classifies query suggestions into automatically generated categories and generates a label for each category. Moreover, SParQS presents some new entities as alternatives to the original query (e.g. "canon" in response to the query "nikon"), together with their query suggestions classified in the same way as the original query's suggestions. We conducted a task-based user study to compare SParQS with a traditional "flat list" query suggestion interface. Our results show that the SParQS interface enables subjects to search more successfully than the flat list case, even though query suggestions presented were exactly the same in the two interfaces. In addition, the subjects found the query suggestions more helpful when they were presented in the SParQS interface rather than in a flat list.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation

Keywords

Query suggestion, search user interface, query log mining, Web search

1. INTRODUCTION

Query suggestion has become one of the most fundamental features of commercial Web search engines. Given a list of query suggestions, the user can simply click on one of them to initiate a new search. Providing effective query suggestions to the user is very important for helping the user express his information need precisely so that he can access the required information. Hence, many query suggestion algorithms based on session and clickthrough data have been proposed [2, 4, 5, 7, 12, 18, 19, 25]. On another front, how to present query suggestions can greatly affect their usefulness and search experience to the user. Even if individual query suggestions are highly related to the user's original query, he may find it difficult to choose a good suggestion from a large list, or to understand the relation between an input query and suggested ones.

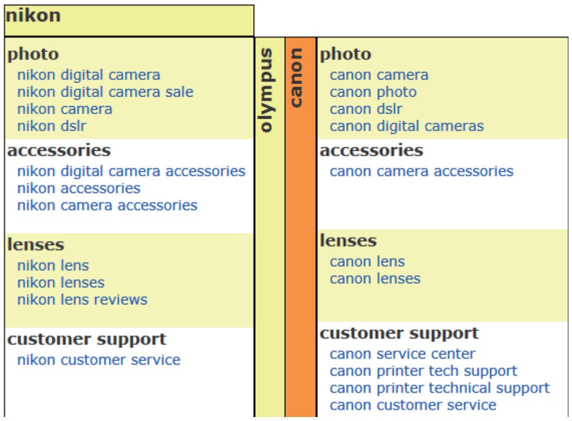
Traditionally, query suggestions are provided to the user in a flat list, as in commercial Web search engines such as Google and Bing. Recently, some richer methods of presenting query suggestions have been explored. For example, Sadikov *et al.* [21] clustered query suggestions using clickthrough and session data. As mentioned in their work, clustering query suggestions can save the user's time and effort to locate a suggestion that serves his need. Going one step further, Guo *et al.* [10] automatically labeled query suggestion clusters based on social annotation. This may help the user understand the meaning of each query suggestion more easily. The first contribution of the present study is that we demonstrate the effect of query suggestion presentation methods on the user's search behavior. To be more specific, our user study shows that subjects using a flat list query suggestion interface and those using a richer query suggestion interface behaved significantly differently even though the set of query suggestions presented was exactly the same.

The second contribution is a new query suggestion presentation method that aims to support different types of query reformulation seamlessly. According to Boldi *et al.* [6], there are two major types of query reformulation: *specialization*, in which a broad or ambiguous query is modified to narrow down the search result; and *parallel movement*, in which the user's topic of interest shifts to another with similar aspects. For example, if the user's original query is "nikon" and he selects "nikon camera" from the query suggestion list, this action is a specialization. Moreover, if the same user who selected the "nikon camera" query is interested in buying *cameras* rather than in the *Nikon* brand, then alternative brands such as "canon ixy" and "olympus camera" may be useful to him as query suggestions. Thus, he may select "canon ixy" from the query suggestion list and may end up buying a Canon camera, even if he may not have known this company at the outset. This action is a parallel movement. Boldi *et al.* reported that these two types of query reformulation constituted 30-38% and 48-56% of a Yahoo! search query log, respectively.

Although the aforementioned two query suggestion presentation methods [10, 21] are effective for either specialization or parallel movement, they do not necessarily support these two types of query

[*] This research was conducted while the first author was an intern at Microsoft Research Asia.

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.



nikon

photo
nikon digital camera
nikon digital camera sale
nikon camera
nikon dslr

accessories
nikon digital camera accessories
nikon accessories
nikon camera accessories

lenses
nikon lens
nikon lenses
nikon lens reviews

customer support
nikon customer service

olympus

canon

photo
canon camera
canon photo
canon dslr
canon digital cameras

accessories
canon camera accessories

lenses
canon lens
canon lenses

customer support
canon service center
canon printer tech support
canon printer technical support
canon customer service

Figure 1: Screenshot of the SParQS interface.

reformulation at the same time. For example, given the current query "nikon," it is not clear how query suggestions like "canon ixy" should be presented to the user, as Canon is an alternative camera brand. If the user wants to select a query suggestion strictly related to Nikon, then showing "nikon camera" and "canon ixy" in the same cluster would be confusing for the user. On the other hand, if the user is open to alternatives such as Canon cameras, then showing both "nikon camera" and "canon ixy" together in some way might help the user. Thus, it may be difficult for simple clustering approaches to support specialization and parallel movement simultaneously.

Motivated by the above observations, we propose a new query suggestion presentation method that aims to support the user's specialization and parallel movement activities seamlessly. We call our prototype system *Specialization and* ***Par****allel movement* ***Q****uery* ***S****uggestion* (SParQS): a screenshot of the SParQS interface is shown in Figure 1. In this example, the current query is "nikon," and some specialization suggestions for "nikon" are classified into several categories such as **photo** and **accessories** in a way similar to the presentation method proposed by Guo *et al.* [10]. Moreover, to support parallel movement, alternative *entities* "canon" and "olympus" are presented vertically, and *their* specialization suggestions are optionally shown to the user. (In Figure 1, "canon" is selected as the current alternative entity.) Note that the specialization suggestions for the original query "nikon" and those for an alternative entity "canon" are arranged in the same way, under common *aspects* such as **photo** and **accessories**. Our hope is that the SParQS interface will encourage the user to compare across multiple entities based on a certain aspect: for example, comparing different camera makers in terms of camera lenses. Moreover, in Figure 1, note that the user can go directly from the current query "nikon" to "canon digital cameras". This is a combination of parallel movement and specialization, which we call *diagonal movement*.

By default, SParQS only shows the classified query suggestions for the current query (e.g., information presented beneath "nikon" in Figure 1), plus the alternative entity names ("canon" and "olympus" shown vertically in Figure 1). The query suggestions for an alternative entity is shown only when that entity is clicked by the user. This is because presenting too much query suggestion information to the user unconditionally may have a negative impact on his search activity.

The SParQS back-end algorithm relies on a log of queries and clicked URLs from Microsoft's Bing[1]. First, SParQS clusters entities (e.g. "nikon," "canon" and "olympus" in Figure 1) based on queries that contain these entities. Second, it clusters queries based on clicked URLs, and selects some clusters as query suggestion categories (e.g. **photo** and **accessories** in Figure 1). Finally, it classifies query suggestions into labeled categories that are shared across entities. Although SParQS can help the user only when the current query contains a named entity, previous research suggests that this is not a major limitation: it has been reported that at least 20-30% of queries input to Microsoft's Bing are named entities [30], and that over 70% of search queries contain named entities [11].

The main objective of the present study is to demonstrate the effect of query suggestion presentation methods and the advantages of the SParQS interface over a flat list query suggestion interface in search tasks. To this end, we designed ten *Information Gathering tasks* (e.g. "Find information about Bill Clinton") and ten *Entity Comparison tasks* (e.g. "Find cars made by Mazda and other manufacturers"), and conducted user experiments involving 20 subjects. Our results show that the SParQS interface enables the subjects to search more successfully than the flat list case, and is particularly advantageous for Entity Comparison tasks. In addition, the subjects found the query suggestions more helpful when they were presented in the SParQS interface rather than in a flat list.

The remainder of this paper proceeds as follows. Section 2 briefly surveys prior art in query suggestion algorithms, its presentation, and knowledge acquisition from query logs. Section 3 formalizes the query suggestion classification problem and Section 4 describes our approach to solving the problem. Section 5 describes the data we input and generated from the SParQS back-end algorithm for the purpose of evaluating the SParQS interface. Section 6 reports on the user study in which we compared the SParQS interface with a flat list one. Finally, Section 7 concludes this paper and discusses future work.

2. RELATED WORK

2.1 Query Suggestion Algorithms

Various query suggestion algorithms have been proposed in the literature. One popular approach is to cluster queries based on the *clicked URLs* within the corresponding search engine result pages. Given a query, its query cluster can be identified, and the other members of that cluster can be presented as query suggestions [4, 7]. Another approach is to incorporate random walk or hitting time in a query and clicked URL bipartite graph [18, 19, 25]. Other studies utilized query sequences to model the user behavior, in order to predict queries that are likely to follow a given query [2, 5, 12]. A recent study tackles the problem of diversifying query suggestions by diversifying their top-returned search results [26].

Any of the above algorithms could be plugged into SParQS, as our problem formulation assumes that query suggestions are already available as input. In the experiments reported in this paper, we use the well-known *hitting time algorithm* [19].

2.2 Query Suggestion Presentation

A variety of search user interfaces have been studied from the viewpoints of usability and the effect on the user's search activities. For example, some interfaces present classified or faceted search results [9, 29]. Recently, as query suggestion has become a common feature of Web search engines, studies on query suggestion presentation have begun to emerge.

Kelly *et al.* [15] investigated the effect of presenting the usage statistics of each query suggestion to the user. Their experiments used four topics, each with eight query suggestions: four frequently-used suggestions and four rarely-used ones. Each query suggestion was accompanied by information as to how many other people used that suggestion. They found that the use frequency of

[1] http://www.bing.com/

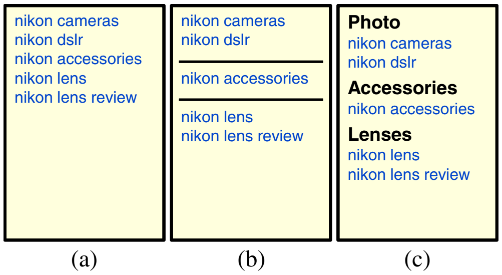


Figure 2: Examples of (a) flat list, (b) clustering and (c) clustering-with-labels interfaces.

suggestions did not affect the subjects. Kelly, Gyllstrom and Baily [16] studied the difference between "term suggestion" and query suggestion. A term suggestion system they developed enables the user to add terms to his original query by clicking on each suggestion term. In their interactive information retrieval study with 55 subjects and 20 topics, subjects preferred query suggestion to term suggestion.

To clarify the differences between SParQS and other typical query suggestion presentation methods, Figure 2 shows examples of (a) flat list, (b) clustering and (c) clustering-with-labels interfaces, which should be compared with our SParQS interface shown in Figure 1. The flat list represents the query suggestion method adopted by traditional search engines. The clustering interface corresponds to work on query suggestion clustering such as that of Sadikov *et al.* [21], who used a random walk model in a query-query and query-document transition graph. The clustering-with-labels interface represents a method proposed by Guo *et al.* [10], who utilized social annotation data for labeling query suggestion clusters.

SParQS differs from these existing query suggestion methods in the following two aspects. First, SParQS provides new entities as alternatives to the current entity, and their query suggestions: see "canon" and "olympus" shown in Figure 1. Second, in contrast to these bottom-up clustering approaches, we first generate categories and then classify query suggestions into these categories. This is because we want categories that apply not only to the current entity but also to the alternative entities: see **photo**, **accessories** etc. in Figure 1. Moreover, while previous work on query suggestion focused mainly on intrinsic evaluation (e.g. accuracy), the main objective of this study is to demonstrate the effect of query suggestion presentation methods and the advantages of the SParQS interface over the traditional flat list one through task-based evaluation.

There are also existing studies on search functionalities that are related to query suggestion but are different: namely, *interactive query expansion* [3] and *query completion* [1, 14, 28]. Interactive query expansion is basically the same as the aforementioned term suggestion, but it appears to have been replaced by query suggestion during the last decade. In contrast, query completion is as common a feature as query suggestion in current search engines and they probably complement each other: while query suggestion provides a static list of possible queries *given a complete initial query*, query completion aims to provide a dynamic list of possibilities *given a prefix string of an initial query*, often within the search query box. Amin *et al.* [1] conducted user studies that investigate organization strategies of autocompletion such as alphabetical ordering and grouping. However, their user study focused on autocompleting an entity name rather than the search query (e.g. for inputting a location name in a Web form).

2.3 Knowledge Acquisition from Query Logs

Query logs are often utilized for acquiring knowledge such as entity names and search intents. For example, some studies [17, 23] extracted named entities from query log data by means of *query contexts*, which are generated by replacing known entity names in queries with a wildcard. The query contexts are then used to capture new entity names. In the present study, SParQS also utilizes query contexts to find alternative entities given a query containing an entity.

Table 1: Symbols with examples.

Symbol	Example
E	E ={"canon", "nikon", "microsoft", ...}
Q	Q ={"nikon photo", "microsoft windows", ...}
U	U ={"`http://www.bing.com/`",...}
S_j	S_2 ={"nikon camera", "nikon lens", ...}
Q_i	Q_1 ={"nikon photo", "canon printer", ...}
$\mathcal{Q}_i = \{Q_i^{(1)}, \ldots\}$	$Q_1^{(2)}$ = {"olympus printer", "canon printer", ...}
$\mathcal{E} = \{E_1, \ldots\}$	E_1 ={"olympus", "nikon", "canon"}
$\mathcal{Q}_i^* = \{Q_i^{*(1)}, \ldots\}$	$Q_1^{*(3)}$ ={"nikon photo", "canon camera", ...}
$Y_i = \{y_i^{(1)}, \ldots\}$	$y_1^{(3)}$ =**photo**
$\mathcal{S}_j^* = \{S_j^{*(1)}, \ldots\}$	$S_2^{*(3)}$ ={"nikon camera", "nikon dslr", ...}

Query logs are also used for mining search intents. For example, Wen *et al.* [27] proposed a query clustering method based on content similarity, e.g. term overlaps between the queries or the similarity between clicked URLs, to organize common intents. Yin and Shah [30] built a taxonomy of Web search intents from clickthrough data, by extracting is-a relationships among queries based on clicked URLs and constructing a tree whose nodes are queries containing a named entity. In contrast to these approaches, SParQS first creates categories by means of query clustering, and then classifies query suggestions into these categories based on the similarity between clicked URLs of queries and those of query suggestions.

3. PROBLEM DEFINITION

As was mentioned earlier, the SParQS back-end first clusters entities, then clusters queries, and finally classifies query suggestions into categories. This section formalizes these three problems.

3.1 Input

We require the following as input:

1. E: a set of entities, which can be prepared, for example, by extracting Wikipedia entry titles or leveraging named entity dictionaries;

2. Q: a set of queries;

3. U: a set of URLs;

4. $w(q, u)$: a click count function, which indicates how many times a URL $u \in U$ presented in response to a query $q \in Q$ has been clicked;

5. S_j: a set of query suggestions for each entity $e_j \in E$; and

6. n: the number of query suggestion categories required.

Examples are shown in Table 1. The input items 2-4 are what we call *clickthrough data*, which can be represented as a bipartite graph with weighted edges. The sets of queries Q and URLs U are nodes, and there is an edge between a query $q \in Q$ and URL $u \in U$ in the bipartite graph iff the URL u presented in response to the query q has been clicked. Each edge is weighted by the click count function $w(q, u)$. A set of query suggestions S_j for each entity $e_j \in E$ is also an input item in this problem. Thus, the problem is independent of how query suggestions are generated.

3.2 Output

We require the following as output (See also Table 1):

1. $\mathcal{E} = \{E_1, E_2, \ldots\}$: a set of entity clusters, obtained by clustering entities in E based on queries that contain the entities;
2. $\mathcal{Q}_i^* = \{Q_i^{*(1)}, \ldots, Q_i^{*(n)}\}$: a set of query suggestion *categories* for each entity cluster E_i, such that $\mathcal{Q}_i^*$ is a subset of a query cluster class $\mathcal{Q}_i = \{Q_i^{(1)}, Q_i^{(2)}, \ldots\}$. The class of query clusters $\mathcal{Q}_i$ is obtained by clustering queries in Q_i, the set of queries that contain any entity from E_i;
3. $Y_i = \{y_i^{(1)}, \ldots, y_i^{(n)}\}$: a set of labels for each entity cluster E_i, where a label $y_i^{(k)}$ is attached to a category $Q_i^{*(k)}$; and
4. $\mathcal{S}_j^* = \{S_j^{*(1)}, \ldots, S_j^{*(n)}\}$: a class of query suggestion sets, where $S_j^{*(k)}$ represents the query suggestion set for an entity $e_j \in E_i$, which has been classified into a category $Q_i^{*(k)}$. The query suggestion set $S_j^{*(k)}$ is a subset of S_j, which is the full set of suggestions for an entity e_j.

The SParQS interface utilizes the above output as follows: in response to a query containing an entity e_j from an entity set E_i, a set of query suggestions S_j is presented to the user. The query suggestions in S_j are classified into categories $\mathcal{Q}_i^*$, and each category $Q_i^{*(k)} \in \mathcal{Q}_i^*$ is a category for query suggestions in $S_j^{*(k)}$. For example, suppose that $E_1 = \{$"olympus", "nikon", "canon"$\}$, and e_2 = "nikon" as shown in Table 1. Given a query "nikon," the SParQS presents its query suggestions S_2 such as "nikon camera" and "nikon lens." The query "nikon camera," which is contained in a set of query suggestions $S_2^{*(3)}$, is classified into a category $Q_1^{*(3)}$ with a label $y_1^{(3)} =$ **photo**. The other entities in the entity set E_1, and their query suggestions classified into the categories $\mathcal{Q}_i^*$, are also presented to the user.

Moreover, we require the above output items to satisfy the following criteria:

Evenness of Categories.

Categories should be chosen so that they are equally useful to all the entities in the cluster. For example, given the entity cluster $\{$"nikon", "canon", "olympus"$\}$, **ixy** is not a good category label as it is a digital camera brand by Canon and does not apply to the other two entities.

Specificity of Categories.

Categories should be chosen so that they are neither too broad nor too narrow. For example, given the entity cluster $\{$"nikon", "canon", "olympus"$\}$, a category **product** may not be specific enough (even though this does satisfy the evenness requirement), and may result in too many suggestions per category. On the other hand, if a category is so specific that it only applies to one particular query suggestion, then the very idea of classifying query suggestions becomes meaningless.

Accuracy of Suggestion Classification.

Given a set of categories, query suggestions should be classified appropriately. For example, "canon printer" should not be classified into the **photo** category. This would only confuse the user.

Note that we have not stated any explicit requirements for entity clustering. However, it follows from the above three requirements that entities should be clustered so that we can obtain categories that are evenly spread across the clustered entities. Subsequently, at the query clustering stage, the evenness and specificity requirements determine the appropriate query cluster size.

4. SParQS BACKEND ALGORITHM

Given the requirements discussed in the previous section, this section explains how the SParQS back-end algorithm works. Because it is difficult to optimize in terms of the aforementioned evenness, specificity and the accuracy at the same time, our approach decomposes the problem into three subproblems and tackles them one by one. First, entities are clustered based on queries that contain the entities. Second, queries are clustered based on their clicked URLs, some of which are selected as query suggestion categories. Finally, we classify query suggestions into the categories that are shared across entities in an entity cluster.

4.1 Clustering Entities

Our first step is to cluster a given set of entities using the method utilized by Sekine and Suzuki [23] and later by Komachi and Suzuki [17]. From a query log, query contexts are obtained for each entity by replacing the occurrences of the entity in queries with a wildcard. For example, given the entity "canon" and queries "canon camera" and "price canon camera," query contexts "$*$ camera" and "price $*$ camera" are obtained.

Formally, a context c is obtained by replacing an entity e in a query "*prefix e suffix*" with a wildcard, i.e. $c =$ "*prefix* $*$ *suffix*," where *prefix* and *suffix* are strings that may be empty. We denote the original query "*prefix e suffix*" by $c(e)$ (obtained by substituting e into c). Given the entity set E and the query set Q, we can define a set of query contexts $C = \{c | c(e) \in Q \land e \in E\}$. Using the l-th query context c_l in C, we define an *entity vector* $\mathbf{v}_e$, whose l-th element is defined as $v_{e,l} = \sum_{u \in U} w(c_l(e), u)$. Recall that $w(q, u)$ is the number of times a URL u has been clicked in response to the query q. Thus, the l-th element is the number of times any URL was clicked after query $c_l(e)$ was input.

The similarity between entities is simply defined as the cosine between entity vectors. For clustering the entity vectors, we conducted some trial experiments and finally chose to use a standard group-average method with cosine similarity. By applying a threshold to hierarchical clusters, we obtain a set of entity clusters $\mathcal{E} = \{E_1, E_2, \ldots\}$. For example, $E_1 = \{$"olympus", "nikon", "canon"$\}$ as shown in Table 1.

4.2 Clustering Queries

Having clustered the entities from E, we obtain a query set $Q_i = \{c(e) | c(e) \in Q \land e \in E_i\}$ (see example in Table 1). That is, this is the set of queries that contain any entity from E_i. We now need to cluster the queries in Q_i. Although some work on the query suggestion and query clustering uses clickthrough data [4, 7, 18, 19, 25, 27], these methods are not directly applicable to our problem. To be more specific, if we cluster queries simply based on clicked URLs, we would likely obtain query clusters where each of them corresponds to a single entity. For example, consider two queries "nikon camera" and "canon camera." The search results for these queries would be completely different, and therefore these queries would not have any clicked URLs in common. Building on such data, traditional query clustering would result in a cluster specifically for "nikon," another cluster specifically for "canon," and so on. In contrast, as our evenness requirement dictates, we want query clusters that apply to multiple entities. Thus, in order to bridge the gaps across different entities, we utilize the idea of query context described earlier, and regard queries that contain different entities but share the same context as identical in the query clustering process. In the above example, "nikon camera" and "canon camera" are considered identical when summing up the click counts.

Using the m-th URL u_m in U, a query q is usually represented as a vector whose m-th element is a URL click count $w(q, u_m)$. In our approach, a query $q = c(e) \in Q_i$ is represented as a vector whose element is the sum of click counts of queries that have the same context c. Thus, the m-th element of a query vector $\mathbf{v}_q$ for the query q is defined as $v_{q,m} = \sum_{e_j \in E_i} w(c(e_j), u_m)$.

In the same way as the entity clustering, queries in Q_i are clustered based on query vectors, which result in a set of query clusters $\mathcal{Q}_i = \{Q_i^{(1)}, Q_i^{(2)}, \ldots\}$ for each entity cluster E_i (an example of $\mathcal{Q}_i$ is shown in Table 1). The categories for E_i will be selected from these query clusters, as we shall describe below.

4.3 Classifying Query Suggestions

Now, we have obtained entity clusters $\mathcal{E} = \{E_1, E_2, \ldots\}$, and query clusters $\mathcal{Q}_i = \{Q_i^{(1)}, Q_i^{(2)}, \ldots\}$ for each entity cluster E_i. In the next step, query suggestions of entities from E_i are classified into each category selected from the query clusters $\mathcal{Q}_i$. For the sake of simplicity, the index i of an entity cluster E_i and a set of query clusters $\mathcal{Q}_i$ for the entity cluster is omitted hereinafter. Thus, we refer to E_i and $\mathcal{Q}_i$ as E and $\mathcal{Q}$, respectively.

A query suggestion can be classified into a query cluster if their similarity is greater than a threshold θ. A query cluster $Q^{(k)} \in \mathcal{Q}$ is represented as a vector $\mathbf{v}_{Q^{(k)}}$ that is the sum of all the query vectors in the query cluster, i.e. $\mathbf{v}_{Q^{(k)}} = \sum_{q \in Q^{(k)}} \mathbf{v}_q$. We represent a query suggestion s as a vector whose m-th element is the click count $w(s, u_m)$. We define the similarity between a query cluster $Q^{(k)}$ and query suggestion s simply as the cosine between their vectors, which we denote by $\mathrm{Sim}(Q^{(k)}, s)$. If $\mathrm{Sim}(Q^{(k)}, s) \geq \theta$, we can classify a query suggestion s into a query cluster $Q^{(k)}$. For each query suggestion, we select exactly one category that satisfies the similarity constraint.

We then choose n query clusters as categories to classify query suggestions, which we denote by $\mathcal{Q}'(\subset \mathcal{Q})$, where $|\mathcal{Q}'| = n$. Recall that the three criteria should be satisfied for this classification, i.e. evenness, specificity, and accuracy. The accuracy criterion corresponds to the similarity constraint, i.e. we can classify a query suggestion s to a category $Q^{(k)}$ if $\mathrm{Sim}(Q^{(k)}, s) \geq \theta$. Below, we describe how to meet the other two criteria.

Evenness dictates that categories should be chosen so that query suggestions classified into a category are spread evenly across entities. We can express this criterion in the form of *query suggestion entropy over entities*. For an entity cluster E and category $Q^{(k)} \in \mathcal{Q}'$, the entropy is defined as follows:

$$H_k(E) = -\sum_{e_j \in E} P_k(e_j) \log P_k(e_j), \qquad (1)$$

where $P_k(e_j)$ represents the probability that a query suggestion classified into a category $Q^{(k)}$ is of an entity e_j. Letting $S_j^{(k)}$ be query suggestions of an entity e_j classified into a category $Q^{(k)}$, the probability $P_k(e_j)$ is defined as follows:

$$P_k(e_j) = \frac{|S_j^{(k)}| + \alpha}{\sum_{e_l \in E} |S_l^{(k)}| + \alpha|E|}, \qquad (2)$$

where α is a smoothing factor to avoid zero for $P_k(e_j)$.

According to the nature of entropy, $H_k(E)$ becomes higher as query suggestions classified into a category $Q^{(k)}$ are distributed more evenly across entities. Thus, we want to choose clusters that achieve a high value of $H_k(E)$, which represents high evenness.

Specificity dictates that categories should be chosen so that they are neither too broad nor too narrow. We express this criterion as *query suggestion entropy over categories*. For an entity $e_j \in E$ and set of categories $\mathcal{Q}'$, the entropy is defined as follows:

$$H_j(\mathcal{Q}') = -\sum_{Q^{(k)} \in \mathcal{Q}'} P_j(Q^{(k)}) \log P_j(Q^{(k)}), \qquad (3)$$

where $P_j(Q^{(k)})$ represents the probability that a query suggestion of an entity e_j is classified into a category $Q^{(k)}$. The probability $P_j(Q^{(k)})$ is defined as follows:

$$P_j(Q^{(k)}) = \frac{|S_j^{(k)}| + \alpha}{\sum_{Q^{(l)} \in \mathcal{Q}'} |S_j^{(l)}| + \alpha|\mathcal{Q}'|}. \qquad (4)$$

The entropy $H_j(\mathcal{Q}')$ becomes higher as query suggestions of an entity e_j are distributed more evenly across categories. A high $H_j(\mathcal{Q}')$ indicates appropriate specificity, since $H_j(\mathcal{Q}')$ achieves its maximum when given query suggestions are exactly evenly classified into categories, i.e. when we have the same number of suggestions per category.

Combining the two entropies, we set an objective function f to be maximized:

$$f(\mathcal{Q}') = \lambda \sum_{Q^{(k)} \in \mathcal{Q}'} H_k(E) + (1 - \lambda) \sum_{e_j \in E} H_j(\mathcal{Q}'), \qquad (5)$$

where λ is a parameter that determines whether evenness or specificity should be emphasized, and $0 \leq \lambda \leq 1$.

Finally, we obtain the set of categories $\mathcal{Q}^*$ that maximizes the objective function f. The objective function f takes a set of categories $\mathcal{Q}'$, and finding the set $\mathcal{Q}^*$ that maximizes the objective function f is NP-hard. If the set function f were *monotonic* and *submodular*, it would be guaranteed that a simple greedy algorithm returns the $(1 - 1/e)$-approximation of the maximum [20]. In our case, however, the function f is neither monotonic nor submodular because of the difficulty of the classification problem.

The major difficulty is that the classification of query suggestions is nondeterministic while selecting categories. As a query suggestion can be classified into exactly one category while there are many candidates of categories, we cannot determine which category a query suggestion should be classified into until all the categories are selected. On the other hand, categories cannot be selected from query clusters without classified query suggestions, since the objective function f relies on how query suggestions are classified. This is why the function f is neither monotonic nor submodular. The objective function f can increase or decrease depending on classified query suggestions.

We therefore propose a greedy algorithm to handle the selection of categories and classification of query suggestions simultaneously. A query cluster is incrementally added to a set of categories that achieves the maximum value of the objective function f when the query cluster is added. The greedy algorithm is described in Algorithm 1. This algorithm repeats two processes until the number of selected categories reaches n: tentative classification of query suggestions (lines 4-13), and selection of the best query cluster as a category (lines 15-22). First, all the query suggestions are tentatively classified into query clusters. A query suggestion s in a set of query suggestions S_j for an entity $e_j \in E$ is tentatively classified into a query cluster $Q^{(l)} \in \mathcal{Q}$ if their similarity $\mathrm{Sim}(Q^{(l)}, s)$ is greater than a parameter θ (lines 8-10). Second, a query cluster is chosen as a category that maximizes the objective function f in Equation 5, using tentatively classified query suggestions (lines 15-17). A selected query cluster $Q^{(x)}$ is removed from a whole query cluster set $\mathcal{Q}$ (line 18). Query suggestions in $S_j^{(x)}$ for each entity e_j are classified into the selected query cluster, or a category $Q^{(x)}$, and also removed from a whole query suggestion set (lines 19-22).

Finally, a label $y^{(k)}$ for each category $Q^{*(k)}$ is selected from queries in the category. We select the query that maximizes the

Algorithm 1 A greedy algorithm

```
Input: An entity cluster $E$, a set of query clusters $\mathcal{Q}$, and a set of
    query suggestions $S_j$ for each entity $e_j \in E$.
Output: Categories $\mathcal{Q}^* = \{Q^{*(1)}, \ldots\}$, and classified query sug-
    gestion sets $\mathcal{S}_j^* = \{S_j^{*(1)}, \ldots\}$ for each entity $e_j \in E$.
$\mathcal{Q}' = \{\}$
for $k = 1$ to $n$ do
    /* tentative classification of query suggestions */
    for each $Q^{(l)} \in \mathcal{Q}$ do
        for each $e_j \in E$ do
            $S_j^{(l)} = \{\}$
            for each $s \in S_j$ do
                if $\mathrm{Sim}(Q^{(l)}, s) \geq \theta$ then
                    $S_j^{(l)} = S_j^{(l)} \cup \{s\}$
                end if
            end for
        end for
    end for
    /* selection of the best query cluster as a category */
    $x = \operatorname{argmax}_l f(\mathcal{Q}' \cup \{Q^{(l)}\})$
    $\mathcal{Q}' = \mathcal{Q}' \cup Q^{(x)}$
    $Q^{*(k)} = Q^{(x)}$
    $\mathcal{Q} = \mathcal{Q} - \{Q^{(x)}\}$
    for each $e_j \in E$ do
        $S_j^{*(k)} = S_j^{(x)}$
        $S_j = S_j - S_j^{(x)}$
    end for
end for
```

similarity to any one of the query suggestions in the category. Formally, a label $y^{(k)}$ for a category $Q^{*(k)}$ is defined as follows:

$$y^{(k)} = \operatorname*{argmax}_{q \in Q^{*(k)}} \frac{\mathbf{v}_q^T \mathbf{v}_s}{\|\mathbf{v}_q\| \|\mathbf{v}_s\|}, \qquad (6)$$

where the variable s represents any query suggestion classified to the category $Q^{*(k)}$, i.e. $s \in \bigcup_{e_j \in E} S_j^{*(k)}$. When presenting a query as a label to the user, the entity is removed from the query.

5. DATA, QUALITY AND ACCURACY

For conducting a task-based user study to evaluate the SParQS interface, we applied the SParQS back-end algorithm to Bing's query log. This section describes the query log used, and the application of the SParQS back-end algorithm to the data. To ensure that the resultant data are of sufficient quality that deserve a user study, this section also reports on the quality of category labels as well as the accuracy of query suggestion classification.

5.1 Data

We used Microsoft Bing's query log from April 25th to May 1st, 2010, where each record consists of a query and clicked URL. The number of records is 3,503,469,327, which contains 76,462,963 unique queries, and 62,978,872 unique URLs. As the SParQS algorithm also requires named entity lists as input, we also manually gathered lists of entities from Web pages for five entity classes: *Company*, *Person*, *Landmark*, *City*, and *Product*. These five entity classes are the top categories from the Extended Named Entity Hierarchy proposed by Sekine *et al.* [22]. The total number of entities is 5,156, which consists of 2,000 companies, 119 people, 1203 landmarks, 388 cities, and 1,446 products.

Table 2: Evaluations of category labels. Bold font indicates the highest value in each row.

	λ				
	0.00	0.25	0.50	0.75	1.00
Assessor 1	0.642	0.655	**0.666**	0.659	0.629
Assessor 2	0.686	0.699	**0.719**	0.716	0.711
Intersection	0.591	0.606	**0.631**	0.619	0.597

To generate query suggestions, we applied the hitting time algorithm proposed by Mei *et al.* [19]. First, we constructed a click-through bipartite graph, where queries and URLs are nodes and each edge represents a URL click in a search engine result page produced in response to a query. The click count $w(q, u)$ is used as the weight of each edge. For every query pair, we computed the hitting time of a random walk, defined as the expected number of steps it takes from a query to another on the bipartite graph. Finally, we selected 20 queries that are the "closest" to each query.

We clustered entities and queries based on the method described in Section 4. Before the clustering, we filtered out queries that occur less than 10 times in the query log. Entity and query vectors in Section 4.1 and 4.2 were weighted by a standard TF-IDF method, and clustered with a group-average clustering method as was mentioned in Section 4.1. Based on some preliminary experiments, the thresholds for entity clustering and query clustering were set to 0.25 and 0.20, respectively. Then, we selected categories and classified query suggestions into the categories using the greedy algorithm described in Algorithm 1. The number of categories n was set to 5, and the parameter θ, the similarity threshold for query suggestion classification, was set to 0.30. As this data construction is primarily for our user study rather than intrinsic evaluation, we did not use any held-out data for parameter tuning.

For an algorithm evaluation, we manually chose 20 entity clusters that had at least two entities from each of the five entity classes. For example, entity clusters such as {"nikon", "canon", "olympus"} and {"sharp", "samsung", "lg", "sony", "panasonic"} were chosen from an entity class *Company*. We hired four assessors for this evaluation, who are graduate students majoring in computer science. Two of them evaluated categories of 100 entity clusters with five types of values for a parameter λ, which determines whether evenness or specificity should be emphasized in the objective function f in Equation 5. We asked the other two assessors to evaluate classified query suggestions for 390 entities.

5.2 Quality of Categories

The two assessors evaluated the quality of the category labels generated by SParQS. We showed the two assessors a list of category labels, a set of entities, and their unclassified query suggestions. They then were asked to rate each label on a 3-point scale: *highly relevant* (appropriate for classifying the query suggestions); *somewhat relevant* (somewhat awkward but may be useful for classifying the suggestions); and *irrelevant* (not appropriate).

For five parameter settings: $\lambda = 0.0$, 0.25, 0.5, 0.75, and 1.0, the two assessors evaluated 495, 493, 492, 490, and 489 categories of 100 entity clusters, respectively (2,459 categories in total). The inter-assessor agreement was substantial: 0.616 in terms of *quadratic-weighted kappa* [24]. The results for the five values of λ (see Eq. 5) are summarized in Table 2, where rows "Assessor 1" and "Assessor 2" show precisions computed based on evaluations by each assessor, and a row "Intersection" shows precisions computed based on only evaluations agreed among the two assessors. Here, precision is defined as the number of highly or somewhat relevant categories divided by the total number of evaluated categories. Note that $\lambda = 0.0$ means using the entropy over categories (specificity) only, and $\lambda = 1.0$ means using the entropy over entities (evenness) only. It can be observed in all the rows that $\lambda = 0.5$, which combines the two entropy measures, is slightly more accurate than when only one of them is used. Although the difference across the five parameters is not significant, this result shows that the two criteria, i.e. specificity and evenness, are both useful for obtaining good categories. Thus, we used $\lambda = 0.5$ for the evaluation of query suggestion classification as well as the user study.

It is difficult to say whether the highest precision of 0.631 in the row "Intersection" is "good" or not as previous work [10] does not report on the accuracy of category labels. However, there is clearly a lot of room for improvement. Our failure analysis shows that many of our categories were good for only one particular entity and not for all the entities even though we incorporated the entropy measure for evenness. Note that if multiple entities simply lack common aspects, then it is simply impossible for SParQS to generate labels that apply to all of the entities. That is, the entity clustering stage may impose an upperbound on the quality of category labels.

5.3 Accuracy of Suggestion Classification

The other two assessors examined whether the query suggestions are appropriately classified. They judged whether it makes sense to classify each query suggestion into the given category. The assessment was done on a 3-point scale: *irrelevant*, *neutral*, and *relevant*. The assessors were asked to consider only the relationship between the category and the query suggestion, and not the quality of the category label or the suggestion.

The two assessors evaluated 2,836 classified query suggestions of 390 entities (about seven query suggestions per entity), and the inter-assessor agreement after excluding the neutral judgments was substantial: 0.700 in kappa coefficient. The precision computed based on the same data, defined as the number of suggestions judged relevant divided by the total number of suggestions, was 0.630. Again, we could not find an appropriate baseline in related work on query suggestion. While our task here is query suggestion *classification* given the categories and query suggestions, previous work evaluated query *cluster* quality [21] or the quality of the query suggestions themselves [10]. Here, we refer to a study on building a taxonomy from clickthrough data by Yin and Shah [30]. They proposed a method to extract is-a relationships among queries based on clicked URLs, and evaluated this method in terms of precision and recall. The precision of their method ranged from 0.125 to 0.195 (is-a relationships between queries), and from 0.333 to 0.810 (is-a relationships between aggregated queries). The aggregated queries in their study are similar to our query contexts. However, we cannot directly compare our results with theirs, as our problem is to classify a query into categories (i.e. groups of aggregated queries) rather than to classify a query into queries, or an aggregated query into aggregated queries.

While we see a lot of room for improvement in the current implementation of our SParQS back-end algorithm, we observed that the results are reasonable and are adequate for our user study to investigate the advantages of the SParQS interface.

6. USER STUDY

To examine the effect of the SParQS interface on users' search behaviors, we conducted a user study similar to the Text Retrieval Conference (TREC) Interactive Track [8]. We prepared 20 tasks, hired 20 subjects and asked them to collect answers relevant to each task within five minutes. For each task, each subject used either the SParQS interface, or a flat list interface as a baseline to complete the task.

As SParQS can only handle queries containing a named entity, we designed search tasks that involve such queries where query suggestion might be useful. As we discussed in Section 1 using the query "nikon" as an example, search tasks that are initiated by a named entity query can generally be classified into two types: finding information about the given entity (e.g. Nikon cameras), and finding information about entities related to the given one in terms of a particular aspect (e.g. competitors such as Canon and Olympus). We thus devised 10 *Information Gathering tasks* and 10 *Entity Comparison tasks*: Information Gathering tasks require the subject to gather information about a given entity from several aspects (e.g. "Find information about Bill Clinton"), while Entity Comparison tasks require the subject to gather information about several different entities in terms of a particular aspect for comparison (e.g. "Find cars made by Mazda and other manufacturers"). In Information Gathering tasks, subjects may possibly utilize query suggestions for specialization in order to focus on a certain aspect of the input entity, while in Entity Comparison tasks, they may possibly use query suggestions for parallel movement to shift their attention to another entity. While these tasks were specifically designed to examine the usefulness of SParQS for specialization and parallel movement and are not "real" search tasks obtained from (say) session data, we argue that it is useful to clarify exactly for what kinds of search tasks SParQS can effectively help the user through our experiments.

6.1 System Design

To conduct our user study, we implemented a system that can present the query suggestion data using either the SParQS interface as shown in Figure 3 or the flat list interface as shown in Figure 4. Query suggestions, generated as was described in Section 5.1, are exactly the same in the two interfaces: only how they are presented is different. To further ensure that exactly the same information is available on the two interfaces, our flat list interface also shows the alternative-entity suggestions obtained from SParQS under regular query suggestions[2]. In Figure 4, for example, alternatives to the query "armani" such as "tom ford" and "hugo boss" are shown under query suggestions for "armani."

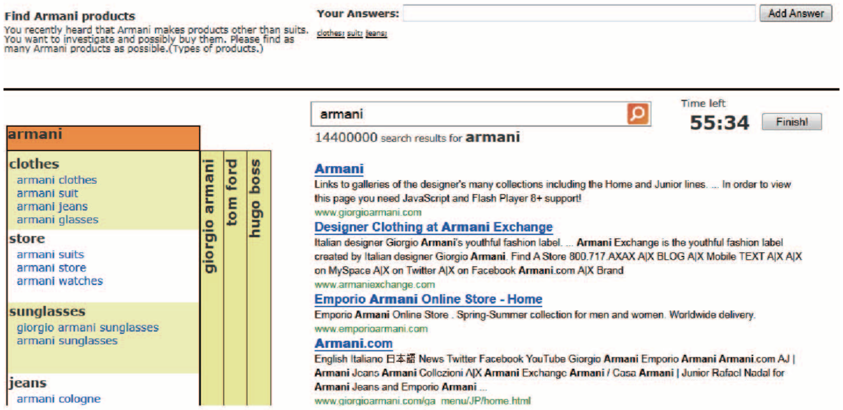
When a query suggestion is clicked, or the subject inputs a query to the search form and clicks the search button, the system retrieves Web pages using Bing API[3]. Subjects were required to collect answers to each given task from either the Bing snippets or the actual body of the retrieved Web pages. Subjects can enter an answer either by typing or by copy-and-paste from a snippet or a Web page. As shown near the top of Figure 3, we provided an answer input box with an Add Answer button so that subjects can submit answers one by one. For a quantitative analysis, we recorded collected answers, queries input, suggestions utilized, documents viewed, and entities to which one or more answers were given.

6.2 Experimental Design

Table 3 shows the 20 tasks we devised. (The actual questions shown to the subjects were a little longer than ones shown in the table, similar to the narratives of TREC topics (See Figure 3)). For example, Task 5 "Find information about Bill Clinton" is a Information Gathering task, and subjects start with the initial query "bill clinton" and are expected to specialize the initial query such as "bill clinton birthplace" and "bill clinton biography" to gather some kinds of information about Bill Clinton. On the other hand, for example, Task 17 "Find tourist attractions in famous European

[2] Yahoo! search engine (`http://www.yahoo.com/`) currently presents query suggestions in a similar way as our flat list interface.

[3] `http://www.bing.com/developers/`


Find Armani products
You recently heard that Armani makes products other than suits. You want to investigate and possibly buy them. Please find as many Armani products as possible.(Types of products.)

Your Answers:
clothes; suit; jeans;

14400000 search results for **armani**

Armani
Links to galleries of the designer's many collections including the Home and Junior lines. ... In order to view this page you need JavaScript and Flash Player 8+ support!
www.giorgioarmani.com

Designer Clothing at Armani Exchange
Italian designer Giorgio **Armani**'s youthful fashion label. ... **Armani** Exchange is the youthful fashion label created by Italian designer Giorgio **Armani**. Find A Store 800.717.AXAX A|X BLOG A|X Mobile TEXT A|X A|X on MySpace A|X on Twitter A|X on Facebook **Armani**.com A|X Brand
www.armaniexchange.com

Emporio Armani Online Store - Home
Emporio **Armani** Online Store . Spring-Summer collection for men and women. Worldwide delivery.
www.emporioarmani.com

Armani.com
English Italiano 日本語 News Twitter Facebook YouTube Giorgio **Armani** Emporio **Armani** **Armani**.com AJ | **Armani** Jcans **Armani** Collczioni Λ|X **Armani** Exchangc **Armani** / Casa **Armani** | Junior Rafacl Nadal for **Armani** Jeans and Emporio **Armani** ...
www.giorgioarmani.com/ga menu/JP/home.html

Figure 3: System for the user study.

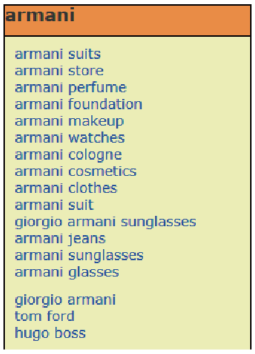


Figure 4: Flat list query suggestion interface.

cities such as Paris" is an Entity Comparison task, with the initial query "paris." In this case, subjects are expected to examine tourist attractions of different cities besides Paris, such as London and Rome.

The initial queries shown in Table 3 were used to automatically produce the initial search engine result page. The subject can then use either the search box or the query suggestion interface to complete the task. Note that we selected two tasks for each combination of the task type (Information Gathering or Entity Comparison) and the entity class (*Company*, *Person*, *Landmark*, *City* and *Product*): for example, Tasks 1 and 2 are Information Gathering tasks for *Company* (Kyocera and Panasonic, respectively).

Our 20 subjects were undergraduate or graduate students in computer science in their twenties. Thus we believe that this is a sample of relatively advanced users who are familiar with Web search and query suggestion. The subjects were asked to collect as many answers relevant to each task as possible within five minutes. Although the time limit was set, we asked the subjects to conduct search tasks as usual so that we could observe natural search behaviors. Upon completion of each task, the subjects filled out a questionnaire about that particular task. Upon completion of the 20 tasks, the subjects filled out a questionnaire about the entire experiment and their preference of interfaces.

We assigned the tasks and the two interfaces to the subjects so that (1) each task was completed with the SParQS and the flat list interface exactly ten times, respectively; and (2) the tasks and subjects were as independent of each other as possible. We randomized the presentation order of tasks, query suggestions, entities and categories (the latter for the SParQS interface only).

6.3 Results and Discussions

In our user study, we observed several significant differences of search behaviors between the SParQS and flat list users. Table 4 summarizes the user study results of our user study. "#Answers" is the number of answers found per question, "#Queries" is the number of queries used (either a query input or a query suggestion clicked), "#QS queries" is the number of query suggestions used, and "#Documents" is the number of documents viewed. "#Entities" is the number of entities to which the subject found one or more answers, and "Successful search rate" is the proportion of searches where the subject found at least one answer. Whereas, Table 5 summarizes the questionnaire results of our user study. "Familiarity" (Are you familiar with this topic?) "Easiness" (Was it easy to complete this task?) "Satisfaction" (Are you satisfied with the answers you collected?) "Enough Time" (Did you have enough time to complete the task?) "Helpfulness" (Was the query suggestion effective to complete the task?) scores are from the online per-task questionnaire: subjects chose from scores 1 (Not at all), 2, 3 (Somewhat), 4, and 5 (Extremely). We performed an ANOVA for two between subjects variables, i.e. task type (Information Gathering vs. Entity Comparison) and interface type (SParQS vs. flat list). Statistical significance at $\alpha = 0.05$ is indicated at the Significance column in Tables 4 and 5. If significant interaction was found in ANOVA, we examined the difference between the two interfaces for each task type with Bonferroni post-hoc test [13]. A dagger in the Significance column indicates significant interaction between the SParQS and flat list interface in either Information Gathering or Entity Comparison tasks.

Table 3: Tasks used in our user study. Type G means Information Gathering task, while C means Entity Comparison task.

ID	Shortened question	Initial query	Type
1	Find Kyocera products	kyocera	G
2	Find Panasonic appliances	panasonic	G
3	Find information about James Cameron	james cameron	G
4	Find information about Bill Clinton	bill clinton	G
5	Find information about the White House	white house	G
6	Find information about the Amazon River	amazon river	G
7	Find travel information about Beijing	beijing	G
8	Find travel information about Budapest	budapest	G
9	Find Rolex watch brands	rolex	G
10	Find Armani products	armani	G
11	Find cars made by Mazda and other manufacturers	mazda	C
12	Find cameras made by Nikon and its competitors	nikon	C
13	Find concert schedules of Norah Jones and other musicians	norah jones	C
14	Find albums by Lady Gaga and other female singers	lady gaga	C
15	Find accommodations for visiting U.S. National Parks such as Yosemite	yosemite national park	C
16	Find events held in Chicago's tourist attractions such as Grant Park	grant park	C
17	Find tourist attractions in famous European cities such as Paris	paris	C
18	Find weather information for Australian cities such as Sydney	sydney	C
19	Find small laptops made by different makers including Acer	acer	C
20	Find prices for Steve Madden and other handbags	steve madden	C

Compared to the flat list users, the SParQS users found more answers in Entity Comparison tasks and fewer answers in Information Gathering tasks [Table 4(a)]. But these are not statistically significant. Especially in Entity Comparison tasks, the SParQS users found answers for significantly more entities than the flat list users [Table 4(e)]. There was a significant interaction between the two interface types in Entity Comparison tasks, $t(198) = 3.02, p < 0.005$. Thus, the SParQS interface was helpful in Entity Comparison tasks in terms of finding answers, but underperformed the flat list interface in Information Gathering tasks. The effectiveness of the SParQS interface might depend on the difficulty of search tasks: according to results shown in Table 4(b) and (d), Entity Comparison tasks needed significantly more queries and documents than

Table 4: User study results of the user study. The mean and standard deviation (SD) are shown as "<Mean> (<SD>)". A dagger in the Significance column indicates significant interaction between the two interfaces in Entity Comparison tasks.

		Information Gathering		Entity Comparison		
		List	SParQS	List	SParQS	Significance
(a)	#Answers	**9.58** (5.81)	8.14 (5.81)	7.58 (7.71)	**9.63** (7.72)	N/A
(b)	#Queries	3.45 (2.49)	**3.99** (2.33)	**5.40** (2.95)	4.83 (1.77)	Task
(c)	#QS queries	2.01 (2.35)	**2.60** (2.24)	2.14 (1.77)	**3.13** (1.95)	Interface
(d)	#Documents	**2.50** (1.96)	2.23 (2.00)	3.45 (2.74)	**4.27** (2.78)	Task
(e)	#Entities	1.14 (0.40)	**1.20** (0.42)	1.65 (0.88)	**2.09** (1.14)	Interface, Task, † (Entity Comparison)
(f)	Successful search rate	0.19 (0.26)	**0.27** (0.30)	0.14 (0.19)	**0.18** (0.24)	Interface, Task

Table 5: Questionnaire results of the user study. The mean and standard deviation (SD) are shown as "<Mean> (<SD>)". A dagger in the Significance column indicates significant interaction between the two interfaces in Information Gathering tasks.

		Information Gathering		Entity Comparison		
		List	SParQS	List	SParQS	Significance
(a)	Familiarity	2.94 (1.24)	**3.13** (1.27)	**2.69** (1.43)	2.63 (1.23)	Task
(b)	Easiness	3.80 (1.08)	**4.16** (0.88)	3.68 (1.13)	**3.72** (1.08)	Task
(c)	Satisfaction	3.73 (1.05)	**4.10** (0.95)	3.61 (1.27)	**3.82** (0.94)	Interface
(d)	Enough time	3.66 (1.08)	**3.88** (0.96)	3.41 (1.28)	**3.62** (0.92)	Task, Interface
(e)	Helpfulness	3.37 (1.23)	**3.91** (0.94)	3.77 (1.12)	**3.86** (1.01)	Interface, † (Information Gathering)

Information Gathering tasks, $F(1, 396) = 32.71, p < 0.01$ and $F(1, 396) = 38.16, p < 0.01$, respectively.

The SParQS users utilized significantly more query suggestions than the flat list users, $F(1, 396) = 14.08, p < 0.01$ [Table 4(c)]. Moreover, the SParQS users were significantly more successful in search than the flat list users: successful search rate of the SParQS users is significantly higher than that of the flat list users, $F(1, 396)$ $= 6.18, p < 0.05$ [Table 4(f)].

For a better understanding of user behavior in the both types of tasks and interfaces, we visualize some representative search behaviors on the two interfaces in Figures 5(a) and (b): they show transitions between query input (Q), query suggestion selection (S), document viewing (D) and answer submission (A) for each subject. A horizontal sequence represents the behavior pattern of each subject, e.g. a subject starts with an initial query (Q), views a document (D), and submits an answer (A). Search behaviors in Information Gathering tasks 3 and 4 are shown in Figures 5(a), where the top two *blocks* present search behaviors of the flat list users, while the bottom two present ones of the SParQS users. In Information Gathering tasks, the flat list users generally input a few queries (blue/green) and found many answers (red) from a few search results and documents, while the SParQS users utilized many query suggestions (green) but failed to efficiently get answers. Thus, the SParQS interface might have unnecessarily encouraged the subjects to utilize many query suggestions, but in fact a few searches might have been sufficient for obtaining many answers. On the other hand, the SParQS users achieved high efficiency in Entity Comparison tasks as seen in Figure 5(b). They utilized many query suggestions (green) and thereby found many answers (red). Especially in Task 15, there is a drastic improvement (many more "A"s) even though the SParQS and flat list interfaces contain exactly the same query suggestions.

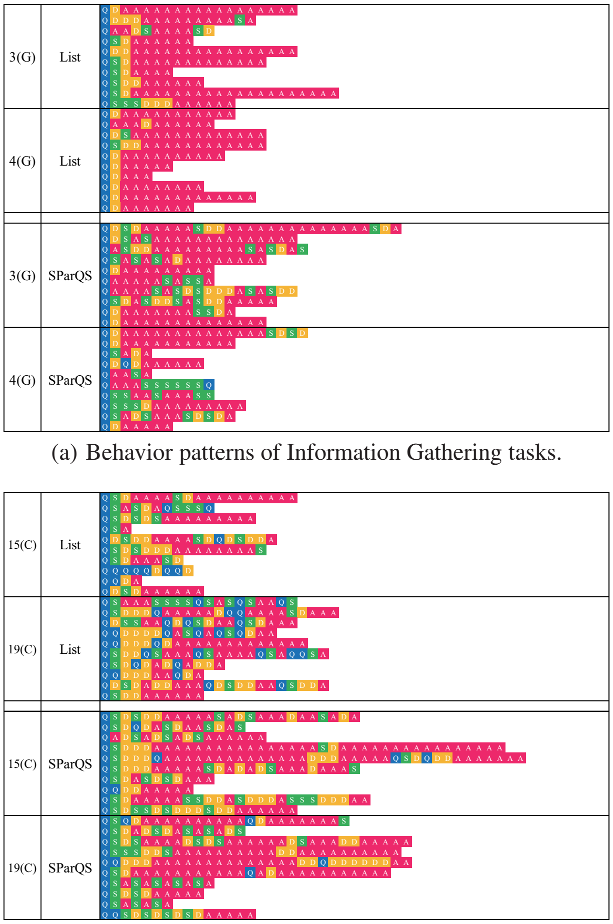
In Table 5, we see some positive questionnaire results. The SParQS users were more satisfied with their answers, $F(1, 396) =$ 7.3476, $p < 0.01$ [Table 5(c)], they felt that they had enough time to complete the task, $F(1, 396) = 3.9557, p < 0.05$ [Table 5(d)], and they felt that query suggestions were more helpful than the flat list case, $F(1, 396) = 8.3381, p < 0.01$ [Table 5(e)]. It seems that the structured query suggestions highly impacted the usability of query suggestions in both types of tasks. Those positive effects were caused probably by the higher successful search rate and accessibility to query suggestions. The SParQS users did in fact utilize more query suggestions than the flat list users as can be seen in Table 4(c).

In post-experiment questionnaires, six subjects remarked that the SParQS interface helped them find effective queries quickly. Moreover, three subjects remarked that SParQS helped them understand the suggested queries better. These comments support claims by Sadikov *et al.* [21] and Guo *et al.* [10]: the categorization of query suggestions can save the user's time to locate a suggestion that serves his need, and can help the user understand the meaning of each query suggestion. Whereas, one subject preferred the flat list to the SParQS saying that the quality of the query suggestion classification was not good enough. Thus, our future work involves improving the SParQS back-end algorithm.

7. CONCLUSIONS

In this paper, we proposed a new method to present query suggestions to the user, which has been designed to help two query reformulation actions: specialization and parallel movement. Our prototype, SParQS classifies query suggestions into automatically generated categories and generates a label for each category. Moreover, SParQS presents some new entities as alternatives to the original query, together with their query suggestions classified in the same way as the original query's suggestions. We conducted a task-based user study to compare SParQS with a traditional flat list query suggestion interface. Our results show that subjects using the flat list query suggestion interface and those using the SParQS interface behaved significantly differently even though the set of query suggestions presented was exactly the same. Moreover, SParQS users could find answers for more entities in Entity Comparison tasks, experienced greater search satisfaction and success, and the SParQS interface was perceived to be more helpful than the flat list.

As future work, we would like to improve the accuracy and entity coverage of the SParQS back-end algorithm. Moreover, as our user study did not directly measure the effect of presenting alternative entities to the user (as our baseline flat list interface also presented alternative entities), we would like to examine this effect in another user study. While our user experiments clearly showed the advantages of SParQS over a flat list, we should clarify exactly what features of SParQS contribute to these differences, e.g. the choice of aspects that apply to multiple entities, and the labels for each aspect. In particular, how users experience diagonal movement (See Section 1) probably deserves an investigation.



(a) Behavior patterns of Information Gathering tasks.

(b) Behavior patterns of Entity Comparison tasks.

Figure 5: Behavior patterns of each task. A horizontal sequence represents the behavior pattern of each subject, which consist of the transition between query input (Q) in blue, query suggestion selection (S) in green, document viewing (D) in yellow, and answer submission (A) in red.

8. ACKNOWLEDGMENTS

This work was supported in part by KAKENHI (No. 10J04687) and a Kyoto University GCOE Program entitled “Informatics Education and Research for Knowledge-Circulating Society.”

9. REFERENCES

[1] A. Amin, M. Hildebrand, J. van Ossenbruggen, V. Evers, and L. Hardman. Organizing suggestions in autocompletion interfaces. In *Proc. of ECIR*, pages 521–529, 2009.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis. An optimization framework for query recommendation. In *Proc. of WSDM*, pages 161–170, 2010.

[3] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *Proc. of SIGIR*, pages 88–95, 2003.

[4] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 588–596, 2004.

[5] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *Proc. of CIKM*, pages 609–618, 2008.

[6] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. From Dango to Japanese Cakes: Query Reformulation Models and Patterns. In *Proc. of WI*, pages 183–190, 2009.

[7] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *Proc. of KDD*, pages 875–883, 2008.

[8] S. Dumais and N. J. Belkin. The TREC interactive tracks: Putting the user into search. In E. Voorhees and D. Harman, editors, *TREC: Experiment and Evaluation in Information Retrieval*, pages 123–152. MIT Press, 2005.

[9] S. Dumais, E. Cutrell, and H. Chen. Optimizing search by showing results in context. In *Proc. of CHI*, pages 277–284, 2001.

[10] J. Guo, X. Cheng, G. Xu, and H. Shen. A Structured Approach to Query Recommendation With Social Annotation Data. In *Proc. of CIKM*, pages 619–628, 2010.

[11] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proc. of SIGIR*, pages 267–274, 2009.

[12] Q. He, D. Jiang, Z. Liao, S. Hoi, K. Chang, E. Lim, and H. Li. Web query recommendation via sequential query prediction. In *Proc. of ICDE*, pages 1443–1454, 2009.

[13] R. Johnson and D. Wichern. *Applied multivariate statistical analysis, 4th ed.* Prentice Hall, 1998.

[14] M. Kamvar and S. Baluja. Query suggestions for mobile search: understanding usage patterns. In *Proc. of CHI*, pages 1013–1016, 2008.

[15] D. Kelly, A. Cushing, M. Dostert, X. Niu, and K. Gyllstrom. Effects of popularity and quality on the usage of query suggestions during information search. In *Proc. of CHI*, pages 45–54, 2010.

[16] D. Kelly, K. Gyllstrom, and E. Bailey. A comparison of query and term suggestion features for interactive searching. In *Proc. of SIGIR*, pages 371–378, 2009.

[17] M. Komachi and H. Suzuki. Minimally supervised learning of semantic knowledge from query logs. In *Proc. of IJCNLP*, pages 358–365, 2008.

[18] H. Ma, H. Yang, I. King, and M. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proc. of CIKM*, pages 709–718, 2008.

[19] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proc. of CIKM*, pages 469–478, 2008.

[20] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978.

[21] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *Proc. of WWW*, pages 841–850, 2010.

[22] S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In *Proc. of the Third International Conference on Language Resources and Evaluation (LREC-2002)*, pages 1818–1824, 2002.

[23] S. Sekine and H. Suzuki. Acquiring ontological knowledge from query logs. In *Proc. of WWW*, pages 1223–1224, 2007.

[24] J. Sim and C. Wright. The kappa statistic in reliability studies: use, interpretation, and sample size requirements. *Physical therapy*, 85(3):257–268, 2005.

[25] Y. Song and L. He. Optimal rare query suggestion with implicit user feedback. In *Proc. of WWW*, pages 901–910, 2010.

[26] Y. Song, D. Zhou, and L.-w. He. Post-ranking query suggestion by diversifying search results. In *Proc. of SIGIR*, pages 815–824, 2011.

[27] J. Wen, J. Nie, and H. Zhang. Clustering user queries of a search engine. In *Proc. of WWW*, pages 162–168, 2001.

[28] R. White and G. Marchionini. Examining the effectiveness of real-time query expansion. *Information Processing & Management*, 43(3):685–704, 2007.

[29] K. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proc. of CHI*, pages 401–408, 2003.

[30] X. Yin and S. Shah. Building taxonomy of web search intents for name entity queries. In *Proc. of WWW*, pages 1001–1010, 2010.