Discovering Tasks from Search Engine Query Logs

CLAUDIO LUCCHESE, ISTI-CNR
SALVATORE ORLANDO, Università Ca' Foscari Venezia
RAFFAELE PEREGO and FABRIZIO SILVESTRI, ISTI-CNR
GABRIELE TOLOMEI, Università Ca' Foscari Venezia

Although Web search engines still answer user queries with lists of *ten blue links* to webpages, people are increasingly issuing queries to accomplish their daily tasks (e.g., *finding a recipe*, *booking a flight*, *reading online news*, etc.). In this work, we propose a two-step methodology for discovering tasks that users try to perform through search engines. First, we identify *user tasks* from individual user sessions stored in search engine query logs. In our vision, a user task is a set of possibly noncontiguous queries (within a user search session), which refer to the same need. Second, we discover *collective tasks* by aggregating similar user tasks, possibly performed by distinct users. To discover user tasks, we propose query similarity functions based on unsupervised and supervised learning approaches. We present a set of query clustering methods that exploit these functions in order to detect user tasks. All the proposed solutions were evaluated on a manually-built ground truth, and two of them performed better than state-of-the-art approaches. To detect collective tasks, we propose four methods that cluster previously discovered user tasks, which in turn are represented by the bag-of-words extracted from their composing queries. These solutions were also evaluated on another manually-built ground truth.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering, Query formulation, Search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms: Algorithms, Design, Experimentation

Additional Key Words and Phrases: Query log analysis, query clustering, user search intent, user search session boundaries, user tasks, user task discovery, collective tasks, collective task discovery

ACM Reference Format:
Lucchese, C., Orlando, S., Perego, R., Silvestri, F., and Tolomei, G. 2013. Discovering tasks from search engine query logs. ACM Trans. Inf. Syst. 31, 3, Article 14 (July 2013), 43 pages.
DOI: http://dx.doi.org/10.1145/2493175.2493179

1. INTRODUCTION

People rely heavily on Web search engines to organize their daily activities. A key reason for the popularity of today's search engines is their user-friendly interface [Baeza-Yates and Ribeiro-Neto 1999], which allows users to easily query for their needs by issuing their own lists of keywords. Users exploit this simple query-based interface to retrieve Web information and resources, which in turn are used to perform one or more Web-mediated *tasks* [Spink et al. 2006], for example, finding a recipe, booking a flight, reading online news, etc.

This work has been partially supported by projects MIDAS (FP7 EU Grant Agreement no. 318786), InGeoCloudS (CIP-PSP EU Grant Agreement no. 297300), and MIUR PRIN ARS TechnoMedia.
Authors' addresses: C. Lucchese, R. Perego, and F. Silvestri, ISTI-CNR, Pisa, Italy; S. Orlando and G. Tolomei, DAIS, Università Ca' Foscari Venezia, Italy; corresponding author's email: gabriele.tolomei@unive.it.

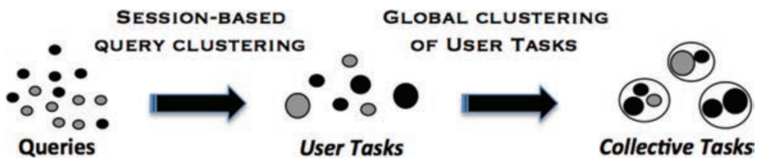


Fig. 1. Our two-step task discovery process: user task and collective task discovery.

Although search engines nowadays offer several mechanisms to help their users (e.g., query suggestion, search-as-you-type, result diversification, etc.), in essence they are document retrieval systems that answer a user query with a simple list of *ten blue links*. If the results returned are not satisfactory, the user may decide to refine his query, thereby getting a new list of results that is hopefully more relevant to his needs. However, this *query-look-refine* paradigm is not effective for all the tasks to be accomplished.

We believe next-generation search engines should progress from being mere Web document retrieval tools to becoming multifaceted systems which fully support users while they are interacting with the Web. This creates novel and exciting research challenges ranging from the ability to recognize latent tasks from the issued queries, to the design of new recommendation strategies and user interfaces for showing relevant results.

This manuscript focuses on this first research challenge and proposes an effective methodology for discovering the tasks that users try to perform through queries they issue to search engines.

Interesting search behaviors and patterns can be revealed by analyzing and mining search engine *query logs*, which record the activities of many users [Broder 2002; Rose and Levinson 2004; Lee et al. 2005; Jansen and Spink 2006; Silvestri 2010]. It has been discovered that query logs are suitable sources from which tasks can be extracted. In concrete terms, a very important aspect we can analyze from a query log is represented by *query sessions*, that is, specific sets/sequences of queries issued by a user while interacting with a search engine.

There are two distinct levels of granularity to consider when detecting the set of tasks from a query log. The first is the *intra-user* level, where tasks are searched for within individual user query sessions. The challenge here is to find relationships that go beyond lexical query similarity, for instance, the case of a query session for the user Alice which contains the queries `new york hotel` and `waldorf astoria`, which are not necessarily issued consecutively. Those two queries clearly refer to the task *reserving a hotel room in New York*, yet they do not share any common terms. The second level of granularity is the *inter-user* level, where even more subtle problems may occur and need to be handled, for instance, let another user Bob type the following two queries: `hotels in new york` and `holiday inn ny`. Both users are clearly trying to achieve the same task (i.e., *reserving a hotel room in New York*), but by means of different queries. This second problem occurs because distinct users tend to phrase even the same task in several different ways.

Therefore, a task discovery methodology has to take into consideration these two preceding aspects in order to be effective. To this end, we divide the problem of discovering tasks into two subproblems, and we address them separately. First, we extract from each individual user session those sets of queries that were issued to achieve specific tasks. We call each of those sets a *user task*, since they strictly depend on each individual user. Second, we consider all the user sessions of the query log, and we identify all the queries related to a common task—possibly performed by distinct users—by grouping together similar user tasks. We refer to each agglomeration of similar user tasks as a *collective task*. Figure 1 shows this two-step task discovery process just described.

The rationale behind this two-step strategy is as follows. In a previous work [Lucchese et al. 2011], we already proved that user tasks can effectively be found by exploiting the lexical and semantic content similarity of queries issued by individuals within specific *search contexts* (i.e., time-bounded subsessions of the original query session). Conversely, the same approach would not be able to discover collective tasks if applied directly to users' queries, because queries that are issued by two users, which are lexically or semantically similar, might refer to different latent needs. In addition, this two-step method guarantees scalability, since the agglomeration needs to process a reduced number of objects, that is, groups of queries rather than single ones.

Finally, two kinds of contributions are presented in this manuscript: one concerns user task discovery while the other is related to collective task discovery, and both are analyzed in the following.

Contributions on User Task Discovery. In Lucchese et al. [2011], we already showed that users perform *multitasking* search activities in the query streams issued to a search engine [Spink et al. 2006]. Multitasking refers to the way users interact with a search engine, by intertwining different tasks within the same time period. This makes it difficult to identify user tasks by just splitting each user session into time-based *sequences* of queries. Thus, a more precise measure of the *task relatedness* between query pairs was needed. To this end, we proposed an *unsupervised learning* approach for measuring task-based query similarity, which relied on a selection of both *internal* and *external* query log features. Internal features are available from the original query log data, whereas external ones can be derived from other data sources. This approach resulted in two *query similarity functions*. The first, σ_1, was a *convex combination* of the selected query log features. The second, σ_2, combined typical lexical content similarity measures with the collaborative knowledge provided by *Wiktionary*[1] and *Wikipedia*.[2] These external knowledge bases were used to enrich the *semantics* of each query, that is, to *wikify* each query in order to make more accurate decisions during the actual user task discovery step. Furthermore, the preceding notion of task relatedness between query pairs was used to discover the final set of user tasks. To this end, we introduced a set of *query clustering* methods with the aim of grouping together task-related queries, namely queries that are assumed to be part of the same user task on the basis of a specified task-relatedness measure. In particular, we compared two techniques derived from well-known clustering algorithms, that is, K-MEANS [MacQueen 1967] and DB-SCAN [Ester et al. 1996] with two other *graph-based* methods. All four proposed solutions were evaluated on a ground truth, that is, a set of manually annotated user tasks. Results showed that the latter two techniques performed better than the former, and they also improved other state-of-the-art approaches noticeably.

As an innovation contribution to this work, we also propose and evaluate a *supervised learning* approach for determining the task-based similarity between query pairs. Unlike the unsupervised learning approach introduced in Lucchese et al. [2011], here the task relatedness is learned by training several classifiers in our ground truth. In particular, we exploit the binary classifiers introduced by Jones and Klinkner [2008] and use the prediction provided by these classifiers in order to determine whether two queries belong to the same task. The probability value associated to that prediction is in turn used as a measure of how strong the task relatedness is between each pair of queries.

We train the classifiers over all the features that Jones and Klinkner [2008] claim to be the most suitable for predicting if two queries belong to the same search goal.

[1]`http://www.wiktionary.org`
[2]`http://www.wikipedia.org`

Moreover, we expand these training features with both Wikipedia, that is, the *wikification* of the query, as well as the URL overlapping degree between the results returned by a search engine for each query, that is, the *Jaccard index* between the top-20 URLs returned for each query. This supervised learning approach leads to a set of new task-based query similarity functions, which are in turn exploited by the two best-performing clustering methods for user task discovery introduced in our previous work, namely QC-WCC and QC-HTC.

Experimental results have shown that combining supervised task-relatedness learning with our query clustering methods does not substantially improve the overall effectiveness in discovering user tasks. However, the performance of the classifiers proposed by Jones and Klinkner [2008] improves significantly with the combined use of wikification and URL overlapping along with the other features.

Furthermore, we test our two best-performing user task discovery methods on a larger dataset (besides evaluating them on the smaller ground truth, as in Lucchese et al. [2011]). Interestingly, we discover that the analysis conducted on both the smaller and the larger datasets are coherent, in other words, they result in similar conclusions. This means that we can be relatively confident that replicating experiments on both datasets would also lead to similar quantitative results.

Contributions on Collective Task Discovery. The true innovation in this work, which completes the overall roadmap we sketched for finding tasks from search engine query logs, is based on the notion of collective task (see Figure 1). In a similar way to discovering user tasks, we provide a second ground truth by manually grouping a set of user tasks into a set of collective tasks. In addition, we present and discuss four methods used to actually discover collective tasks. All of them are user task clustering techniques, where each user task is represented by the *bag of words* of its composing queries. Each solution adopts a different clustering strategy (i.e., *partitional* vs. *agglomerative*) and a different user task similarity measure (i.e., *cosine similarity* vs. *Pearson's coefficient*). We quantitatively evaluate the four methods on the ground truth of collective tasks. The best results were obtained by a partitional clustering technique, which uses the cosine similarity measure. Finally, this best-performing technique is also run on a larger dataset of user tasks, and its performance is assessed by means of examples of evidence.

Structure of the Article. The rest of the article is organized as follows. Section 2 describes related work on query log analysis and mostly focuses on query session boundaries detection. Section 3 provides the description and analysis of our benchmark dataset, that is, the 2006 AOL query log. In Section 4, we propose our theoretical model and the statement of the *user task discovery problem* (UTDP). Section 5 presents the construction of our ground truth by manually grouping queries that are considered to be task related in a portion of our sample dataset. In addition, we propose some statistics relating to this corpus of manually identified user tasks. Section 6 introduces several approaches for measuring the task relatedness between query pairs, that is, task-based query similarity functions, which in turn are exploited by the user task discovery methods proposed and compared in Section 7. Thus, Section 8 shows the experiments we conducted on user task discovery as well as the results we obtained. Section 9 bridges the gap between user task and collective task discovery by introducing the idea of collective tasks, and it introduces a set of four algorithms for finding collective tasks from the set of previously discovered user tasks. We test the quality of all the proposed solutions by comparing them with a common manually-built ground truth. To test the strength of the best-performing solution for collective task discovery, we apply it to a larger dataset and illustrate some resulting evidence. Finally, Section 10 presents our conclusions and indicates some possible future research.

2. RELATED WORK

The analysis of query logs collected by Web search engines has increasingly gained interest within the Web mining research community. Query logs record information about the search activities of users and are thus precious data sources for understanding how people search the Web [Silvestri 2010]. Moreover, a number of different applications, that is, caching, index partitioning, document prioritization, query suggestion, can benefit from analysis performed on search engine query logs [Silvestri et al. 2008].

Typical statistics that can be drawn from query logs simply consider the query set in order to measure query popularity, term popularity, average query length, distance between repetitions of queries or terms, etc. A more in-depth analysis consists in studying *search sessions*, that is, temporal sequences of queries issued by users.

The first study on a query log from a commercial search engine was conducted by Jansen et al. [1998]. In their research, the authors analyzed a one-day log collected by the Excite search engine, which contains 51,473 queries issued by 18,113 users. In addition, Silverstein et al. [1999] present an exhaustive analysis of a very large query log collected by the AltaVista search engine, which consists of about a billion queries submitted in a period of 42 days by approximately 285 million users. The authors show interesting results, including the analysis of user query sessions and of correlations between query terms.

However, most works relating to mining query logs aim to understand the real intent behind queries issued by users. Broder [2002] claims that the "need behind the query" in a Web context is not clearly informational, like in a standard information retrieval domain. Hence, he introduces a taxonomy of Web searches by classifying the queries into three classes according to their intent: (i) *navigational*, whose intent is to reach a specific Web site; (ii) *informational*, which aims to acquire some information from one or more Web documents; and (iii) *transactional*, whose intent is to perform some Web-mediated activity. Rose and Levinson [2004] propose their own user search goals classification by adding more hierarchical levels to this taxonomy. Lee et al. [2005] describe whether and how search goal identification processes behind a user query might be automatically performed on the basis of two features, namely *past user-click behavior* and *anchor-link distribution*.

Many works deal with the identification of user search sessions boundaries. Previous papers on this topic, which is very relevant for our work, can be classified into the following groups on the basis of the technique used: (i) *time-based*, (ii) *content-based*, and (iii) *ad-hoc* techniques, which usually combine (i) and (ii).

Time-Based. Time-based techniques have been extensively proposed in past research works for detecting meaningful search sessions because of their simplicity and ease of implementation. Indeed, these approaches are based on the assumption that the time interval between queries issued by the same user is the predominant factor in determining a topic shift in search activities. Roughly speaking, two consecutive user queries are likely to be related if the time gap between them is lower than a fixed threshold.

With this view in mind, Silverstein et al. [1999] first defined the concept of *session* as follows: two consecutive queries are part of the same session if they are issued within a five-minute time window. On the basis of this definition, they split the benchmark dataset into sessions containing 2.02 queries on average. He and Göker [2000] use different thresholds ranging from 1 to 50 minutes to devise user sessions from a Excite query log.

Radlinski and Joachims [2005] observe that users often perform a sequence of queries based on a similar information need, and they refer to those sequences of reformulated queries as *query chains*. Their work presents a method for automatically detecting query chains in query and click-through logs using a 30-minute threshold to determine if two consecutive queries belong to the same search session.

Jansen and Spink [2006] make a comparison of nine search engine transaction logs from the perspectives of session length, query length, query complexity, and content viewed. In their paper, they provide another definition of session, that is, *search episode*, and describe it as the period of time between the first and the last recorded time stamp on the search engine server from a particular user on a single day, so the session length may vary from less than a minute to a few hours.

By using the same concept of search episode, Spink et al. [2006] investigate multitasking behaviors for user interaction with the search engine. Multitasking during Web searches involves a seek-and-switch process between several topics within a single user session. Again, the authors define a user session as the entire series of queries submitted by a user during one interaction with the search engine, so that the session length may vary from less than one minute to a few hours. The results of this analysis performed on an AltaVista query log showed that multitasking is a common characteristic of Web searching. In our work, we reveal the presence of multitasking also within shorter user activities.

Shi and Yang [2006] describe the so-called *dynamic sliding window segmentation* method, which is based on three temporal constraints: α as the maximum time interval between two consecutive queries in the same session, β as the maximum inactivity time within the same session, and γ as the maximum length of a single session. The authors empirically set α, β, and γ to be 5 minutes, 24 hours, and 60 minutes, respectively.

Finally, Richardson [2008] shows the value of long-term search engine query logs with respect to short-term, that is, within-session query information. He claims that long-term query logs can be used to better understand the world where we live and shows that query effects are long lasting. Basically, in his work, Richardson does not look at term co-occurrences only within a search session (which he agrees to be a 30-minute time window) but rather across entire query histories.

Content-Based. Some previous works propose exploiting the lexical content of queries in order to determine *session boundaries* corresponding to possible topic shifts in the stream of queries issued by users [Lau and Horvitz 1999; He et al. 2002; Ozmutlu and Çavdur 2005]. To this end, several *search patterns* are proposed by means of lexical comparison based on different string similarity metrics (e.g., Levenshtein, Jaccard, etc.).

Nevertheless, approaches relying only on content features suffer the so-called *vocabulary-mismatch problem*, namely the existence of topically-related queries without any shared terms (e.g., the queries `nba` and `kobe bryant` are completely different from a lexical content perspective, but they are undoubtedly related). In order to overcome this issue, Shen et al. [2005] compare expanded representations of queries instead of the actual queries themselves. Each individual expanded query is obtained by concatenating the titles and the Web snippets for the top 50 results provided by a search engine for the specific query. Therefore, the relatedness between query pairs is computed by using the *cosine similarity* between the corresponding expanded queries.

Ad-Hoc. Jansen et al. [2007] assume that a new search pattern always identifies the start of a new session. Moreover, He et al. [2002] show that statistical information collected from query logs can be used to estimate the probability that a search pattern actually implies a session boundary. In particular, they extend a previous work [He and Göker 2000] to consider both temporal and lexical information.

Similarly, Ozmutlu and Çavdur [2005] describe a mechanism for identifying topic changes in user search behavior by combining time and query content features. They test the validity of their approach using a genetic algorithm in order to learn the parameters of the topic identification task. The algorithm takes into account the topic shift and continuation probabilities of the dataset leveraging on query patterns (i.e., lexical content) and time intervals.

Seco and Cardoso [2006] propose that a candidate query belongs to a new session if it does not have any terms in common with the queries of the current session or the time interval between the candidate query and the last query in the current session is greater than 60 minutes.

Gayo-Avello [2009] presents an exhaustive survey on session boundary detection methods. Furthermore, the author introduces a new technique which works on the basis of a geometric interpretation of both time gap and content similarity between consecutive query pairs.

Other approaches tackle the session boundary detection problem by leveraging more complex models and by combining more advanced features.

Boldi et al. [2008] introduce the *query-flow graph* (QFG) as a model for representing data collected in search engine query logs. Intuitively, in the QFG, a directed edge from query q_i to query q_j means that the two queries are likely to be part of the same search goal. Any path over the QFG may be seen as a searching behavior, the likelihood of which is given by the strength of the edges along the path. The authors exploit this model to segment the query stream into sets of related information-seeking queries, thus reducing the problem to an instance of the *asymmetric traveling salesman problem* (ATSP).

Jones and Klinkner [2008] address a problem that appears to be similar to ours. In particular, they argue that within a user's query stream, it is possible to recognize specific hierarchical units, that is, *search missions*, which are in turn divided into disjoint *search goals*. A search goal is defined as an atomic information need resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Given a manually-generated ground-truth, the authors investigate how to learn a suitable binary classifier, which is aimed at detecting whether two queries belong to the same task. One of the various results is that they realize that timeouts, whatever their lengths, are of limited use in predicting whether two queries belong to the same goal and unsuitable for identifying session boundaries. However, the authors do not explore how such binary classifier could be exploited for actually segmenting users' query streams into goals and missions.

Mei et al. [2009] present a general framework for studying sequences of search activities performed by users. According to the hierarchical user search model introduced by Jones and Klinkner [2008], this framework captures the user behavior at multiple levels of granularity: whole search sessions, search missions and their composite subtasks (i.e., search goals), blocks of related queries, individual queries, click-through data, and eye-tracking fixations.

Donato et al. [2010] adopt the same hierarchical structure of user search behavior and show how search missions can be automatically identified on-the-fly as the user interacts with the Web search engine. In particular, the authors plug this automatic search mission discovery mechanism into SearchPad, a novel Yahoo! application aimed at helping users keep track of results they have consulted. The novelty of this approach however is that it is automatically triggered only when the system decides, with a fair level of confidence, that the user is actually undertaking a search mission.

However, when no labeled training set is available, suitable mechanisms for identifying session boundaries, and thus for extracting search goals, may rely on *unsupervised learning* approaches. Typically, these mechanisms are based on query clustering algorithms which group queries that are related to the same need. The rationale for using query clustering is based on the assumption that if two queries end up in the same cluster, then they are topically related.

Beeferman and Berger [2000] introduce a technique for mining a collection of user transactions with a search engine, which discovers clusters of similar queries and similar URLs. The proposed algorithm does not consider query content but analyzes URL co-occurences within the click-through data stored in the query log, which is modeled as a bipartite graph.

Wen et al. [2002] describe a query clustering method that makes use of user logs to identify the Web documents users have selected for a certain query. The similarity between two queries may be deduced from counting the documents the users clicked on and that are shared between the queries. Fu et al. [2003] propose a hybrid method to cluster queries by utilizing both query terms and query results, showing that this combination performs better than using either method alone.

Cao et al. [2008] propose a novel clustering algorithm for summarizing queries into concepts throughout a click-through bipartite graph built from a search log.

Leung et al. [2008] develop online techniques to extract concepts from the snippets of query results and use these concepts to identify related queries. Moreover, the authors propose a new two-phase personalized agglomerative clustering algorithm that is able to generate personalized query clusters.

A couple of more recent related works have appeared. Kotov et al. [2011] discuss methods for modeling and analyzing user search behavior that extends over multiple search sessions. The authors focus on two problems: (i) given a user query, identify all related queries from previous sessions that the user has issued, and (ii) given a multi-query task for a user, predict whether the user will return to this task in the future. Both problems are modeled within a classification framework that uses features of individual queries and long-term user search behavior at different granularity. The outcomes of this work have improved search for complex information needs and have helped in designing search engine features to support cross-session search tasks.

Finally, Guo et al. [2011] introduce the concept of *intent-aware* query similarity, namely a novel approach for computing query pair similarity, which takes into account the potential search intents behind user queries and then measures query similarity for different intents using intent-aware representations. The authors show the usefulness of their approach by applying it to query recommendation, thereby suggesting diverse queries in a structured way to search users.

3. QUERY LOG ANALYSIS

The research challenges we want to address in this work rely on extracting useful patterns from search engine query logs. However, query log analysis is often hard to perform due to the lack of publicly-available datasets. As a result, we used the 2006 AOL query log as our referential dataset. This query log is a very large and long-term collection consisting of about 20 million Web queries issued by more than 657,000 users over three months (from 03/01/2006 to 05/31/2006).[3]

3.1. Session Size Distribution

We analyzed the entire AOL query log and extracted several statistics, such as the total number of queries, the number of queries in each session, the average duration of sessions, etc.

The distribution of long-term sessions size over the entire collection is depicted in Figure 2. This is characterized by a *Zipf's law*, that is, one of a family of related discrete *power law* probability distributions [Reed 2001]. Indeed, 67.5% of user sessions contains less than 30 queries, meaning that more than 2/3 of the users issued about ten queries per month on average. Besides this, longer user sessions, that is, sessions with more

[3]`http://sifaka.cs.uiuc.edu/xshen/aol_querylog.html`

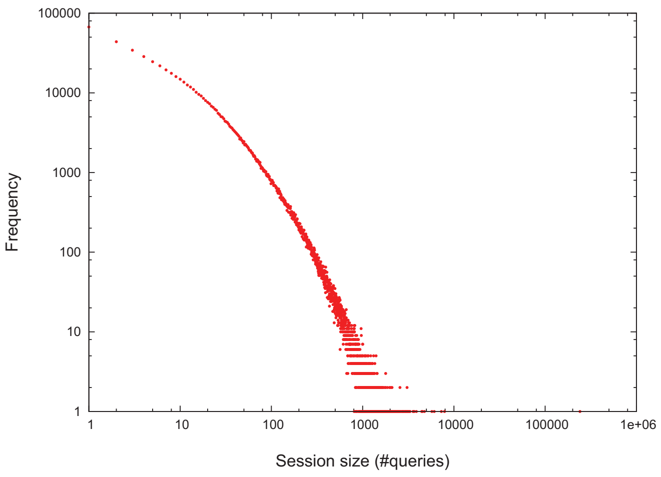


Fig. 2. The distribution of long-term sessions size (log-log scale).

than 1,000 queries over three months, represent only about 0.14% of the whole records. Interestingly, this is compliant with the analysis of long-tailed distributions of query frequencies and query-term frequencies showed by Baeza-Yates et al. [2008]. Here, the authors observed that a small portion of the terms appearing in a large query log were used very often, while the remaining terms were individually used less often. In the same way, a small portion of the user sessions contains a large number of queries, while the remaining sessions are composed of few queries. Finally, it is worth pointing out that in our analysis we do not consider empty sessions, which account for less than 1% of the total.

3.2. Query Time-Gap Distribution

Since users tend to issue bursts of queries for relatively short periods of time, usually followed by longer periods of inactivity, the time gap between queries plays a significant role in detecting session boundaries.

According to Silverstein et al. [1999], session boundaries are detected by considering the user's inactivity periods, that is, the time gaps between consecutive queries in each long-term user session. To establish whether a time gap between two queries actually refers to a session boundary, a suitable threshold Δ is needed. This may be obtained by analyzing the distribution of time gaps between all the consecutive query pairs in our dataset.

We divide all the time gaps into several *buckets* of 60-seconds each. We then analyze the query interarrival times distribution which, again, is revealed to be a power-law (see Figure 3). This model closely fits user behavior during Web search activities when consecutive queries issued within a short period of time are often not independent because they are also topically-related.

More formally, given the following general form of a power-law distribution $p(x)$,

$$p(x) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x}{x_{\min}} \right)^{-\alpha},$$

where $\alpha > 1$ and $x_{\min}$ is the minimum value of x from which the law holds, we are interested in finding the value $\bar{x}$, such that two consecutive queries whose time gap is smaller than $\bar{x}$ are considered to belong to the same time-gap session. When the underlying distribution is unknown, it makes sense to assume a Gaussian distribution

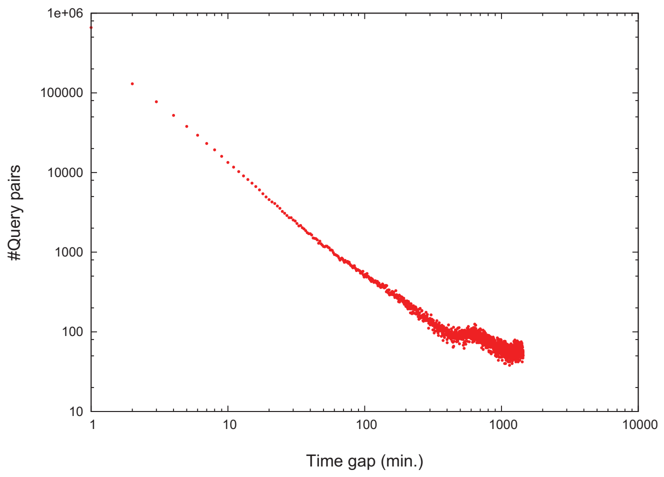


Fig. 3. The distribution of time gaps between each consecutive query pair (log-log scale).

and use a threshold $\bar{x} = \mu + \sigma$ being equal to mean μ plus standard deviation σ, which results in "accepting" $\lambda = 84.1\%$ of the samples. This is equivalent to considering the cumulative distribution $P(\bar{x}) = Pr(X \leq \bar{x})$ and to determining $\bar{x}$, such that $P(\bar{x}) = \lambda$.

Since we know the underlying distribution, and since $P(\bar{x}) = Pr(X \leq \bar{x}) = 1 - Pr(X > \bar{x})$, we map the threshold λ into our context as follows:

$$Pr(X > \bar{x}) = C \int_{\bar{x}}^{\infty} p(X)\, dX = \frac{\alpha - 1}{x_{min}^{-\alpha+1}} \int_{\bar{x}}^{\infty} X^{-\alpha}\, dX = \left(\frac{\bar{x}}{x_{\min}}\right)^{-\alpha+1}.$$

Hence, for our purpose, we have to solve for $\bar{x}$ in the following equation:

$$P(\bar{x}) = 1 - Pr(X > \bar{x}) = 1 - \left(\frac{\bar{x}}{x_{\min}}\right)^{-\alpha+1} = \lambda = 0.841. \tag{1}$$

The value $x_{\min}$ represents the minimum query pair time gap and corresponds to the first interval, that is, 60 seconds. Therefore, we estimate $\alpha = 1.564$, and finally we can solve Eq. (1), finding $\bar{x} \simeq 26$ minutes. This means assuming 84.1% of consecutive query pairs are issued within 26 minutes. As described in detail in Section 8.2.1, in our experiments, we use this value as the threshold Δ for splitting each long-term user session of the the query log.

4. USER TASK DISCOVERY PROBLEM

In this section, we address our research challenge, namely to find concrete user tasks recorded in a search engine query log whose final aim is to satisfy a specific latent need. We use the generic term *task* to refer to this type of atomic latent need, whereas we give the name *user task* to any concrete instance of a task performed by a particular user. Indeed, users may repeatedly enact the same task over the time and often by means of several distinct queries.

In the following, we describe the theoretical model as well as the notation we adopt in order to formally define our research problem.

4.1. Theoretical Model

Let $\mathcal{QL}$ be a query log, which records the queries—along with other information, such as user IDs, time stamps, clicked URLs, etc.—issued by a set $\mathcal{U}$ of N distinct users.

We denote by $\mathcal{S}_u = \langle q_1, q_2, \ldots, q_{|\mathcal{S}_u|} \rangle$ the chronologically ordered sequence of all the queries in $\mathcal{QL}$ issued by a user $u \in \mathcal{U}$. The sequence $\mathcal{S}_u$ of the queries submitted by u

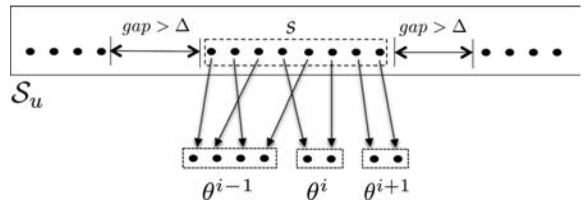


Fig. 4. A generic time-gap session decomposed in a set of interleaved user tasks θ^{i-1}, θ^{i}, and θ^{i+1}.

is the result of multiple long-term interactions with the search engine. Therefore, we consider each sequence $\mathcal{S}_u$ to be composed of a sequence of *time-gap sessions s*, which result from applying a time-splitting technique, that is, $\mathcal{S}_u = \bigcup_{s_i \in \mathcal{S}_u} s_i$. Basically, each time-gap session contains the set of contiguous queries submitted by a user such that each pair of consecutive queries is submitted within a time-gap threshold Δ.

Note that this time-splitting technique does not impose any restrictions on the total time elapsed between the first and the last query of a time-gap session. However, it provides an inactivity time boundary to reasonably determine a shift in user task. Usually, such an inactivity threshold is fixed arbitrarily. Conversely, we set $\Delta = 26$ minutes on the basis of the analysis described in Section 3.2.

Since a time-gap session may include queries related to different needs due to multitasking [Spink et al. 2006; Lucchese et al. 2011], we further identify a partitioning of each time-gap session into subsets of related queries constituting distinct user tasks.

Definition 4.1 (*User Task*). Given a time-gap session $s \subseteq \mathcal{S}_u$ of user $u \in \mathcal{U}$, a *user task* θ, $\theta \subseteq s$, is the maximal subsequence of possibly nonconsecutive queries in s referring to the same latent need. The set of all user tasks in s is a partitioning of s.

The set of user tasks performed by $u \in \mathcal{U}$ is denoted by $\Theta_u = \bigcup_{s_i \subseteq \mathcal{S}_u} \bigcup_{\theta_j \subseteq s_i} \theta_j$, while the set of all the user tasks from all the N users is denoted by $\Theta = \bigcup_{u \in \mathcal{U}} \Theta_u$.

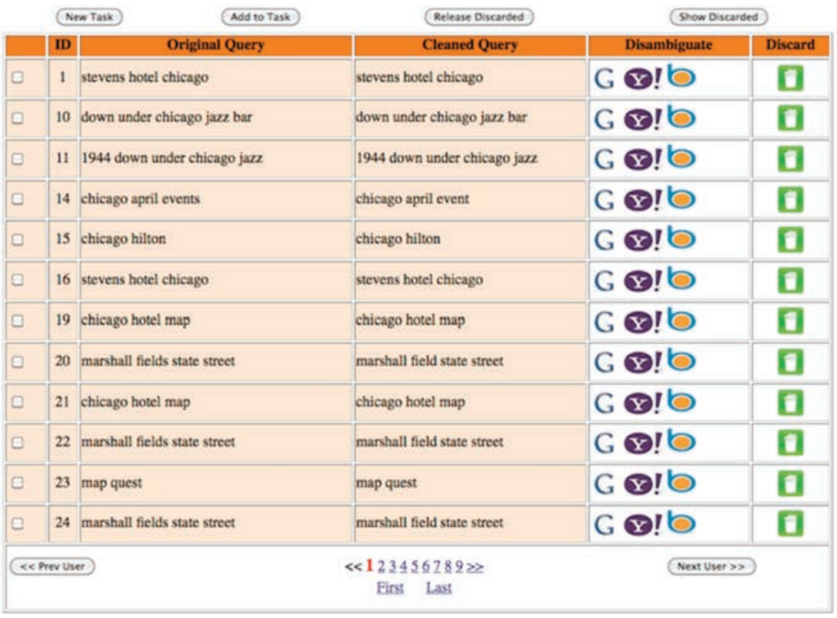
In general, the queries belonging to the various user tasks θ in each time-gap session s may interleave due to multitasking. Thus we order the user tasks by looking at the time stamps of the first query issued within each θ. Using this ordering, let θ^i denote the ith user task performed by the user within a time-gap session. This allows us to represent each Θ_u as an ordered set, namely a *sequence*, of user tasks, that is, $\Theta_u = \langle \theta^1, \theta^2, \ldots, \theta^{|\Theta_u|} \rangle$. Figure 4 illustrates the partitioning of a generic time-gap session $s \subseteq \mathcal{S}_u$ to identify the various user tasks θ^i and also summarizes the notation we have just introduced.

Therefore, the problem of finding $\Theta = \bigcup_{u \in \mathcal{U}} \Theta_u$ in a given query log $\mathcal{QL}$ can be formulated as the *user task discovery problem* (UTDP), whose goal is to find the best *query partitioning strategy* π that approximates the actual set of user tasks Θ_u, when used to segment the time-gap sessions recorded in the query log.

User Task Discovery Problem (UTDP): Given a query log $\mathcal{QL}$ and a user $u \in \mathcal{U}$, let $\mathcal{T}_u$ be the set of user tasks discovered by the query partitioning strategy π applied to the time-gap sessions of $\mathcal{S}_u$, that is, $\mathcal{T}_u = \bigcup_{s_i \subseteq \mathcal{S}_u} \pi(s_i)$. Also, let $\Theta = \bigcup_{u \in \mathcal{U}} \Theta_u$ and $\mathcal{T} = \bigcup_{u \in \mathcal{U}} \mathcal{T}_u$.
The UTDP requires the best partitioning $\bar{\pi}$, such that

$$\bar{\pi} = \underset{\pi}{\text{argmax}}\ \xi(\Theta, \mathcal{T}, \pi), \tag{2}$$

where function $\xi(\cdot)$ is an accuracy measure which evaluates how well the query partitioning strategy π approximates the actual user tasks Θ.



	ID	Original Query	Cleaned Query	Disambiguate	Discard
☐	1	stevens hotel chicago	stevens hotel chicago	G Y! b	🗑
☐	10	down under chicago jazz bar	down under chicago jazz bar	G Y! b	🗑
☐	11	1944 down under chicago jazz	1944 down under chicago jazz	G Y! b	🗑
☐	14	chicago april events	chicago april event	G Y! b	🗑
☐	15	chicago hilton	chicago hilton	G Y! b	🗑
☐	16	stevens hotel chicago	stevens hotel chicago	G Y! b	🗑
☐	19	chicago hotel map	chicago hotel map	G Y! b	🗑
☐	20	marshall fields state street	marshall field state street	G Y! b	🗑
☐	21	chicago hotel map	chicago hotel map	G Y! b	🗑
☐	22	marshall fields state street	marshall field state street	G Y! b	🗑
☐	23	map quest	map quest	G Y! b	🗑
☐	24	marshall fields state street	marshall field state street	G Y! b	🗑

Fig. 5. Snapshot of the Web application used for generating the ground truth.

Several quality measures can be used to evaluate the accuracy of a user task extraction method, and consequently, several ξ functions can be devised. In Section 8, we instantiate ξ in terms of F1, the Rand index, and the Jaccard index.

5. GROUND TRUTH: DEFINITION AND ANALYSIS

According to the *user task discovery problem* statement, we need to find the query partitioning strategy $\bar{\pi}$ that gives the set of user tasks $\mathcal{T}$ which best approximates the actual user tasks Θ. Such optimal user task partitioning can be manually built from a real-world search engine query log.

To this end, we developed a Web application that helped human assessors manually identify the optimal set of user tasks from the AOL query log. As a result, we produced a ground truth that can be used for evaluating any automatic user task discovery method, and which is also freely available to download.[4] In Figure 5 we show a sample snapshot of this Web application.

For each time-gap session, human annotators were presented with the sequence of queries as they were originally issued. They therefore grouped together those queries that they considered to be task related. In addition, they had opportunity to discard meaningless queries from those sessions as well as to submit ambiguous queries to the most important search engines (i.e., Google, Yahoo!, and Bing). For each manually identified user task (i.e., set of task-related queries), the evaluators added a *tag* and, if appropriate, a longer description. The resulting dataset can be seen as a semantic knowledge base of users' search goals (i.e., a taxonomy of user tasks).

Furthermore, dealing with such massive query log datasets typically needs a preprocessing phase in order to clean the collection and to make it more suitable for performing further analysis. This meant that we removed query log records containing both empty queries and query strings composed of only punctuation symbols. In

[4]`http://miles.isti.cnr.it/~tolomei/downloads/aol-task-ground-truth.tar.gz`

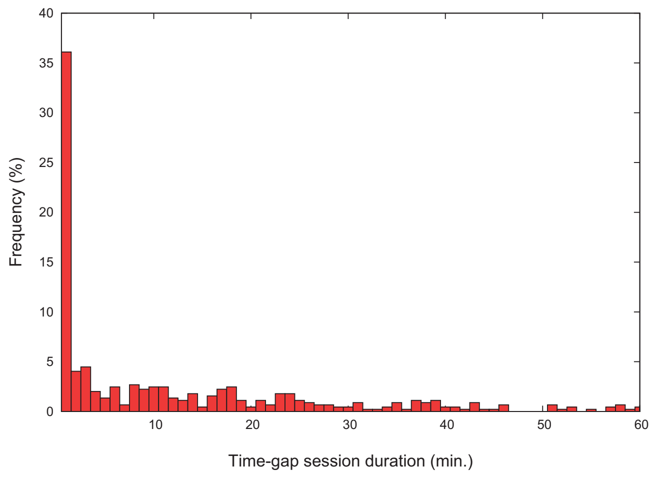


Fig. 6. The distribution of time-gap session duration.

addition, we removed all the *stop words* from each query string. Then, we ran the Porter stemming algorithm [Porter 1980] in order to remove the most common morphological and inflectional English endings from the terms of each query string. Finally, we discarded from the initial dataset those long-term user sessions containing too many queries, which were probably generated by robots. In particular, we are referring to those sessions which had a total number of queries that would have been physically difficult for a human user to issue even, over a period of three months. Indeed, the longest user session contains 240,180 queries, which amounts to approximately 2,669 queries per day. On average, this would be like issuing approximately two queries each minute 24 hours a day for 90 days, which is a highly unlikely query submission rate for a human user.

Our sample dataset was based on the 500 user sessions with the highest number of queries, herein called the *top-500* dataset. This dataset contains a total amount of 518,790 queries, meaning that each user issued on average approximately 1,038 queries in three months, that is, roughly 12 queries per day. The maximum number of queries in a user session is 7,892 and the minimum is 729. This means users submitted from approximately 8 to 88 queries per day. However, only a small fraction of the whole dataset (i.e., the first week of user activities) was showed to annotators in order to simplify the overall manual labeling step. From now on, we will refer to that subset as *top-500-1week*. As regards the human evaluators, they were selected from our laboratory but were not directly involved in this work.

The manual annotation procedure referred to a total of 2,004 queries, from which 446 time-gap sessions were extracted automatically. one hundred thirty-nine time-gap sessions were discarded as meaningless by the annotators and were therefore removed from the ground truth. In the end, 1,424 queries were actually clustered from 307 time-gap sessions.

Figure 6 shows the distribution of time-gap session length, using a discretization factor of 60 seconds. While there are a large number of sessions which are shorter than one minute and which usually contain only one or two queries, the duration of a time-gap session is nevertheless 15 minutes on average. Indeed, as can be observed, sessions lasting 40 minutes (or more) occurred on a fairly frequent basis. Even in these cases, the session lengths suggest that the interaction of users with search engines is nontrivial and that it is likely to include multitasking behaviors. The longest time-gap

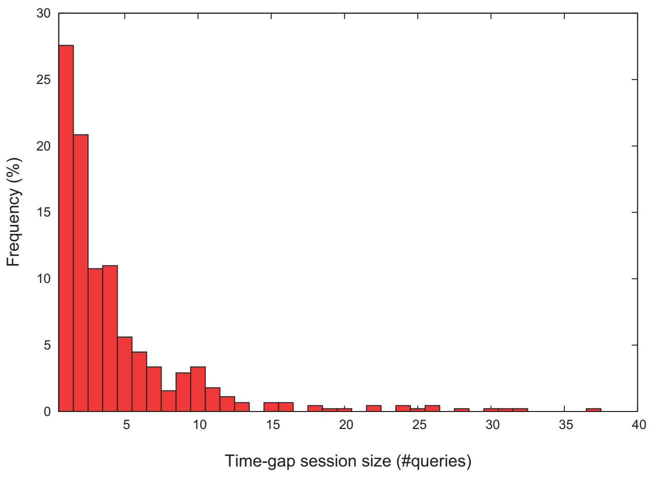


Fig. 7. The distribution of time-gap session size.

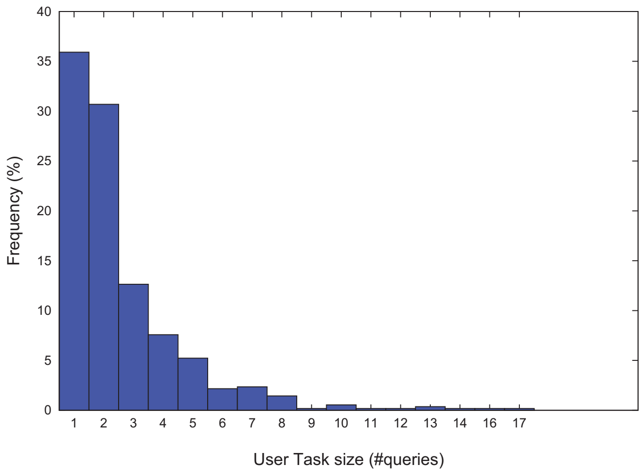


Fig. 8. The distribution of user task size.

session lasted 9,207 seconds, that is, about two and a half hours, and this happened only once in our dataset.[5]

In Figure 7, we report the time-gap session size distribution. On average, each time-gap session contained 4.49 queries, and sessions with at most five queries covered slightly more than half of the query log. The other half of the query log contained longer sessions, where there was a high likelihood that multiple tasks were carried out.

The total number of human-annotated user tasks were 554, with an average of 2.57 queries per user task. The user task size distribution is illustrated in Figure 8. Furthermore, the average number of user tasks accomplished in a time-gap session was 1.80 (see Figure 9).

Among all 307 time-gap sessions considered, 145 contained multiple user tasks. We found that this 50% split between single-tasking and multitasking sessions was

[5]It is highly likely that this user was a robot or a script.

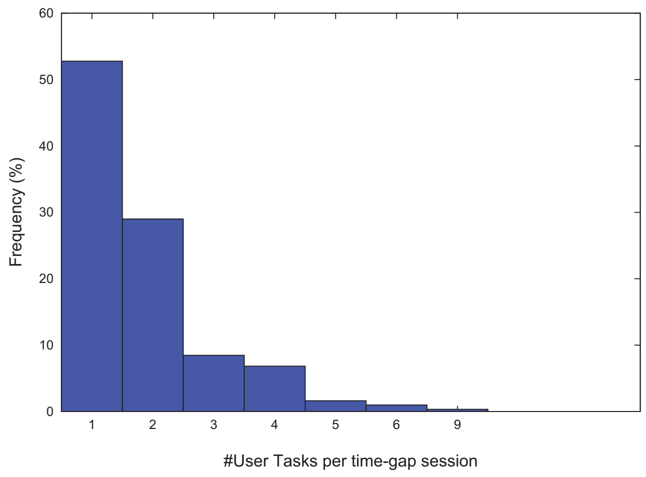


Fig. 9. The distribution of user tasks per time-gap session.

consistent across the various users. Interestingly enough, this shows that a good task detection algorithm has to be able to handle efficiently both single- and multitasking sessions. If we considered all the queries included in each user task, then 1,046 out of 1,424 queries were included in multitasking sessions, meaning that about 74% of the user activity was multitasking.

Finally, we also evaluated the degree of multitasking by taking into account the number of overlapping user tasks. We say that a *jump* occurs whenever two queries in a manually labeled user task are not consecutive. For instance, let $s_u = \langle q_1, q_2, \ldots, q_9 \rangle$ be a time-gap session in $\mathcal{S}_u$, and let $\theta_u^1, \theta_u^2, \theta_u^3$ be the result of the manual annotation procedure for s_u, where $\theta_u^1 = \{q_1, q_2, q_3, q_4\}$, $\theta_u^2 = \{q_5, q_7\}$, and $\theta_u^3 = \{q_6, q_8, q_9\}$. In this case, the number of jumps observed in s_u is two, because there are two query pairs $(q_5, q_7) \in \theta_u^2$ and $(q_6, q_8) \in \theta_u^3$ which are not consecutive. The number of jumps gives a measure of the *simultaneous* multitasking activities. We denote by $j(s_u)$ the simultaneous multitasking degree of s_u as the ratio of user tasks in s_u which have at least one jump. In the previous example, $j(s_u) \simeq 0.67$, since two out of three user tasks contain at least one jump. In Figure 10, we show the distribution of the multitasking degree over all the time-gap sessions. Note that the result for $j(s_u) = 0$ is omitted, because we already know that 50% of the sessions were single-tasking.

6. TASK-BASED QUERY SIMILARITY

Task relatedness is the probability that two queries belong to the same user task, where the user task is not known in advance. A good task-relatedness measure is the building block for discovering user tasks from a given query log $\mathcal{QL}$.

In this section, we describe three different ways of computing task relatedness, the result of which is a set of *task-based query similarity functions*. The first approach, named *time-based*, considers only the time gap between two adjacent queries. We subsequently evaluate both *unsupervised* and *supervised learning* approaches by exploiting a number of query log features as well as external knowledge bases, such as Wikipedia and Yahoo! Search Boss API.

6.1. Time-Based Approach

The simplest approach for measuring task-based query similarity was presented by Silverstein et al. [1999]. The proposed measure is based on the assumption that if two

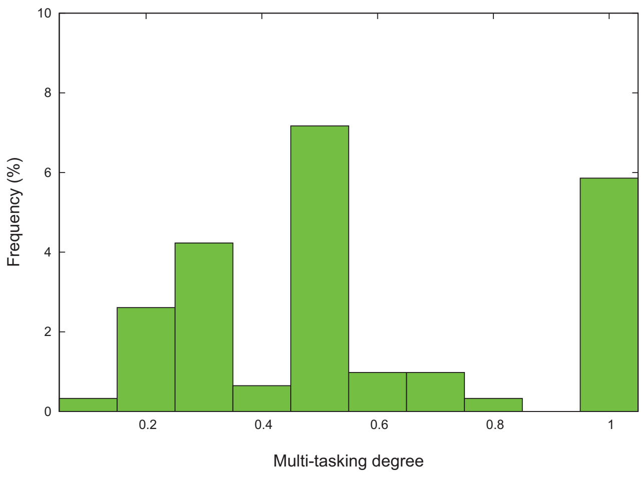


Fig. 10. The distribution of multitasking degree.

consecutive queries are issued within a small enough time window $\bar{t}$, then they are also very likely to be task related. In this approach, only the query submission time of a query, denoted by $\tau(q)$, is taken into account.

Given an adjacent query pair (q_i, q_{i+1}) belonging to the same s_u, the following *binary* similarity function σ_{time} is defined.

$$\sigma_{time}(q_i, q_{i+1}) = \begin{cases} 1, & \text{if } |\tau(q_{i+1}) - \tau(q_i)| \leq \bar{t}; \\ 0, & \text{if } |\tau(q_{i+1}) - \tau(q_i)| > \bar{t}. \end{cases} \tag{3}$$

The effectiveness of this approach depends on the length of the time window $\bar{t}$. Several past works empirically evaluated different values of $\bar{t}$, ranging from 5 to 60 minutes [Silverstein et al. 1999; Shi and Yang 2006; Richardson 2008]. Instead, in this work, we devised a threshold value $\bar{t} = 26$ minutes from a statistical analysis of the query log, as already discussed in Section 3.2.

6.2. Unsupervised Learning Approach

We first discuss two classes of features, that is, *internal* and *external*. Then, we define two task-based query similarity functions which exploit both classes of features.

6.2.1. Feature Selection. Most of the previous approaches used to measure query similarity are based on the lexical content of queries [Salton and Mcgill 1986]. The effectiveness of those approaches is, however, quite low due to the short length of queries (about 2.5 words per query, on average) [Jansen et al. 1998; Silverstein et al. 1999], as well as the lack of contextual information in which queries are issued [Wen et al. 2002].

Some approaches try to expand these short queries by exploiting URLs returned by Web search engines [Glance 2001], as well as retrieved Web documents [Raghavan and Sever 1995], or even Web document snippets [Leung et al. 2008], that is, two queries are similar if they return similar results.

In line with these approaches, we propose two similarity measures by considering queries' *lexical content* and *semantics*.

Content-Based ($\sigma_{content}$). Two queries that share some common terms are likely to be related. Sometimes, the terms may be very similar, but not identical, due to misspelling or different prefixes/suffixes. To capture content similarity even in those cases, we adopt a Jaccard index on character tri-grams [Järvelin et al. 2007]. Let $T(q)$ be the character tri-grams from the terms of query q, we define the similarity $\sigma_{jaccard}$ as follows.

$$\sigma_{jaccard}(q_i, q_j) = \frac{|T(q_i) \cap T(q_j)|}{|T(q_i) \cup T(q_j)|}.$$

In addition, we exploit a normalized Levenshtein similarity $\sigma_{levenshtein}$, which Jones and Klinkner [2008] claimed to be the best edit-based feature for identifying goal boundaries. Finally, the overall content-based similarity is computed as follows.

$$\sigma_{content}(q_i, q_j) = \frac{\sigma_{jaccard}(q_i, q_j) + \sigma_{levenshtein}(q_i, q_j)}{2}.$$

Semantic-Based ($\sigma_{semantic}$). In order to enrich queries with a semantic-based context, it is possible to use external knowledge bases. Several semantic relatedness metrics dealing with semantic resources have previously been proposed. They can be classified as follows.

—*Path-based*, in which knowledge is modeled as a graph of concepts, and the metrics rely on paths over that graph [Rada et al. 1989; Leacock and Chodorow 1998].
—*Information content-based* takes into account the information content of a concept [Resnik 1995].
—*Gloss-based* considers term overlaps between definitions of concepts [Lesk 1986].
—*Vector-based* models each concept as a vector of terms [Gabrilovich and Markovitch 2007] or anchor links [Milne and Witten 2008].

On the basis of the last approach, we assume that either a Wiktionary entry or a Wikipedia article describes a certain *concept* and that the presence of a term in a given article is evidence of the correlation between that term and that concept. We describe the wikification $\vec{C}(t)$ of a term t as its representation in a high-dimensional concept space $\vec{C}(t) = (c_1, c_2, \ldots, c_W)$, where W is the number of articles in the knowledge base, and c_i scores the relevance of the term t for the ith article. We measure the relevance c_i by using the well-known tf-idf score [Salton and Mcgill 1986].

In order to *wikify* a query q, we sum up the contribution from its terms.

$$\vec{C}(q) = \sum_{t \in q} \vec{C}(t).$$

The task relatedness $\sigma_{wikification}(q_i, q_j)$ between two queries q_i and q_j is estimated by the *cosine similarity* of their wikification.

$$\sigma_{wikification}(q_i, q_j) = \frac{\vec{C}(q_i) \cdot \vec{C}(q_j)}{\|\vec{C}(q_i)\| \|\vec{C}(q_j)\|}.$$

We use this wikification process by exploiting the Wiktionary or Wikipedia knowledge bases to compute the query similarity measures $\sigma_{wiktionary}$ and $\sigma_{wikipedia}$. Finally, the overall semantic-based similarity is computed as follows.

$$\sigma_{semantic}(q_i, q_j) = \max(\sigma_{wiktionary}(q_i, q_j), \sigma_{wikipedia}(q_i, q_j)).$$

6.2.2. Similarity Functions. We blend the lexical content $\sigma_{content}$ and semantic $\sigma_{semantic}$ similarity scores is through a *convex combination*.

$$\sigma_1 = \alpha \cdot \sigma_{content} + (1 - \alpha) \cdot \sigma_{semantic}. \quad (4)$$

We also propose a novel similarity function σ_2 on the basis of the following conjecture. If two queries have similar lexical content, they are very likely to be task related, and semantic expansion could be useless. On the other hand, queries could be similar even if they do not share any lexical features, for example, the two queries `nba` and `kobe bryant`. If there is great similarity in content, then we can be confident that queries are task related. Otherwise, we compute the similarity score as the maximum value between content- and semantic-based similarities.

$$\sigma_2 = \begin{cases} \sigma_{content}, & \text{if } \sigma_{content} \geq \mathbf{t}; \\ \max(\sigma_{content}, \mathbf{b} \cdot \sigma_{semantic}), & \text{otherwise.} \end{cases} \quad (5)$$

Both σ_1 and σ_2 rely on the estimation of certain parameters, that is, α for σ_1, $\mathbf{t}$ and $\mathbf{b}$ for σ_2, which were learned directly from the ground truth.

6.3. Supervised Learning Approach

Jones and Klinkner [2008] argued that user search activities can be divided into hierarchical units, that is, *search missions*, which are in turn composed of disjoint *search goals*. According to the authors, a search goal is defined as an atomic information need resulting in one or more queries, while a search mission is a set of topically-related information needs resulting in one or more goals. Therefore a search goal is equivalent to our definition of user task (see Def. 4.1).

Jones and Klinkner [2008] investigated a *supervised learning* approach. Basically, given a pair of queries, they used several query features, namely temporal, word- and character-edit, query log sequence, and Web search features, to predict whether the two queries belonged to the same goal/mission. It is worth noting that the authors did not explore how such binary classifiers could be exploited for segmenting user query streams into goals and missions, and thus for actually discovering search goals, that is, user tasks and search missions.

In addition, we extend our study to include supervised learning approaches in order to detect queries belonging to the same user task. The set of features we use includes those used by Jones and Klinkner [2008], $\mathcal{F}_{jk}$, and the features we have already defined in previous sections.

We train several binary classifiers using various combinations of those features on our ground truth of user tasks. Finally, the output of these classifiers determines the 12 new task-based query similarity scores, which in turn will be exploited by our two best-performing user task discovery methods, as specified later in Section 7.2.1.

6.3.1. Feature Selection.

According to Jones and Klinkner [2008], given any two queries q_i and q_j, the following set of features $\mathcal{F}_{jk}$ provides best accuracy results in predicting whether they are part of the same user task.

—*edlevGT2*. This is a binary feature that evaluates as 1 if the normalized Levenshtein edit distance between q_i and q_j is greater than 2, or 0 otherwise.
—*wordr*. This feature corresponds to the Jaccard distance between the sets of words which q_i and q_j are composed of.
—*char_suf*. This feature counts the number of common characters between q_i and q_j, starting from the right.
—*nsubst_q_j_X*. Given $P(q_i \longrightarrow q_j)$, the probability of q_i being reformulated as q_j, this feature is computed as $count(X : \exists\ P(q_j \longrightarrow X))$.
—*time_diff*. This feature represents the inter-query time gap between q_i and q_j, expressed in seconds.
—*sequential*. This binary feature is positive if the queries q_i and q_j are sequentially issued.
—*prisma*. This feature refers to the cosine distance between vectors obtained from the top-50 Web pages returned as search results for the terms of q_i and q_j, respectively.

—*entropy_q_i_X*. This feature relates to the *entropy* of rewrite probabilities from query q_i and it is computed as $\sum_k P(q_k|q_i)\log_2(P(q_k|q_i))$.

In addition to this set $\mathcal{F}_{jk}$, we propose two further features. The first is the semantic feature we already used for computing the task relatedness measures of our unsupervised learning approach, that is, $\sigma_{wikipedia}$ (see Section 6.2.1). The second is $\sigma_{jaccard\_url}$, which measures the Jaccard similarity between the top 20 URLs returned as search results in response to q_i and q_j. The rationale for this feature is to capture the similarity between two apparently different queries, which share many relevant links to Web documents, that is, URLs retrieved by the most popular Web search engines. Given that $url_{20}(q) = \{u_1, u_2, \ldots, u_{20}\}$ is the set of top-20 URLs returned by a search engine in response to the query q, this feature was computed as follows.

$$\sigma_{jaccard\_url}(q_i, q_j) = \frac{|url_{20}(q_i) \cap url_{20}(q_j)|}{|url_{20}(q_i) \cup url_{20}(q_j)|}.$$

In our test, we retrieved each $url_{20}(q)$ by querying the Yahoo! search engine via its Search Boss API.[6]

6.3.2. Binary Classifiers. We exploited the features described in Section 6.3.1 to train several binary classifiers. In particular, we devised four different combinations of features extracted from our manually-generated ground truth described in Section 5.

—$\mathcal{F}_1 \equiv \mathcal{F}_{jk}$.
—$\mathcal{F}_2 = \mathcal{F}_{jk} \cup \sigma_{wikipedia}$.
—$\mathcal{F}_3 = \mathcal{F}_{jk} \cup \sigma_{jaccard\_url}$.
—$\mathcal{F}_4 = \mathcal{F}_{jk} \cup \sigma_{wikipedia} \cup \sigma_{jaccard\_url}$.

In particular, we used three different kinds of classification algorithms.

—$\mathcal{C}_{dt}$. A clone of the C4.5 decision tree learner [Quinlan 1993].
—$\mathcal{C}_{nb}$. A naïve bayesian learner.
—$\mathcal{C}_{lr}$. A logistic regression learner.

Therefore, the classification step requires training in the preceding set of three classifiers, that is, $\mathcal{C} = \{\mathcal{C}_{dt}, \mathcal{C}_{nb}, \mathcal{C}_{lr}\}$, using the four possible feature sets, that is, $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$. By combining each of the three classifier models with each feature set, we obtained 12 distinct classifiers, that is, $\mathcal{C}_x^y$, where $x \in \{dt, nb, lr\}$ and $y \in \{1, 2, 3, 4\}$.

The training set of each classifier was generated by considering each query pair (q_i, q_j) in our ground truth, and to each of them, we assigned a binary *class attribute* `same_task` = `yes` if and only if q_i and q_j were part of the same task, otherwise `same_task` = `no`. Note that these supervised methods are the only ones which exploit the task labeling of the ground truth.

6.3.3. Similarity Functions. The `same_task = yes` class probability prediction provided by each binary classifier can be interpreted as a *similarity score*. This in turn is exploited by the clustering techniques presented in Section 7.2.1.

Table I summarizes the similarity functions defined as a combination of the classification algorithm and the set of exploited features.

In the following, we describe the performance of each similarity function. All the evaluations were measured on the basis of ten-fold cross-validation.

First, we start by showing some stratified cross-validation statistics, namely *Kappa coefficient*, *Mean absolute error*, *Relative absolute error*, and *Root relative squared error*.

[6]`http://developer.yahoo.com/search/boss/`

Table I. Supervised Task-Based Query Similarity Functions

		Feature Set			
		$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
Classification Algorithm	$\mathcal{C}_{dt}$	σ^1_{dt}	σ^2_{dt}	σ^3_{dt}	σ^4_{dt}
	$\mathcal{C}_{nb}$	σ^1_{nb}	σ^2_{nb}	σ^3_{nb}	σ^4_{nb}
	$\mathcal{C}_{lr}$	σ^1_{lr}	σ^2_{lr}	σ^3_{lr}	σ^4_{lr}

Table II. Statistical Indicators on the Set of Classifiers Derived from $\mathcal{C}_{dt}$

		Statistical Indicators			
		Kappa	*Mean abs. err.*	*Rel. abs. err. (%)*	*Root rel. sq. err. (%)*
Classifiers	$\mathcal{C}^1_{dt}$	0.61	0.02	48.39	72.30
	$\mathcal{C}^2_{dt}$	0.62	0.02	48.22	72.54
	$\mathcal{C}^3_{dt}$	0.63	0.02	47.34	72.40
	$\mathcal{C}^4_{dt}$	**0.63**	0.02	**46.90**	**72.30**

Then, we express the performance of each binary classifier in terms of *TP Rate*, *FP Rate*, *Precision*, *Recall*, and *F1*.

All these measures are shown separately for the two classes of prediction, that is, `same_task` = `yes` and `same_task` = `no`, and then appropriately weight-averaged to provide the reader with a global performance indicator. In particular, TP Rate refers to the ratio of *true positive* examples, that is, queries that are correctly classified (for each of the two classes of prediction), and it is equivalent to Recall. Similarly, FP Rate describes the ratio of *false positive* examples, that is, queries that are misclassified (for each of the two classes of prediction). Furthermore, Precision is the proportion of examples of a certain class among all those that are classified with that class. Finally, F1 is the harmonic mean of Precision and Recall.

$$F1 = \frac{2 \times Precision \times Recall}{(Precision + Recall)}.$$

Note that the distribution of query pairs across these two classes are far from being uniform: the class `same_task` = `yes` is much less probable than `same_task = no`. Therefore, the classifier performance measured on the first class might be noticeably lower than the one computed on the second class.

Decision Tree Classifier ($\mathcal{C}_{dt}$). This classification algorithm is based on a clone of the C4.5 decision tree learner [Quinlan 1993]. First, Table II shows some stratified cross-validation statistical indicators on the four binary classifiers, that is, $\mathcal{C}^1_{dt}, \mathcal{C}^2_{dt}, \mathcal{C}^3_{dt}$, and $\mathcal{C}^4_{dt}$. Here, it is worth noting that our newly introduced features, namely $\sigma_{wikipedia}$ and $\sigma_{jaccard\_url}$, improve the performance of the classifier, especially when both are used, as in the case of $\mathcal{C}^4_{dt}$.

Moreover, Table III describes the performance of the binary classifiers in terms of TP Rate, FP Rate, Precision, Recall, and F1. For each classifier, we indicate the values of all the preceding measures both individually for each class of prediction and globally, by weight-averaging them over the two classes.

Interestingly, we can derive that our newly proposed features affect two complimentary aspects of the performance. Indeed, on the one hand, the introduction of $\sigma_{wikipedia}$ increases the recall of positive examples, that is, those that are actually labeled with `same_task` = `yes`. On the other hand, the usage of $\sigma_{jaccard\_url}$ helps to increase the precision of positive examples. Finally, when combined together in $\mathcal{C}^4_{dt}$, we obtained the

Table III. Performance Evaluation of the Set of Classifiers Derived from $\mathcal{C}_{dt}$

		Accuracy Measures					
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F1*	`same_task`
Classifiers	$\mathcal{C}_{dt}^1$	0.997	0.473	0.990	0.997	0.993	no
		0.527	0.003	0.768	0.527	0.625	yes
		0.987	0.463	0.985	0.987	0.985	(*weighted avg.*)
	$\mathcal{C}_{dt}^2$	0.996	0.467	0.990	0.996	0.993	no
		0.533	0.004	0.760	**0.533**	0.627	yes
		0.987	0.457	0.985	0.987	0.985	(*weighted avg.*)
	$\mathcal{C}_{dt}^3$	0.997	0.467	0.990	0.997	0.993	no
		0.531	0.003	**0.786**	0.531	0.635	yes
		0.987	0.457	0.986	0.987	0.986	(*weighted avg.*)
	$\mathcal{C}_{dt}^4$	0.997	0.460	0.990	0.997	0.993	no
		0.540	0.003	0.780	**0.540**	**0.640**	yes
		0.987	**0.450**	**0.986**	**0.987**	**0.986**	(*weighted avg.*)

Table IV. Statistical Indicators on the Set of Classifiers Derived from $\mathcal{C}_{nb}$

		Statistical Indicators			
		Kappa	*Mean abs. err.*	*Rel. abs. err. (%)*	*Root rel. sq. err. (%)*
Classifiers	$\mathcal{C}_{nb}^1$	0.52	0.03	76.01	91.26
	$\mathcal{C}_{nb}^2$	0.50	0.02	**59.14**	89.16
	$\mathcal{C}_{nb}^3$	**0.53**	0.03	76.69	91.65
	$\mathcal{C}_{nb}^4$	0.51	0.02	59.51	89.21

Table V. Performance Evaluation of the Set of Classifiers Derived from $\mathcal{C}_{nb}$

		Accuracy Measures					
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F1*	`same_task`
Classifiers	$\mathcal{C}_{nb}^1$	0.992	0.499	0.989	0.992	0.991	no
		0.501	0.008	0.576	0.501	0.536	yes
		0.982	**0.488**	**0.981**	**0.982**	**0.981**	(*weighted avg.*)
	$\mathcal{C}_{nb}^2$	0.992	0.527	0.989	0.992	0.990	no
		0.473	0.008	0.563	0.473	0.514	yes
		0.981	0.516	0.980	0.981	0.980	(*weighted avg.*)
	$\mathcal{C}_{nb}^3$	0.992	0.499	0.989	0.992	0.991	no
		0.501	0.008	0.574	0.501	0.535	yes
		0.982	**0.489**	**0.981**	**0.982**	**0.981**	(*weighted avg.*)
	$\mathcal{C}_{nb}^4$	0.992	0.526	0.989	0.992	0.990	no
		0.474	0.008	0.562	0.474	0.514	yes
		0.981	0.515	0.980	0.981	0.980	(*weighted avg.*)

very best results. This means that the similarity function of choice for this classification algorithm is σ_{dt}^4.

Naïve Bayesian Classifier ($\mathcal{C}_{nb}$). This classification algorithm is based on a naïve bayesian learner. Table IV shows the statistical indicators for each classifier, that is, $\mathcal{C}_{nb}^1$, $\mathcal{C}_{nb}^2$, $\mathcal{C}_{nb}^3$, and $\mathcal{C}_{nb}^4$. Moreover, Table V describes the performance of these four binary classifiers.

In this case, the very best accuracy is obtained both with $\mathcal{C}_{nb}^1$ and $\mathcal{C}_{nb}^3$, by using the set of features $\mathcal{F}_1$ and $\mathcal{F}_3$, respectively. This means that the newly introduced features, namely $\sigma_{wikipedia}$ and $\sigma_{jaccard\_url}$, do not significantly improve the performance of the classifier. Therefore, the chosen similarity functions can be either σ_{nb}^1 or σ_{nb}^3.

Table VI. Statistical Indicators on the Set of Classifiers Derived from $\mathcal{C}_{lr}$

		Statistical Indicators			
		Kappa	*Mean abs. err.*	*Rel. abs. err. (%)*	*Root rel. sq. err. (%)*
Classifiers	$\mathcal{C}_{lr}^1$	0.48	0.02	60.89	78.03
	$\mathcal{C}_{lr}^2$	0.48	0.02	60.84	77.99
	$\mathcal{C}_{lr}^3$	0.48	0.02	60.88	78.02
	$\mathcal{C}_{lr}^4$	0.48	0.02	**60.82**	**77.98**

Table VII. Performance Evaluation of the Set of Classifiers Derived From $\mathcal{C}_{lr}$

		Accuracy Measures					
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F1*	`same_task`
Classifiers	$\mathcal{C}_{lr}^1$	0.997	0.630	0.987	0.997	0.992	no
		0.370	0.003	0.702	0.370	0.484	yes
		0.983	0.617	0.981	0.983	0.981	(*weighted avg.*)
	$\mathcal{C}_{lr}^2$	0.997	0.629	0.987	0.997	0.992	no
		0.371	0.003	0.703	0.371	0.485	yes
		0.983	0.616	0.981	0.983	0.981	(*weighted avg.*)
	$\mathcal{C}_{lr}^3$	0.997	0.625	0.987	0.997	0.992	no
		0.375	0.003	0.702	0.375	0.489	yes
		0.983	0.612	0.981	0.983	0.981	(*weighted avg.*)
	$\mathcal{C}_{lr}^4$	0.997	0.622	0.987	0.997	0.992	no
		0.380	0.003	0.705	0.380	**0.492**	yes
		0.984	**0.609**	0.981	0.984	0.981	(*weighted avg.*)

Logistic Regression Classifier ($\mathcal{C}_{lr}$). This classification algorithm is based on logistic regression. Table VI describes the statistical indicators of four binary classifiers, that is, $\mathcal{C}_{lr}^1, \mathcal{C}_{lr}^2, \mathcal{C}_{lr}^3$, and $\mathcal{C}_{lr}^4$, which are obtained using this approach in combination with the sets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$, and $\mathcal{F}_4$. As this table highlights, no significant differences arise from this comparison. Instead, a more detailed evaluation of the performance of each classifier is provided in Table VII. Although all four classifiers behave similarly in general, comments can be made nevertheless. In particular, it is worth noting that adding our new set of features results in better true positive and false positive rates. It is nonetheless true that these enhancements are not crucial to the overall performance. Therefore, any classifier could be chosen almost arbitrarily as well, as their relating similarity functions, that is, σ_{lr}^*, where $* \in \{1, 2, 3, 4\}$.

We can conclude that the very best performing classifier is $\mathcal{C}_{dt}^4$. Indeed, considering the weighted-average performances, it gains nearly 0.5% in terms of F1, and it reduces the FP Rate by about 8.4% to the best naïve bayesian classifiers, that is, $\mathcal{C}_{nb}^1$ and $\mathcal{C}_{nb}^3$. Similarly, it gains roughly 0.5% in terms of F1 and it reduces the FP Rate by approximately 35.3% to the best logistic regression classifiers, that is, $\mathcal{C}_{lr}^4$. This means that σ_{dt}^4 could be considered the very best query similarity function in order to determine task relatedness.

As a last note, we would like to comment on why the supervised learning approach proposed by Jones and Klinkner [2008] alone is not suitable for effectively discovering user tasks, and why we used it only to learn the task relatedness, which in turn is fed into more complex user task discovery methods, described in Section 7.

Let us consider only the very best classifier, namely $\mathcal{C}_{dt}^4$. Among a total of 113,474 classified query pairs, 112,009 (i.e., 98.7%) were correctly classified. However, the distribution of query pairs across the two classes is very skewed, since 111,080 (i.e., 97.9%) belong to one class, namely `same_task` = `no`. It turns out that evaluating the performance of the classifier only in terms of its accuracy might overestimate its actual effectiveness. A fairer approach is to validate the classifier on the rarest class, which is `same_task = yes`. If we focus only on the ability of the classifier to correctly predict queries that were actually in the same tasks, then precision reaches at most 78% in the best case, which is considerably lower than the 98.7% obtained on average.

Section 8.2.3 shows the outcomes from the two best-performing user task discovery methods using the three best supervised similarity scores, that is, σ^4_{dt}, σ^1_{nb} (or, equivalently, σ^3_{nb}), and σ^*_{lr}, where $* \in \{1, 2, 3, 4\}$.

7. DISCOVERING USER TASKS

In this section, we tackle the user task discovery problem (UTDP) in Section 4.1, as well as present and discuss several clustering techniques which adopt the query relatedness measures presented in Section 6. We use as baseline the time-based task relatedness measure σ_{time} (see Section 6.1), that is, a simple splitting by using a time threshold $\bar{t}$.

7.1. Time Splitting

For any consecutive query pair (q_i, q_{i+1}), if $\sigma_{time}(q_i, q_{i+1}) = 1$, then time splitting considers both queries as part of the same session. Otherwise, q_i is the last (resp. q_{i+1} is the first) query in a distinct session. The time complexity of time-splitting is linear in the number of input queries, and in this work, we use time splitting as the preprocessing step in order to approach UTDP. In fact, there exist task-oriented sessions that are made up of sequences of consecutive queries (i.e., no multitasking). In such cases, time-splitting methods are a suitable choice. Therefore, we choose to adopt time-splitting techniques as the baseline method to discover user tasks, by using the time thresholds TS-5 and TS-15, that is, 5- and 15-minute thresholds, as well as TS-26 [Silverstein et al. 1999; He and Göker 2000] (Section 8.2). The threshold of $\bar{t} = \Delta = 26$ minutes (TS-26) was determined on the basis of the statistical analysis we conducted on the testing dataset (see Section 3.2).

Time-splitting techniques are not able to deal with task-related sessions, since they can only identify sequences of timely-consecutive queries, whereas multitasking sessions represent a significant sample of all the task-related sessions (see Section 5 for more details on the analysis).

7.2. Query Clustering

In order to discover user tasks, in the following, we present several clustering algorithms that we apply to TS-26 split time-gap sessions

We start by describing two algorithms derived from well-known clustering methods: QC-MEANS [MacQueen 1967] and QC-SCAN [Ester et al. 1996]. In addition, we introduce two graph-based techniques: QC-WCC and its computationally-lighter variation QC-HTC. The effectiveness of all these methods mostly depends on the *robustness* of the *similarity functions*, that is, the measures of task-based query similarity (as described in Section 6) which are exploited by the algorithms.

7.2.1. Algorithms. While QC-MEANS and QC-SCAN are inspired to well-known clustering algorithms, QC-WCC and QC-HTC follow a graph-based approach. QC-WCC identifies user tasks from the connected components of a query similarity graph, while QC-HTC, which is a variation of QC-WCC, is aimed at reducing the computational cost of clustering without affecting the overall effectiveness.

Each query clustering method is associated with a specific partitioning strategy π and operates on each time-gap session. Let $s = \langle q_i \dots q_{i+n-1} \rangle$ be a generic time-gap session belonging to a long-term session $\mathcal{S}$ of user u, where $|s| = n$, and $i \geq 1, 1 \leq n \leq |\mathcal{S}| - i + 1$. It is worth noting that we do not use the subscript symbol u when there is no ambiguity for users in order to simplify the readability of notation. Each algorithm provides as output $\pi(s) = \{t_1, t_2, \ldots t_{|\pi(s)|}\}$, that is, the set of user tasks of s, obtained by applying a partitioning strategy $\pi \in$ {QC-MEANS, QC-SCAN, QC-WCC, QC-HTC}.

QC-Means. This is a *centroid-based* algorithm and a variation of the well-known K-MEANS [MacQueen 1967]. We replaced the usual K parameter, that is, the number of clusters to be extracted, with a ρ threshold which establishes the maximum radius of each cluster. This allowed us to better deal with the varying lengths of the various user sessions as well as to avoid specifying the number K of final clusters a priori.

At each step, a query $q_i \in s \subseteq \mathcal{S}$ is either added to an existing cluster of queries t_j if its similarity with respect to the centroid query of t_j is at least $1 - \rho$; otherwise q_i itself becomes the centroid of a new cluster t_k. The worst case is when each cluster contains a single query. In this case, we need to compute the similarity between all query pairs and the complexity of QC-MEANS becomes quadratic in the size of the input, that is, $O(n^2)$.

QC-SCAN. It is the *density-based* DB-SCAN algorithm [Ester et al. 1996], specifically tailored to extract user tasks from Web search engine query logs. The rationale for also evaluating a variation of DB-SCAN is that a centroid-based approach may suffer from the presence of noise in query logs. Again, QC-SCAN may require computing the similarity of all query pairs, thereby making its worst-case time complexity quadratic in the size of the input.

QC-WCC. This algorithm extracts query clusters corresponding to *weighted connected components* of a graph [Lucchese et al. 2011]. Given a time-gap session $s \subseteq \mathcal{S}$, we first build a complete graph $G_s = (V, E, w)$ whose vertices V are the queries in s, that is, $V = \{q_i \mid q_i \in s\}$, and whose E edges are weighted by the similarity of the corresponding vertices. The weighting function w, $w : E \longmapsto [0, 1]$, is computed in terms of the task-based query similarity functions proposed in Section 6. Thus, the graph G_s models the task-based similarity between any pair of queries in the given time-gap session.

The algorithm works in two steps. In the first, given the graph G_s, we remove *weak edges* whose weights are smaller than a given threshold, that is, $w(e) < \eta$, thus obtaining a pruned graph G'_s. In the second step, we extract the connected components of the graph and consider them as clusters of task-related queries $\pi(s) = \{t_1, t_2, \ldots t_{|\pi(s)|}\}$.

Assuming a robust similarity function, the QC-WCC algorithm is able to handle the multitasking nature of users sessions. Groups of related queries are isolated by the pruning of weak edges. Links with high similarity identify the generalization/specialization steps of the users, as well as restart from a previous query when the current query chain is found to be unsuccessful.

The computational complexity of QC-WCC is dominated by the construction of the graph G_s. The similarity between any pair of edges must be computed, resulting in a number of computations which is quadratic in the number of vertices, that is, $O(|s|^2)$. On the other hand, the connected components of a graph can be easily computed in linear time (in terms of the numbers of vertices and edges of the graph) using either breadth-first search or depth-first search [Hopcroft and Tarjan 1973]. In either case, a search that begins at a particular vertex v eventually finds the entire connected component containing v before returning.

QC-HTC. This is a variation of the preceding QC-WCC algorithm, which does not need to compute the full similarity graph yet maintains the quality of the obtained query clustering QC-WCC [Lucchese et al. 2011]. The graph we consider is not complete. We use an edge-weighting function w, $w : E \longmapsto [0, 1]$, which is computed in terms of the task-based query similarity functions proposed in Section 6. Similarly to QC-WCC, for

QC-HTC, we also exploit a threshold η: two queries q and q' cannot be considered task related if $w(e(q,q')) < \eta$, where $e(q,q') \in E$.

The algorithm works in two phases. In the first, we identify *query chains* within each time-gap session $s \subseteq \mathcal{S}$. Each chain, called a *sequential cluster*, is denoted by $\tilde{t}_j$ and only contains consecutive queries in a given time-gap session, where each query is similar (task related) to the chronologically following one. This means that to detect the various $\tilde{t}_j$, we only need to compute the weights of the edges $e(q_i, q_{i+1})$, where queries q_i and q_{i+1} occur consecutively in the session. Note that a chain of k task-related queries $(q_j, \ldots, q_{j+k})$ must be maximal. If q_{j-1} (resp. q_{j+k+1}) exists in s, then $w(e(q_{j-1}, q_j)) < \eta$ (resp. $w(e(q_{j+k}, q_{j+k+1})) < \eta$).

The rationale for first detecting query chains is that without losing generality, a user task can be decomposed into a set of these chains even in the presence of multitasking. Unsurprisingly, due to multitasking, chains of different user tasks can interleave in a given time-gap session. Thus, the algorithm has to finally identify user tasks by recomposing these chains.

The latter phase of the algorithm therefore merges the sequential clusters but does not compute the similarity measures between all the queries included in each cluster.

Instead, we guess that a sequential cluster can be well described by its (chronologically-) first and last queries, respectively, denoted by $head(\tilde{t}_j)$ and $tail(\tilde{t}_j)$ [Lucchese et al. 2011]. This is because a user involved in a given task often carries out a process of specialization/generalization of queries, and thus the middle queries might be less representative of the user's real intent. For example, two users could start a chain from the same query (head) and end in two different query specializations (tail), whereas they could start a chain from different queries (head) and end in the same specialization (tail). Therefore, the similarity *sim* between two sequential clusters $\tilde{t}_j$, $\tilde{t}_k$ is computed as follows.

$$sim(\tilde{t}_j, \tilde{t}_k) = \min_{q \in \{head(\tilde{t}_j), tail(\tilde{t}_j)\} \quad q' \in \{head(\tilde{t}_k), tail(\tilde{t}_k)\}} w(e(q,q')),$$

where w weights the edge $e(q,q')$ linking the queries q and q' with respect to their task-based similarity, analogously to QC-WCC.

We can finally discuss in more detail how this second clustering phase works. The first cluster t_1 is initialized with the oldest sequential cluster $\tilde{t}_1$ in a given session, and $\tilde{t}_1$ is removed from the set of sequential clusters. Then, t_1 is compared with any other chronologically-ordered sequential cluster $\tilde{t}_j$ by computing the similarity as previously. We still use the threshold η: only if $sim(t_1, \tilde{t}_j) \geq \eta$, then $\tilde{t}_j$ is merged into t_1, the *head* and *tail* queries of t_1 are updated consequently, and $\tilde{t}_j$ is removed from the set of sequential clusters. The algorithm continues comparing the new cluster t_1 with the remaining sequential clusters. When all the sequential clusters have been considered, the oldest sequential cluster available is used to build a new cluster t_2, and so on. The algorithm iterates this procedure until no more sequential clusters are left.

The worst-case complexity of QC-HTC is still quadratic in the number of queries in s; in practice there are frequent cases in which the real execution time results to be greatly reduced with respect to QC-WCC. First, note that the first step of QC-HTC only computes the similarity between time-adjacent queries, and thus its computational cost is linear in the number of queries in s. We already showed that 52.8% of the time-gap sessions contain one user task only. Hence, it is highly likely that such user tasks are just found after the first step of the algorithm, if these tasks exactly correspond to chains of task-related queries. To detect multitasking sessions, the second step of the algorithm merges chains, and thus the complexity of is quadratic in the number m of sequential clusters extracted, that is, $O(m^2)$. If $m = \beta \cdot |s|$, with $0 < \beta \leq 1$, while the asymptotic complexity is still quadratic in $|s|$ since β is a constant, in practice the execution time of the second step is reduced by a factor β^2. In addition, the algorithm can run even faster, since QC-HTC algorithm does not compute all the pairwise similarities among the sequential clusters in advance.

8. EXPERIMENTS ON USER TASK DISCOVERY

In this section, we discuss the results obtained with all the user task discovery methods for approaching the UTDP, which were described in Section 7. In addition, we compare our results with those provided by two other task discovery methods: (i) the simple time-splitting technique TS-26, which is considered as the baseline solution, and (ii) the session extraction method based on the query-flow graph (QFG) proposed by Boldi et al. [2008], which can be considered as the state-of-the-art approach.

For all our clustering methods we can either use the *unsupervised* learned task-based query similarities (i.e., σ_1 and σ_2) or the 12 *supervised* learned similarities. We have chosen to start from the unsupervised learned similarity functions [Lucchese et al. 2011], and we show that QC-WCC and QC-HTC outperform both QC-MEANS and QC-SCAN, but also state-of-the-art approaches, that is, QFG introduced by Boldi et al. [2008]. Then, we concentrate on QC-WCC and QC-HTC only and instantiate their function w for weighting the query graph with the supervised learned similarities, that is, $w = \sigma^4_{dt}$, $w = \sigma^1_{nb}$ (or, analogously, $w = \sigma^3_{nb}$), and $w = \sigma^*_{lr}$, where $* \in \{1, 2, 3, 4\}$.

8.1. Measures of Clustering Validity

In order to evaluate all the methods we mentioned, we needed to measure the degree of correspondence between manually-extracted user tasks of the ground truth (see Section 5) and user tasks produced by our algorithms. To this end, we used both *classification-* and *similarity-oriented* measures [Tan et al. 2005]. In the following, *predicted class* is the user task where a query is assigned to by a specific algorithm, whereas *true class* indicates the user task where the same query was in the ground truth.

Classification-oriented approaches measure the degree to which predicted classes correspond to true classes, and F1 is one of the most popular scores in this category, as it combines both precision and recall. In our case, *precision* measures the fraction of queries that were assigned to a user taskand that were actually part of that user task. Instead, *recall* measures how many queries were assigned to a user task among all the queries that were really contained in that user task. Globally, F1 evaluates the extent to which a user task contains only and all the queries that were actually part of it. Given $p(i, j)$, $r(i, j)$ is the precision and recall of user task i with respect to class j, the F1 corresponds to the following weighted harmonic mean of $p(i, j)$ and $r(i, j)$.

$$F1(i, j) = \frac{2 \times p(i, j) \times r(i, j)}{p(i, j) + r(i, j)}.$$

To compute a global F1, we first considered the set of *predicted tasks* $\mathcal{T}$ associated with each long-term session $\mathcal{S}$, which is obtained as $\mathcal{T} = \bigcup_{s \in \mathcal{S}} \pi(s) = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$, namely as the union set of all the user tasks extracted from each time-gap session by using the partitioning strategy π. Analogously, we took into account the set of *true tasks* $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\}$, that is, the set of tasks performed by user u according to the ground truth.

In addition, in order for the two sets $\mathcal{T}$ and Θ to have the same size, that is, $|\mathcal{T}| = |\Theta|$, we padded them with all the *unclassified* queries, which are all the queries that appear in session $\mathcal{S}$ but that were discarded during the automatic and/or the manual clustering. This can, in some way, be equivalent to considering discarded queries as singleton clusters, that is, single tasks composed of only one query.

Thus, for each predicted task t_j, we computed the *maximum* F1, that is, $F1_{\max}(t_j)$, with respect to the true tasks as follows.

$$F1_{\max}(t_j) = \underset{k}{\operatorname{argmax}}\ F1(t_j, \theta_k).$$

Globally, F1 is averaged on the set of all predicted tasks for all the users $u \in \mathcal{U}$ in the training set $\mathcal{T} = \bigcup_{u\in\mathcal{U}} \mathcal{T}_u$ with respect to the set of all true tasks $\Theta = \bigcup_{u\in\mathcal{U}} \Theta_u$ as follows.

$$F1(\mathcal{T}, \Theta) = \frac{w_j \cdot F1_{\max}(t_j)}{\sum_{j=1}^{|\mathcal{T}|} w_j},$$

where $w_j = |t_j|$.

Similarity-oriented measures consider pairs of objects instead of single objects. Again, let $s \subseteq \mathcal{S}$ be the generic time-gap session of a long-term session $\mathcal{S}$ such that $|s| > 1$. Furthermore, let $\mathcal{T}$ and Θ be the sets of predicted and true tasks of $\mathcal{S}$, respectively (both padded with discarded queries as described previously). Thus, for each $\mathcal{S}$ we computed the following quantities.

—tn = number of query pairs that are in different true tasks and in different predicted tasks (*true negatives*).
—fp = number of query pairs that are in different true tasks but in the same predicted task (*false positives*).
—fn = number of query pairs that are in the same true task but in different predicted tasks (*false negatives*).
—tp = number of query pairs that are in the same true task and in the same predicted tasks (*true positives*).

Then, we used two different measures.

—*Rand index.* $R(\mathcal{T}) = \frac{tn+tp}{tn+fp+fn+tp}$.
—*Jaccard index.* $J(\mathcal{T}) = \frac{tp}{fp+fn+tp}$.

A global value of both the Rand and Jaccard index, that is, R and J respectively, might be computed as follows:

$$R = \frac{w_j \cdot R(\mathcal{T})}{\sum_{j=1}^{|\mathcal{T}|} w_j}, \qquad J = \frac{w_j \cdot J(\mathcal{T})}{\sum_{j=1}^{|\mathcal{T}|} w_j},$$

where $w_j = |\mathcal{S}|$.

As specified before, when computing both the Rand and Jaccard index, we did not consider time-gap sessions containing only one singleton task, that is, time-gap sessions containing only one single-query cluster. However, we did take into account time-gap sessions that were composed of a single task with more than one query.

8.2. Evaluation on the Ground Truth

In the following, we show the results we obtained using our two sets of user task discovery methods, namely *time-splitting* and *query clustering* methods, respectively. Also, we compare them with a state-of-the-art approach based on the query-flow graph (QFG) [Boldi et al. 2008].

8.2.1. Time-Splitting. This set of task discovery methods is exclusively based on the task-based query similarity function described in Section 6.1, that is, σ_{time}. In particular, here we compare three different time-splitting techniques—TS-5, TS-15, and TS-26—which use 5-, 15-, and 26-minute thresholds of $\bar{t}$, respectively.

Table VIII. TS-5, TS-15, and TS-26

	F1	*Rand*	*Jaccard*
TS-5	0.28	**0.75**	0.03
TS-15	0.28	0.71	0.08
TS-26	**0.65**	0.34	**0.34**

Table IX. QFG Varying the Threshold η

	η	*F1*	*Rand*	*Jaccard*
QFG	0.1	0.68	0.47	0.36
	0.2	0.68	0.49	0.36
	0.3	0.69	0.51	0.37
	0.4	0.70	0.55	0.38
	0.5	0.71	0.59	0.38
	0.6	0.74	0.65	0.39
	0.7	**0.77**	**0.71**	**0.40**
	0.8	0.77	0.71	0.40
	0.9	0.77	0.71	0.40

Table VIII shows the results we obtained using these techniques on the ground truth. The best result in terms of F1 is found by considering all the time-gap sessions identified with TS-26, without splitting them into shorter time-gap sessions. Hence, we consider TS-26 as the baseline approach for addressing UTDP. Roughly speaking, this is equivalent to identifying user tasks with time-gap sessions.

8.2.2. Query-Flow Graph. QFG is constructed over a training segment of the AOL top-500 user sessions. This method uses *chaining probabilities* measured by means of a machine-learning method. First, we extracted some features from the training search engine log and stored them in a compressed graph representation. In particular, we considered 25 different features (i.e., time-related, session, and textual features) for each pair of queries (q, q') that were issued consecutively in at least one session of the query log.

The validity of QFG was tested on the ground truth, and the results we obtained are shown in Table IX. We found the best values using a threshold $\eta = 0.7$. In fact, it was shown that results do not improve when using a greater threshold value.

QFG significantly improves the baseline TS-26. In particular, F1 gains about 16%, while Rand and Jaccard roughly gain 52% and 15%, respectively.

8.2.3. Query Clustering. We now evaluate all the clustering-oriented user task discovery methods described in Section 7.2.1.

First, we present the results we obtained using the task-based query similarity functions derived from the unsupervised learning approach described in Section 6.2 that is, σ_1 and σ_2. Therefore, as a major innovative contribution to this work, we also show the outcomes of two of these task discovery methods, that is, QC-wcc and QC-htc, when exploiting the supervised learned similarities proposed in Section 6.3, that is, σ^4_{dt}, σ^1_{nb} (or, equivalently, σ^3_{nb}), and σ^*_{lr}, where $* \in \{1, 2, 3, 4\}$.

Unsupervised Learned Task-Based Similarity. We start on the premise that the QC-Means clustering algorithm uses both the unsupervised learned query similarities σ_1 and σ_2. We empirically set the radius ρ of this centroid-based algorithm to 0.4 for both similarity functions, that is, two queries could be part of the same cluster if and only if their similarity is equal to or greater than 0.6. The overall results of this method are shown in Table X.

Table X. QC-Means Using Unsupervised Learned Task-Based Query Similarities σ_1 and σ_2

QC-Means σ_1				
		F1	*Rand*	*Jaccard*
α	$(1-\alpha)$			
1	0	0.71	0.73	0.26
0.5	0.5	0.68	0.70	0.14
0	1	0.68	0.70	0.13

QC-Means σ_2				
		F1	*Rand*	*Jaccard*
t	**b**			
0.5	4	**0.72**	**0.74**	**0.27**

Table XI. QC-Scan Using Unsupervised Learned Task-Based Query Similarities σ_1 and σ_2

QC-Scan σ_1				
		F1	*Rand*	*Jaccard*
α	$(1-\alpha)$			
1	0	0.77	0.71	0.17
0.5	0.5	0.74	0.68	0.06
0	1	0.75	0.68	0.07

QC-Scan σ_2				
		F1	*Rand*	*Jaccard*
t	**b**			
0.5	4	**0.77**	**0.71**	**0.19**

Concerning σ_1, the best results were obtained by using only the content-based similarity, that is, with $\alpha = 1$. However, the very best results for QC-Means were found when using σ_2. Here, we significantly improve the baseline TS-26 in terms of F1 ($\approx$10%) and Rand ($\approx$54%), while we lose nearly 21% in terms of Jaccard. Moreover, if we compare the best QC-Means with the best QFG, we notice that QC-Means loses about 6% in terms of F1, 33% in terms of Jaccard, but it gains approximately 4% in terms of Rand.

We now discuss the QC-Scan algorithm, again using both the similarity functions σ_1 and σ_2. We used several combinations of the two density-based parameters, that is, *minPts* and *eps*, and we found the best results with $minPts = 2$ and $eps = 0.4$.

Table XI illustrates the fact that QC-Scan provides globally better results than QC-Means for both σ_1 and σ_2. Similarly, for σ_1 the best results were obtained by using only content-based similarity, that is, with $\alpha = 1$. However, our proposed conditional function μ_2 reveals a significant improvement with respect to all measures.

Finally, it is worth noting that QC-Scan behaves exactly the same as QFG, except for the Jaccard where QC-Scan roughly loses 53%.

The third algorithm we consider is QC-wcc. Here, we used a breadth-first search in order to find the connected components of the graph which represent each time-gap session [Hopcroft and Tarjan 1973]. Table XII shows the results we found using this algorithm either with σ_1 and σ_2, and by varying the pruning threshold η. In particular, concerning σ_1 we only consider the best convex combination when $\alpha = 0.5$.

The best results with σ_1 were obtained when $\eta = 0.2$, while even better results were found with σ_2 when $\eta = 0.3$. In this last case, the overall evaluation is significantly higher than the baseline TS-26 but also higher than the state-of-the-art approach QFG. With regard to TS-26, the best QC-wcc gains about 20%, 56%, and 23% in terms of F1, Rand, and Jaccard, respectively. Moreover, QC-wcc also improves the results of QFG, gaining nearly 5% in terms of F1, about 9% in terms of Rand, and approximately 10% in terms of Jaccard.

QC-htc is the last algorithm we introduced and represents one of the innovative contributions to our previous work [Lucchese et al. 2011]. The results obtained from using this approach with both similarity functions σ_1 and σ_2 varying the pruning threshold η are shown in Table XIII. Similarly to QC-wcc, with regard to σ_1, we only consider the best convex combination when $\alpha = 0.5$. Again, the best results with σ_1 were obtained when $\eta = 0.2$, while the global best results were found with σ_2 when $\eta = 0.3$. As the table shows, the overall results are very close to those obtained with QC-wcc. In particular, QC-htc improves TS-26 by roughly gaining 19%, 56%, and 21% in terms

Table XII. QC-WCC Using Unsupervised Learned Task-Based Query Similarities σ_1 and σ_2

QC-WCC $_{\sigma_1}$ ($\alpha = 0.5$)			
η	*F1*	*Rand*	*Jaccard*
0.1	0.78	0.71	0.42
0.2	**0.81**	**0.78**	**0.43**
0.3	0.79	0.77	0.37
0.4	0.75	0.73	0.27
0.5	0.72	0.71	0.20
0.6	0.75	0.70	0.14
0.7	0.74	0.69	0.11
0.8	0.74	0.68	0.07
0.9	0.72	0.67	0.04

QC-WCC $_{\sigma_2}$ ($\mathbf{t} = 0.5$, $\mathbf{b} = 4$)			
η	*F1*	*Rand*	*Jaccard*
0.1	0.67	0.45	0.33
0.2	0.78	0.71	0.42
0.3	**0.81**	**0.78**	**0.44**
0.4	0.81	0.78	0.41
0.5	0.80	0.77	0.37
0.6	0.78	0.75	0.32
0.7	0.75	0.73	0.23
0.8	0.71	0.70	0.15
0.9	0.69	0.68	0.08

Table XIII. QC-HTC Using Unsupervised Learned Task-Based Query Similarities σ_1 and σ_2

QC-HTC $_{\sigma_1}$ ($\alpha = 0.5$)			
η	*F1*	*Rand*	*Jaccard*
0.1	0.78	0.72	0.41
0.2	**0.80**	**0.78**	**0.41**
0.3	0.78	0.76	0.35
0.4	0.75	0.73	0.25
0.5	0.73	0.70	0.18
0.6	0.75	0.70	0.13
0.7	0.74	0.69	0.10
0.8	0.74	0.68	0.06
0.9	0.72	0.67	0.03

QC-HTC $_{\sigma_2}$ ($\mathbf{t} = 0.5$, $\mathbf{b} = 4$)			
η	*F1*	*Rand*	*Jaccard*
0.1	0.68	0.56	0.32
0.2	0.78	0.73	0.41
0.3	**0.80**	**0.78**	**0.43**
0.4	0.80	0.77	0.38
0.5	0.78	0.76	0.34
0.6	0.77	0.74	0.30
0.7	0.74	0.72	0.21
0.8	0.71	0.70	0.14
0.9	0.68	0.67	0.07

of F1, Rand, and Jaccard, respectively. It is therefore clear that QC-HTC provides better results than QFG and gains about 4% in terms of F1, nearly 9% in terms of Rand, and approximately 8% in terms of Jaccard.

Supervised Learned Task-Based Similarity. Another major contribution to this work concerns the supervised learning approach for computing the task-based query similarity functions, as described in Section 6.3.

In short, a set of query similarity functions was learned by training a family of classifiers on a set of both internal and external query log features. This contrasts with the unsupervised learning approach, where query similarity functions were directly derived from the query log data without any supervised learning step.

Therefore, here we also evaluate how this new approach for measuring the task relatedness between query pairs impacts the effectiveness of the two best-performing clustering-oriented task discovery methods, that is, QC-WCC and QC-HTC.

It is worth remembering that supervised learned similarities affect the way in which we build the similarity graph either in QC-WCC or in QC-HTC. Indeed, an edge between a query pair (q_i, q_j) is created whenever the considered classifier assigns the class attribute `same_task` = `yes` to (q_i, q_j). Moreover, the weight assigned to each created edge corresponds to the prediction accuracy value provided by the classifier.

Based on the performance evaluation of the classifiers we proposed in Section 6.3.3, we ran both the QC-WCC and QC-HTC algorithms using the three best task-based query similarity functions: σ_{dt}^4, σ_{nb}^1 (or, analogously, σ_{nb}^3), and σ_{lr}^*, where $* \in \{1, 2, 3, 4\}$. These similarity scores were used to compute the weighting edge similarity function w of our graph-based algorithms.

Table XIV. QC-wcc vs. QC-htc Using Supervised Learned Task-Based Query Similarity σ_{dt}^4

QC-wcc using σ_{dt}^4			
η	*F1*	*Rand*	*Jaccard*
0.0	0.76	0.69	0.43
0.1	0.76	0.69	0.43
0.2	0.76	0.69	0.43
0.3	0.76	0.69	0.43
0.4	0.76	0.69	0.43
0.5	0.76	0.69	0.43
0.6	0.78	0.77	0.46
0.7	0.79	0.78	0.45
0.8	**0.80**	**0.79**	**0.45**
0.9	0.80	0.79	0.42
1.0	0.71	0.70	0.13

QC-htc using σ_{dt}^4			
η	*F1*	*Rand*	*Jaccard*
0.0	0.76	0.73	0.42
0.1	0.76	0.73	0.42
0.2	0.76	0.73	0.42
0.3	0.76	0.73	0.42
0.4	0.76	0.73	0.42
0.5	0.76	0.73	0.42
0.6	0.78	0.79	0.44
0.7	**0.79**	**0.79**	**0.43**
0.8	0.79	0.79	0.42
0.9	0.78	0.78	0.38
1.0	0.68	0.69	0.10

Table XV. QC-wcc vs. QC-htc Using Supervised Learned Task-Based Query Similarities σ_{nb}^1 or σ_{nb}^3

QC-wcc using σ_{nb}^1 or σ_{nb}^3			
η	*F1*	*Rand*	*Jaccard*
0.0	0.65	0.36	**0.33**
0.1	0.65	0.36	**0.33**
0.2	0.65	0.36	**0.33**
0.3	0.65	0.36	**0.33**
0.4	0.65	0.36	**0.33**
0.5	0.65	0.36	**0.33**
0.6	0.65	0.36	**0.33**
0.7	0.65	0.37	**0.33**
0.8	0.64	0.40	0.32
0.9	0.65	0.48	0.30
1.0	**0.75**	**0.73**	0.24

QC-htc using σ_{nb}^1 or σ_{nb}^3			
η	*F1*	*Rand*	*Jaccard*
0.0	0.65	0.38	**0.33**
0.1	0.65	0.38	**0.33**
0.2	0.65	0.38	**0.33**
0.3	0.65	0.38	**0.33**
0.4	0.65	0.38	**0.33**
0.5	0.65	0.38	**0.33**
0.6	0.65	0.38	**0.33**
0.7	0.65	0.39	**0.33**
0.8	0.64	0.42	0.31
0.9	0.65	0.50	0.30
1.0	**0.75**	**0.72**	0.22

Table XIV illustrates the results obtained both with QC-wcc and QC-htc using the supervised learned similarity σ_{dt}^4. Concerning QC-wcc, the best results were provided when $\eta = 0.8$, while QC-htc performed best when $\eta = 0.7$.

Similarly, Table XV shows the results obtained both with QC-wcc and QC-htc using the supervised learned similarity σ_{nb}^1 (or, analogously, σ_{nb}^3). In both cases, best F1 and Rand values were obtained when $\eta = 1.0$, whereas best Jaccard results were obtained when $0.0 \leq \eta \leq 0.7$. However, all the validity measures are significantly worst than those obtained when using σ_{dt}^4. Another difference is that when using σ_{dt}^4, the best results are around a unique value of the threshold η, that is, $\eta = 0.8$ and $\eta = 0.7$, whereas here it appears there is a less strong relationship between the overall best results and η.

Table XVI shows the results obtained both with QC-wcc and QC-htc using the supervised learned similarity σ_{lr}^*. Both QC-wcc and QC-htc achieve their best outcomes when the threshold $\eta = 0.7$. However, even in this case, all the validity measures lose significant value with respect to QC-wcc and QC-htc when using σ_{dt}^4. With regard to σ_{dt}^4 there is a clear relationship between the best validity measures and the value of η.

Table XVII compares the best results found with each approach and highlights similar behaviors when using unsupervised or supervised learned similarities.

Finally, Table XVIII clearly points out the benefit of exploiting collaborative knowledge like Wikipedia. QC-htc used the similarity function σ_2 to capture and group together two queries that are completely different from a content-based perspective,

Table XVI. QC-WCC vs. QC-HTC Using Supervised Learned Task-Based Query Similarities σ_{lr}^{*} $(* \in \{1, 2, 3, 4\})$

QC-WCC using σ_{lr}^{*}			
η	*F1*	*Rand*	*Jaccard*
0.0	0.65	0.50	0.30
0.1	0.65	0.50	0.30
0.2	0.65	0.50	0.30
0.3	0.65	0.50	0.30
0.4	0.65	0.50	0.30
0.5	0.65	0.50	0.30
0.6	0.70	0.64	0.30
0.7	**0.77**	**0.75**	**0.31**
0.8	0.76	0.73	0.24
0.9	0.74	0.70	0.15
1.0	0.73	0.66	0.00

QC-HTC using σ_{lr}^{*}			
η	*F1*	*Rand*	*Jaccard*
0.0	0.65	0.51	0.30
0.1	0.65	0.51	0.30
0.2	0.65	0.51	0.30
0.3	0.65	0.51	0.30
0.4	0.65	0.51	0.30
0.5	0.65	0.51	0.30
0.6	0.68	0.65	0.28
0.7	**0.76**	**0.75**	**0.30**
0.8	0.75	0.73	0.24
0.9	0.74	0.70	0.14
1.0	0.73	0.66	0.00

Table XVII. Best Results Obtained with Each Method Using Both Unsupervised and Supervised Learned Similarities

		F1	*Rand*	*Jaccard*
TS-26 (*baseline*)		0.65	0.34	0.34
QFG $_{best}$ (*state of the art*)		0.77	0.71	0.40
unsupervised learned similarity σ_2	QC-MEANS $_{best}$	0.72	0.74	0.27
	QC-SCAN $_{best}$	0.77	0.71	0.19
	QC-WCC $_{best}$	**0.81**	0.78	0.44
	QC-HTC $_{best}$	0.80	0.78	0.43
supervised learned similarity σ_{dt}^{4}	QC-WCC $_{best}$	0.80	**0.79**	**0.45**
	QC-HTC $_{best}$	0.79	**0.79**	0.43

Table XVIII. The Impact of Wikipedia: σ_1 vs. σ_2

QC-HTC $_{\sigma_1}$ $(\alpha = 1)$		QC-HTC $_{\sigma_2}$ $(0.5, 4)$	
Query ID	Query String	Query ID	Query String
		63	`los cabos`
		64	`cancun`
65	`hurricane wilma`	65	`hurricane wilma`
68	`hurricane wilma`	68	`hurricane wilma`

but that are closely correlated from a the point of view of semantics. Indeed, Cancun is one of the regions affected by Hurricane Wilma which hit in 2005 (see the cross reference in the corresponding Wikipedia article[7]). Moreover, Los Cabos and Cancun are both in Mexico despite being a great distance apart. It might be the case that the user was looking for the relative position of Los Cabos from Cancun in order to understand if Los Cabos was hit by the hurricane as well.

8.3. Evaluation on a Larger Dataset

So far, we have evaluated our user task discovery methods on a manually-labeled dataset, which we referred to as our ground truth. However, an evaluation on a larger dataset may give useful hints on whether our proposed techniques are able to generalize.

In this section, we consider the two best approaches we proposed, that is, QC-WCC and QC-HTC. Both these methods were run on the public dataset *top-500-1week*, which

[7] `http://en.wikipedia.org/wiki/Cancun`

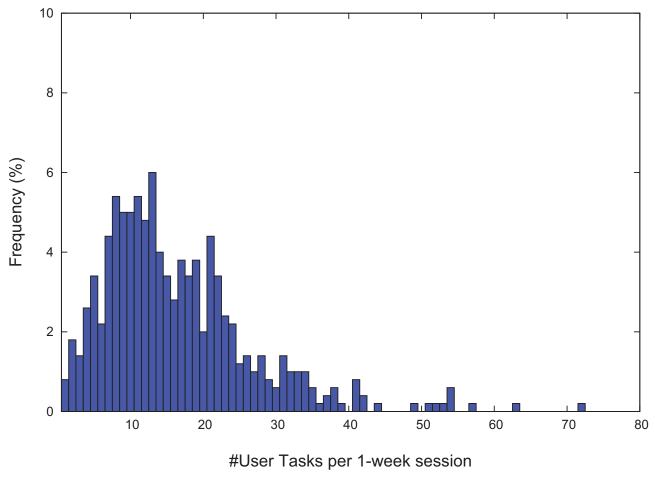


Fig. 11. The distribution of user task frequency using the QC-wcc algorithm.

refers to the 500 user sessions with the highest number of queries, yet limited to the first week of logging. It is worth noting that a subset of *top-500-1week* was used to build our ground truth (see Section 5). This dataset contains a total amount of 48,257 queries, meaning about 97 queries per week per user on average, which corresponds to nearly 14 queries per day per user, and it is available for download.[8] Here, the longest user session, that is, the session with the highest number of queries, is 1,774 queries.

8.3.1. QC-wcc. When the QC-wcc algorithm was executed on this larger dataset, a total number of 8,191 user tasks was found. In Figure 11, we plot the frequency distribution of user tasks over the user sessions contained in the dataset. The maximum number of discovered tasks for a single user session is 72. On average, each user performed 16.4 tasks per week.

Moreover, the user task size distribution (i.e., the number of queries for each discovered user task) is depicted in Figure 12. This plot shows that the user task size distribution in the larger dataset reflects the ground truth, which was reported in Figure 8. However, the actual average number of queries per user task is about 3.93, which is slightly greater than the ground truth (i.e., about 2.57).

On the basis of the analysis conducted on the ground truth and described in Section 5, we also evaluated how user tasks were distributed over time-gap sessions, namely how many user tasks were discovered within the same time-gap session, using the QC-wcc algorithm. The plot in Figure 13 shows some similarities to the one depicted in Figure 9, which instead refers to the ground truth. However, the QC-wcc algorithm discovered about 1.34 user tasks per time-gap session, as opposed to 1.80 found in the ground truth.

8.3.2. QC-htc. The QC-htc algorithm identified a total figure of 8,301 user tasks on the larger dataset. Figure 14 depicts the frequency distribution of the number of user tasks for each one-week session. Here, the maximum number of discovered user tasks for a single user session is 163, whereas the minimum is 1. This means each user performed about 16.6 tasks, on average.

As regards the QC-wcc, we also evaluated the user task size distribution using this algorithm, and the result is shown in Figure 15. Here, not only is the curve progress

[8]`http://miles.isti.cnr.it/~tolomei/downloads/aol-top500-1w.tgz`

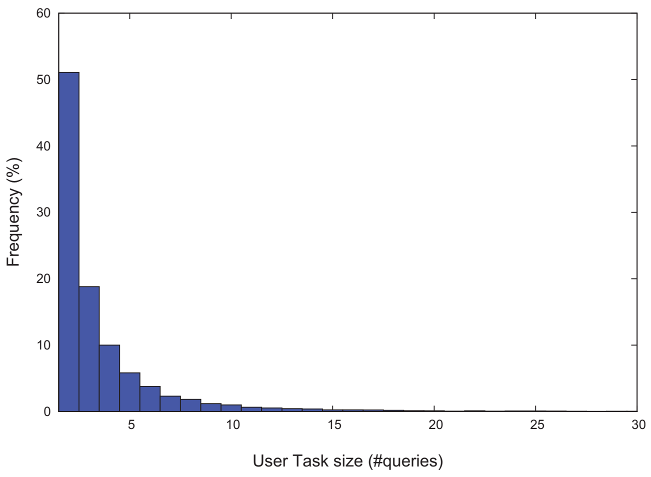


Fig. 12. The distribution of user task size using the QC-WCC algorithm.

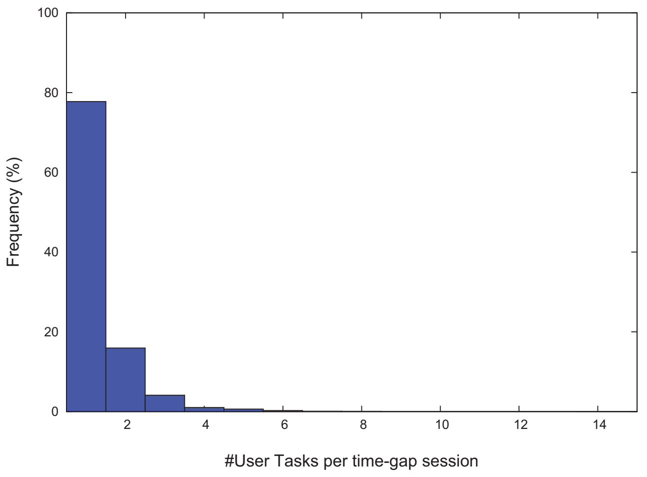


Fig. 13. The distribution of user tasks across time-gap sessions using the QC-WCC algorithm.

compliant with the user task size distribution of the ground truth, but also, the average number of queries per task (i.e., about 3.38) is closer to the one we discovered in our golden set. Interestingly, the QC-HTC was able to detect about 1.49 user tasks per time-gap session, and the whole distribution is shown in Figure 16.

9. DISCOVERING COLLECTIVE TASKS

The last major contribution to this work is a method for detecting collective tasks. Given the set $\mathcal{T}$ of user tasks extracted from the query log with one of one of the techniques previously discussed, let $t_i \in \mathcal{T}$ be a generic user task, and $\overline{t_i}$ its bag-of-words representation. More specifically, if $\overline{q}$ is the bag-of-words representation of a query $q \in \mathcal{QL}$, it follows that $\overline{t_i} = \biguplus_{q \in t_i} \overline{q}$, where $\biguplus$ is the bag union operator. Therefore, each $\overline{t_i}$ can be considered as a *text document*, and the problem of discovering the collective tasks can be reduced to the clustering of similar text documents [Zhao and Karypis 2002, 2004].

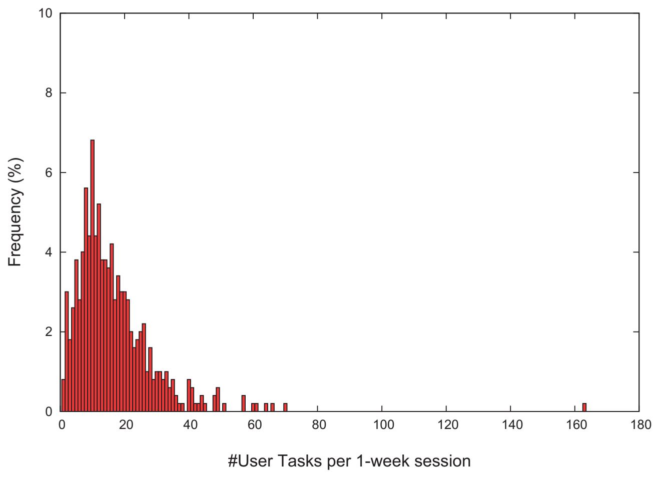


Fig. 14. The distribution of user task frequency using the QC-HTC algorithm.

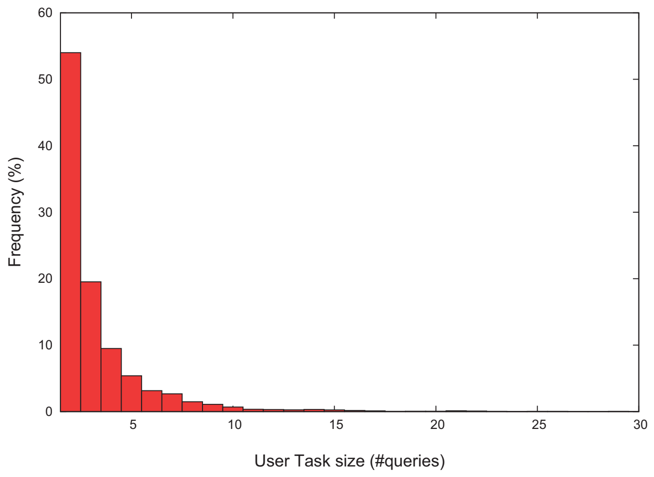


Fig. 15. The distribution of user task size using the QC-HTC algorithm.

In the rest of this section, we first present a manually-generated ground truth of collective tasks, which is used to evaluate the quality of the collective tasks extracted by means of a user task clustering algorithm. Then, we discuss a set of possible user task clustering algorithms and their evaluation.

9.1. Ground Truth of Collective Tasks

The user tasks that were manually annotated to create the ground truth of collective tasks were identified by running QC-HTC on the same portion of the *top-500-1week* query log, which had previously been used to generate the ground truth of user tasks (see Section 5).

QC-HTC discovered a total amount of 318 user tasks, which in turn were manually grouped into a set of collective tasks using the same annotators employed in constructing the ground truth of user tasks. The annotators discarded 16 user tasks, since no agreement was reached on the cluster assignments for these user tasks. They grouped

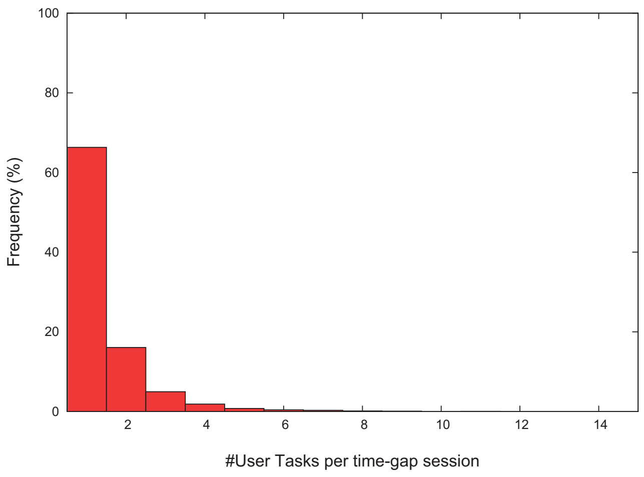


Fig. 16. The distribution of user tasks across time-gap sessions using the QC-HTC algorithm.

Table XIX. Statistical Indicators on Manually-Identified Collective Tasks

Cluster Size				
Avg.	*Std. Dev.*	*Max*	*Min*	*Median*
5.70	13.27	61	1	5

the remaining 302 (i.e., ≈95% of the total) into 53 collective tasks, each one containing on average 5.70 user tasks. Table XIX shows some statistics relating to the size of the clusters which had been manually generated.

9.2. Clustering Algorithms

In order to automatically discover collective tasks, we propose clustering the set of already detected user tasks. In particular, in order to cluster the set $\mathcal{T}$ of user tasks, we selected a set of algorithms included in the CLUTO[9] toolkit. Each algorithm produces a set of K clusters, namely a set of K collective tasks.

Regardless of the clustering algorithm chosen, three input parameters have to be provided: (i) a similarity measure, (ii) an objective function, and (iii) a number K of clusters. For the first option, we adopted two measures, namely the well-known *cosine similarity* and the *Pearson's correlation coefficient*. Concerning the second, we chose to maximize the intra-cluster similarity according to the following function:

$$\max \sum_{i=1}^{K} \sqrt{\sum_{u,v \in S_i} sim(u, v)},$$

where K is the total number of produced clusters, S_i is the set of objects assigned to the ith cluster, and $sim(u, v)$ is the similarity between the two objects $u, v \in S_i$ (i.e., either cosine or Pearson's correlation coefficient).

Method 1: Repeated Bisections (rbr). This is the first clustering approach we used, where the desired K-way clustering solution is computed by performing a sequence of $K - 1$ *repeated bisections*. Here, the similarity matrix is first clustered into two groups, then one of these groups is selected and bisected further. This process continues until

[9]`http://glaros.dtc.umn.edu/gkhome/views/cluto`

the desired number of clusters is found. During each step, the cluster is bisected so that the resulting two-way clustering solution optimizes the chosen criterion function. The cluster that is selected for further partitioning is customizable, and by default, it coincides with the biggest cluster at each stage. Note that this approach ensures that the criterion function is locally optimized within each bisection, but in general, it is not globally optimized. Therefore, we selected a variant of this method which, in the end, globally optimizes the objective function.

Method 2: Agglomerative (agg). In this approach, the desired K-way clustering solution is computed using the *agglomerative* paradigm whose goal is again to locally optimize the selected clustering objective function. The solution is obtained by stopping the agglomeration process when K clusters are left.

Eventually, we came up with four solutions by mixing the two preceding clustering methods with the two similarity scores, namely *rbr-cosine*, *rbr-pearson*, *agg-cosine*, and *agg-pearson*.

9.3. Evaluation on the Ground Truth

All the automatic solutions just described were run on the same set of user tasks we used to manually build the ground truth of collective tasks (see Section 9.1). In order to evaluate our clustering algorithms, we set the final number of clusters as $K = 53$, which is the exact number of collective tasks discovered by the human assessors.

Similarly to Section 8.1, we refer to classification-oriented measures of validity in order to assess the performance of the various clustering methods in relation to the collective tasks in the ground truth, namely precision, recall, and F1. Table XX reports these measurements of clustering validity for the various algorithms, along with some statistical indicators.

(a) *rbr-cosine.* This solution produces the set of $K = 53$ output clusters by performing a sequence of $K - 1$ repeated bisections. Furthermore, it uses the cosine similarity to compare the textual representations of any two user tasks. From the original set of 318 user tasks, 297 were clustered (i.e., $\approx$93%), whereas 21 were discarded. On average, each collective task contains about 5.60 user tasks, which is close to the value obtained from the ground truth.

(b) *rbr-pearson.* As with the previous method, *rbr-pearson* produces the final set of output clusters by performing a sequence of repeated bisections. However, it uses the Pearson's correlation to measure the similarity between pairs of user tasks. From the original set of 318 user tasks, 293 were clustered (i.e., $\approx$92%), whereas 25 were discarded.

(c) *agg-cosine.* This solution agglomerates user tasks by locally optimizing the selected criterion function, which is based on the cosine similarity between textual representations of any two user tasks. All 318 original user tasks were clustered, thereby each collective task has six user tasks on average. On the basis of the quality of produced clusters, this method results in a significant drop in precision, recall, and F1 scores compared to the preceding partitional methods.

(d) *agg-pearson.* Similarly to the preceding method , *agg-pearson* agglomerates user tasks. This solution uses Pearson's correlation to measure the similarity between user tasks. As with *agg-cosine*, all 318 original user tasks were clustered, thereby each collective task has six user tasks on average. Again, the overall quality of clustering is worse than that obtained with partitional methods.

Table XX. Statistical Indicators and Quality Evaluation of Each User Task Clustering Algorithm

	Cluster Size				
	Avg.	*Std. Dev.*	*Max*	*Min*	*Median*
rbr-cosine	5.60	14.19	89	2	3
rbr-pearson	5.53	14.41	80	3	5
agg-cosine	6.00	16.97	97	1	3
agg-pearson	6.00	43.86	250	1	1

	Cluster Quality		
	Precision	*Recall*	*F1*
rbr-cosine	**0.71**	**0.48**	**0.57**
rbr-pearson	0.68	0.46	0.55
agg-cosine	0.59	0.42	0.49
agg-pearson	0.54	0.39	0.45

Table XXI. Statistical Indicators on User Task Clustering Using *rbr-cosine*

Cluster Size				
Avg.	*Std. Dev.*	*Max*	*Min*	*Median*
6.81	15.43	479	1	5

Intra-Cluster Similarity		
Avg.	*Std. Dev.*	*Median*
0.59	0.17	0.56

Finally, from the results reported in Table XX, the best clustering algorithm is partitional (i.e., top-down), that is, *rbr-cosine*, which achieves the highest values of precision, recall, and F1, respectively.

9.4. Evaluation on a Larger Dataset

In this section, we assess the behavior of the best-performing clustering algorithm (i.e., *rbr-cosine*) when applied to a larger collection of tasks. These user tasks were extracted from the whole *top-500-1week* dataset (see Section 8.3.2) by the QC-HTC algorithm.

Unlike the previous tests, which were conducted to find the best algorithm, in this case we do not have an a priori knowledge of the number of collective tasks in the whole dataset. In order to select the number K of clusters, we thus observed the behavior of the objective function by varying K. It was noted as K increases, the objective function monotonically increases as well. Indeed, the maximum intra-cluster similarity is obtained when K is equal to the number of documents to be clustered (i.e., when each cluster contains exactly one document). It is clear that we need to find a trade-off, and this is indicated by a well-established empirical criterion, also known as the *elbow method*. Generally speaking, we chose $K = \bar{K}$ such that for any K, $K > \bar{K}$ the slope of our objective function appeared to increase less than for any K, $K < \bar{K}$. The reason for selecting this method is to choose a number of clusters such that adding another cluster does not give a much better model for fitting the data. By following this method on the original collection of 8,301 user tasks, we eventually obtained a set of $K = 1{,}024$ collective tasks.

Since we do not have a ground truth for such a large dataset, we first show some statistics relating to the obtained clusters, for example, the number of user tasks within each collective task, the intra-cluster similarity, etc. Furthermore, we present some analysis on the *popularity* of collective tasks, namely we show how collective tasks are actually distributed across the original user sessions stored on the query log. Finally, we illustrate some examples as anecdotical evidence.

From the initial input set of 8,301 user tasks, 6,970 ($\approx$84%) were actually clustered. Table XXI shows some statistical indicators on the output clusters of user tasks. In the left-hand table, we indicate the variety in cluster size, that is, the number of user tasks contained within a collective task. We found that the collective task with the highest number contained 479 user tasks. By manually inspecting this large collective task, we found it mainly contains *navigational* user tasks [Broder 2002], mostly related to contents of a sexual nature. On average, each collective task included approximately *seven* user tasks. In Figure 17, we also plot the cluster size distribution yet limited to collective tasks of less than 50 user tasks. Indeed, it is worth noting that the vast

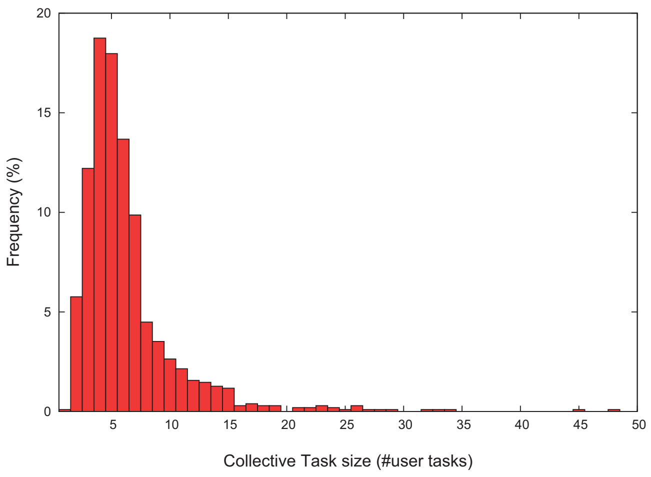


Fig. 17. The distribution of collective task size by means of the number of composing user tasks.

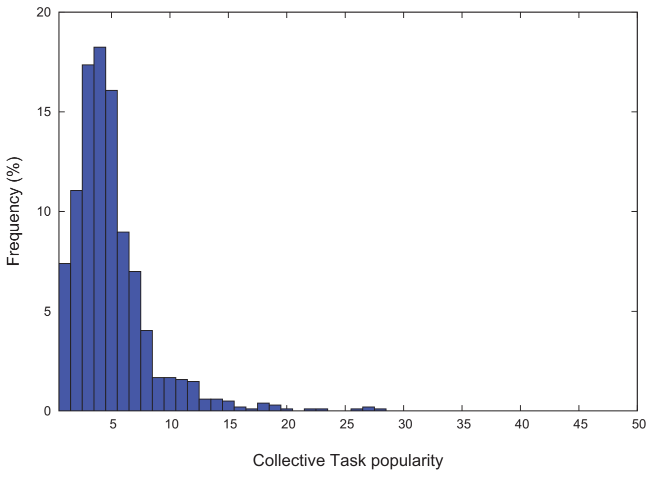


Fig. 18. The distribution of collective task popularity across the original set of user sessions.

majority of collective tasks (i.e., 99.9%) contained less than 50 user tasks. The right-hand table shows some indicators of intra-cluster similarity.

In addition, we were interested in checking if some collective tasks occurred more frequently in the query log, both within the same user session and across distinct user sessions. To this end, we rewrote each original user session as a set of user tasks (i.e., actually collective tasks), instead of a sequence. Therefore, for each collective task, we computed the percentage of user sessions where this appeared, disregarding its order and any possible repetition within a single user session. According to this study, the top-most popular collective task occurred in 183 out of 500 user sessions (i.e., about 36.6%). In contrast, the vast majority of collective tasks (i.e., about 93.5%) appeared in less than 11 user sessions, and a collective task occurred in about five user sessions, on average. In Figure 18, we show the popularity distribution of collective tasks across user sessions.

Table XXII. A Collective Task Containing User Tasks Referring to *hobby/gardening*

Collective Task # 314	
User Task IDs	Queries
2439668-18-1	`cottage garden qvc`, `cottage garden roses`
1188448-3-7	`private hot tub garden calistoga area lodging`
1012899-3-2	`vegetable garden`, `vegetable garden ideas`
2061454-23-1	`japanese garden`, `decor japanese garden`, ...
...	...
679436-11-2	`tv garden shows`, `rebecca cole garden show tv`, ...
297468-3-2	`dry garden`, `dry garden berkeley`, ...
297468-10-1	`california garden blog`, `garden blog`, `best garden blogs`
297468-21-1	`open garden`, `open garden day sacramento`
297468-26-1	`horton farm iris garden`

Table XXIII. A Collective Task Containing User Tasks Referring to *History of Rome*

Collective Task # 578	
User Task IDs	Queries
12472900-4-1	`louis xvi descended clovis`, `descendants roman nobility`, ...
8566671-21-3	`roman history`, ...
4110454-5-4	`roman claim conquest`, `roman historian ivy` ...

Table XXIV. A Collective Task Containing User Tasks Referring to *medical diseases*

Collective Task # 693	
User Task IDs	Queries
57424-1-1	`california sweats company`, `low sugar sweetener`, ...
1524276-2-1	`hypoglycemia periods`, `low blood sugar periods menstrual cycle`, ...
257689-1-1	`blood sugar 500`, `blood sugar chart`
543587-32-3	`prolonged periods perimenopausal`, ...
4401012-9-1	`high blood sugar use diuretics`, `blood sugar levels fasting`, ...

Table XXV. A Collective Task Containing User Tasks Referring to *math/physics*

Collective Task # 946	
User Task IDs	Queries
292860-6-1	`calculate moment rotational inertia`, `kinematic equations`
349670-16-2	`entropy equations`, `entropy`
1411796-16-1	`schroedinger's equation`, `schrodinger's equation`

Finally, Tables XXII, XXIII, XXIV, and XXV show some examples of anecdotical evidence within the collective tasks found. In particular, we show the user tasks[10] of four collective tasks—discovered by our *rbr-cosine* clustering—which refer to four real-life situations.

10. CONCLUSIONS

This work addresses some important research challenges in developing next-generation Web search engines which better satisfy user needs. We claim that people increasingly phrase queries to search engines in order to find information which can simplify their daily tasks. Examples of these tasks include *finding a recipe*, *booking a flight*, *reading online news*, etc. To verify this theory and to discover those tasks, we carried out a detailed analysis of the historical data recorded in long-term search engine query

[10]User task IDs are uniquely determined by the following pattern: `userID-sessionID-taskID`.

logs. Our approach involves a two-step methodology. First, we identify user tasks from individual user sessions stored in the query log. In our vision, a user task is a set of possibly noncontiguous queries occurring within a search session which relates to the same need. Then, as a second step, we discover collective tasks by aggregating similar user tasks, possibly performed by distinct users.

For the initial step, we define the user task discovery problem (UTDP) as the problem of finding the best partitioning of a set of queries into subsets of queries related to the same user task. The UTDP involves two main issues: (i) it requires a robust measure to evaluate the task relatedness between any two queries, and (ii) it needs an effective method in order to discover user tasks on the basis of this measure. With reference to (i), we propose both unsupervised and supervised learning approaches for devising several task-based query similarities, whereas we tackle (ii) by introducing a set of query clustering methods specifically designed to discover user tasks.

We evaluate all the proposed solutions by means of a manually-built ground truth, namely a task-oriented partitioning of the queries in our benchmarking dataset performed by human annotators. In particular, two of the proposed clustering methods, that is, QC-WCC and QC-HTC, have been shown to outperform state-of-the-art solutions.

For the second stage, we introduce and investigate the problem of discovering collective tasks. To this end, we propose four methods for clustering previously mined user tasks, which are represented by the bag-of-words extracted from the associated queries.

We evaluate all these solutions both on a manually-built ground truth and on a larger dataset. The experiments conducted reveal that our two-step approach can effectively detect similar latent needs from a query log by first mining the search behavior of each single user, and then by aggregating the similar user tasks performed by different users.

As future work, we plan to exploit the collective tasks mined from the query log to build a model for representing the task-by-task search behavior of users. This model could subsequently be used to devise novel applications like a task recommender system that goes beyond query suggestion mechanisms currently offered by modern Web search engines.

ACKNOWLEDGMENTS

We acknowledge the authors of Boldi et al. [2008] and the Yahoo! Research Labs in Barcelona, Spain, for providing us their query-flow graph implementation, and Franco Maria Nardini for adapting this implementation to our needs.

REFERENCES

BAEZA-YATES, R., GIONIS, A., JUNQUEIRA, F. P., MURDOCK, V., PLACHOURAS, V., AND SILVESTRI, F. 2008. Design trade-offs for search engine caching. *ACM Trans. Web 2*, 4, 1–28.

BAEZA-YATES, R. AND RIBEIRO-NETO, B. 1999. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

BEEFERMAN, D. AND BERGER, A. 2000. Agglomerative clustering of a search engine query log. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*. ACM, New York, NY. 407–416.

BOLDI, P., BONCHI, F., CASTILLO, C., DONATO, D., GIONIS, A., AND VIGNA, S. 2008. The query-flow graph: Model and applications. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management (CIKM'08)*. ACM, New York, NY, 609–618.

BRODER, A. 2002. A taxonomy of Web search. *SIGIR Forum 36*, 2, 2, 3–10.

CAO, H., JIANG, D., PEI, J., HE, Q., LIAO, Z., CHEN, E., AND LI, H. 2008. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, New York, NY, 875–883.

DONATO, D., BONCHI, F., CHI, T., AND MAAREK, Y. 2010. Do you want to take notes?: Identifying research missions in Yahoo! Search Pad. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. ACM, New York, NY, 321–330.

ESTER, M., KRIEGEL, H. P., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'96)*. ACM, New York, NY, 226–231.

FU, L., GOH, D. H.-L., FOO, S. S.-B., AND NA, J.-C. 2003. Collaborative querying through a hybrid query clustering approach. In *Proceedings of the 6th International Conference on Asian Digital Libraries (ICADL'03)*. Lecture Notes in Computer Science, vol. 2911, Springer-Verlag, Berlin Heidelberg, 111–122.

GABRILOVICH, E. AND MARKOVITCH, S. 2007. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. 6–12.

GAYO-AVELLO, D. 2009. A survey on session detection methods in query logs and a proposal for future evaluation. *Info. Sci. 179*, 12, 1822–1843.

GLANCE, N. S. 2001. Community search assistant. In *Proceedings of the 6th ACM International Conference on Intelligent User Interfaces (IUI'01)*. ACM, New York, NY, 91–96.

GUO, J., CHENG, X., XU, G., AND ZHU, X. 2011. Intent-aware query similarity. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*. ACM, New York, NY, 259–268.

HE, D. AND GÖKER, A. 2000. Detecting session boundaries from Web user logs. In *Proceedings of the 22nd Annual Colloquium on Information Retrieval Research (BCS-IRSG)*. 57–66.

HE, D., GÖKER, A., AND HARPER, D. J. 2002. Combining evidence for automatic web session identification. *Info. Process. Manage.* 38, 5, 727–742.

HOPCROFT, J. AND TARJAN, R. 1973. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM 16*, 6, 372–378.

JANSEN, B. J. AND SPINK, A. 2006. How are we searching the world wide Web?: A comparison of nine search engine transaction logs. *Info. Process. Manage. 42*, 1, 248–263.

JANSEN, B. J., SPINK, A., BATEMAN, J., AND SARACEVIC, T. 1998. Real life information retrieval: A study of user queries on the web. *SIGIR Forum 32*, 1, 5–17.

JANSEN, B. J., SPINK, A., BLAKELY, C., AND KOSHMAN, S. 2007. Defining a session on Web search engines: Research articles. *J. Amer. Soci. Info. Scie. Technol. 58*, 6, 862–871.

JÄRVELIN, A., JÄRVELIN, A., AND JÄRVELIN, K. 2007. s-grams: Defining generalized n-grams for information retrieval. *Info. Process. Manage. 43*, 4, 1005–1019.

JONES, R. AND KLINKNER, K. L. 2008. Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management (CIKM'08)*. ACM, New York, NY, 699–708.

KOTOV, A., BENNETT, P. N., WHITE, R. W., DUMAIS, S. T., AND Teevan, J. 2011. Modeling and analysis of cross-session search tasks. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. ACM, New York, NY, 5–14.

LAU, T. AND HORVITZ, E. 1999. Patterns of search: Analyzing and modeling Web query refinement. In *Proceedings of the 7th International Conference on User Modeling*. Springer-Verlag, Berlin, 119–128.

LEACOCK, C. AND CHODOROW, M. 1998. *Combining Local Context and WordNet Similarity for Word Sense Identification*. The MIT Press, Cambridge, MA, 11, 265–283.

LEE, U., LIU, Z., AND CHO, J. 2005. Automatic identification of user goals in Web search. In *Proceedings of the 14th International World Wide Web Conference (WWW'05)*. ACM, New York, NY, 391–400.

LESK, M. 1986. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th ACM International Conference on Systems Documentation (SIGDOC'86)*. ACM, New York, NY, 24–26.

LEUNG, K. W. T., NG, W., AND LEE, D. L. 2008. Personalized concept-based clustering of search engine queries. *IEEE Trans. Knowl. Data Engi. 20*, 11, 1505–1518.

LUCCHESE, C., ORLANDO, S., PEREGO, R., SILVESTRI, F., AND TOLOMEI, G. 2011. Identifying task-based sessions in search engine query logs. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*. ACM, New York, NY, 277–286.

MACQUEEN, J. B. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, L. M. Le Cam and J. Neyman* Eds., Vol. 1. University of California Press, Berkeley, CA, 281–297.

MEI, Q., KLINKNER, K., KUMAR, R., AND TOMKINS, A. 2009. An analysis framework for search sequences. In *Proceeding of the 18th Conference on Information and Knowledge Management (CIKM'09)*. ACM, New York, NY, 1991–1994.

MILNE, D. AND WITTEN, I. H. 2008. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI'08)*. AAAI Press, Menlo Park, CA, 25–30.

OZMUTLU, H. C. AND ÇAVDUR, F. 2005. Application of automatic topic identification on excite web search engine data logs. *Info. Process. Manage. 41*, 5, 1243–1262.

PORTER, M. F. 1980. *An Algorithm for Suffix Stripping* Vol. 14. Morgan Kaufmann Publishers, San Francisco, CA, 130–137.

QUINLAN, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Francisco, CA.

RADA, R., MILI, H., BICKNELL, E., AND BLETTNER, M. 1989. Development and application of a metric on semantic nets. *IEEE Trans. Syst. Man Cybernet. 19*, 1, 17–30.

RADLINSKI, F. AND JOACHIMS, T. 2005. Query chains: Learning to rank from implicit feedback. In *Proceedings of the KDD Cup Workshop at the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*. ACM, New York, NY, 239–248.

RAGHAVAN, V. V. AND SEVER, H. 1995. On the reuse of past optimal queries. In *Proceedings of the 18th ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR'95)*. ACM, New York, NY, 344–350.

REED, W. 2001. The Pareto, zipf and other power laws. *Econ. Lett. 74*, 1, 15–19.

RESNIK, P. 1995. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*. 448–453.

RICHARDSON, M. 2008. Learning about the world through long-term query logs. *ACM Trans. Web 2*, 4, 1–27.

ROSE, D. E. AND LEVINSON, D. 2004. Understanding user goals in web search. In *Proceedings of the 13th International World Wide Web Conference (WWW'04)*. ACM, New York, NY, 13–19.

SALTON, G. AND MCGILL, M. J. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY.

SECO, N. AND CARDOSO, N. 2006. Detecting user sessions in the tumba! web log. Tech. rep. Faculdade de Ciências da Universidade de Lisboa.

SHEN, X., TAN, B., AND ZHAI, C. 2005. Implicit user modeling for personalized search. In *Proceeding of the 14th Conference on Information and Knowledge Management (CIKM'05)*. ACM, New York, NY, 824–831.

SHI, X. AND YANG, C. C. 2006. Mining related queries from search engine query logs. In *Proceedings of the 15th International World Wide Web Conference (WWW'06)*. ACM, New York, NY, 943–944.

SILVERSTEIN, C., MARAIS, H., HENZINGER, M., AND MORICZ, M. 1999. Analysis of a very large Web search engine query log. *SIGIR Forum 33*, 1, 6–12.

SILVESTRI, F. 2010. Mining Query Logs: Turning search usage data into knowledge. *Found. Trends Info. Ret. 1*, 1–2, 1–174.

SILVESTRI, F., BARAGLIA, R., LUCCHESE, C., ORLANDO, S., AND PEREGO, R. 2008. (Query) history teaches everything, including the future. In *Proceedings of the 6th Latin American Web Congress (LA-WEB'08)*. IEEE Computer Society, Washington, DC, 12–22.

SPINK, A., PARK, M., JANSEN, B. J., AND PEDERSEN, J. 2006. Multitasking during Web search sessions. *Info. Process. Manage. 42*, 1, 264–275.

TAN, P. N., STEINBACH, M., AND KUMAR, V. 2005. *Introduction to Data Mining*. Addison-Wesley, Boston, MA.

WEN, J. R., NIE, J. Y., AND ZHANG, H. 2002. Query clustering using user logs. *ACM Trans. Info. Syst. 20*, 1, 59–81.

ZHAO, Y. AND KARYPIS, G. 2002. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceeding of the 11th Conference on Information and Knowledge Management (CIKM'02)*. ACM, New York, NY, 515–524.

ZHAO, Y. AND KARYPIS, G. 2004. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learn. 55*, 3, 311–331.

Received May 2011; revised June, November 2012, March 2013; accepted March 2013