A Dataset for Evaluating Query Suggestion Algorithms in Information Retrieval

Ioan Bădărînză
Department of Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania
ionutb@cs.ubbcluj.ro

Adrian Sterca
Department of Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania
forest@cs.ubbcluj.ro

Darius Bufnea
Department of Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania
bufny@cs.ubbcluj.ro

***Abstract*—This paper presents a dataset that can be used for evaluating query suggestion algorithms in textual information retrieval. The dataset is public and offered free of charge to the information retrieval research community. The data was gathered in an experiment that lasted more than 2 months and to which participated a number of 119 users, mainly faculty students. The dataset contains web browsing history and query history (submitted to the Google search engine) from all these users. The data is indexed in a database and downloadable in a database dump format. The dataset is very useful for evaluating general query suggestion algorithms by themselves (in a standalone manner) or against Google's MPC query suggestion algorithm. At the same time, the dataset supports building and testing personalized query suggestion algorithms that consider the user context/profile when computing query suggestions.**

***Index Terms*—dataset, query suggestion, information retrieval, search engine**

I. Introduction

Retrieving information from the web can be very difficult for a user sometimes. This is because, depending on the specific information need, users may not know what terms to enter in the search input of a search engine to better describe their information needs. In [1], [2] we can see that most of the search queries are very short, one or two words on average and in [3], [4] we can see that these words are ambiguous. In order to help the user when performing a search, most search engines like Google, Yahoo!, Bing and others, use techniques like query auto-completion and query suggestions. In almost all search engines we can see how, after we start typing letters, the system automatically tries to predict what we actually want to type. This mechanism is called *query auto-completion* [5]. The way it works is, whenever the user types a character in the search input, the system takes that prefix (also known as *subquery*) and matches it gainst other queries that were previously used in the search engine, and offers a complete list of alternative queries that might help the user to better express his information need. The query suggestions list is a list that contains from eight to ten words (or groups of words), which are usually prefixed with the subquery that the user is typing, items that are extracted from a huge log of queries submitted by all users. A very well known technique of extracting suggestions from a common query log is called Most Popular Completion (MPC) [6]. Query suggestion and query auto-completion are very similar. The main scope of both of them is to save user keystrokes when performing a search. A query suggestion is an enhanced, proposed query that the user might be looking for, whereas an auto-completion is the possible query term that the user might want to type immediately after he/she started typing the first letter.

Query suggestions algorithms are very different than search and retrieve algorithms, although they are both implemented by the search engine. The search and retrieve algorithm gets a query as input parameter and based on various TF (term frequency) / IDF (inverse document frequency) formulas, it retrieves from the inverted index, a list of documents, usually 10, most relevant to the input query [7]. On the other had, query suggestions algorithms have as input a query prefix (i.e. a subquery) and based on various forms of user logs (i.e. user history collected by the search engine server or by the browser), it returns a list of query suggestions that can be used by the user, meaning the user can pick one of the query suggestions and submit it to the search engine server where the search and retrieve algorithm is ran. Therefore, the execution of the query suggestions algorithm precedes the execution of the search and retrieve algorithm, in a search session. In the current paper we only deal with query suggestions algorithms.

When evaluating algorithms for textual information retrieval, there are two possible choices for an evaluation dataset: a) private datasets like the ones taken from Google or Bing server logs which are not available to researches that do not have a collaboration with those companies and b) datasets produced by organizations like NIST which are available to researchers at a cost within a conference (like the TREC conferences). The datasets that are available free of charge are very few and usually they are very small (e.g. datasets provided by the Kaggle platform, see Section II). But to the best of our knowledge, there are no datasets available, with a fee or no fee, for evaluating query suggestions algorithms for information retrieval. All the datasets that are available are for evaluating the information retrieval algorithm itself (i.e. the search and retrieve algorithm), not the algorithm that provides query suggestions. The contribution of this paper is a dataset useful for evaluating query suggestions algorithms in information retrieval. The dataset contains data collected

over a period of more than 2 months in an experiment with 119 volunteers. It is provided free of charge to the research community. The paper details the structure of the dataset, the performed experiment for collecting the data and ways this dataset can be used.

The rest of this paper is organized as follows. Section II presents other data collections available for the research community involved in the Information Retrieval field, discussing the advantages and disadvantages. Section III presents our data collecting methodology, also describing the architecture of the browser plug-in developed for this, while section IV contains a detailed view of how our dataset is organized. In section V, we present several ways in which our dataset can be used for evaluating a query suggestions algorithm in information retrieval. The paper ends with conclusions and acknowledgments.

II. Related work

There are some datasets available for the Web Information Retrieval scientific community. Most of them contain crawled documents, either indexed or just the raw documents pre-processed, ready to be indexed. Some of them also contain reference search topics and queries together with the ranking of the documents in the dataset with respect to a particular query (ranking performed manually by human operators). Some rather old datasets are the Reuters dataset [8] from 2000-2005 and the AOL dataset [9] from 2006. The Reuters dataset comes in 2 volumes, RCV1 and RCV2, and contains news stories from Reuters, a text and television news agency. Each story/document is manually assigned to a category. RCV1 contains approximately 810,000 documents and RCV2 contains over 487,000 documents. The AOL dataset consists of approximately 17 millions queries submitted by 657,426 unique users to the AOL search engine.

The most well known datasets for testing textual retrieval algorithms are the ones provided for TREC tracks [10]. There are many datasets available there, many of them being used for domain-specific search (like medical search, legal search etc.), but there are also datasets for testing general textual retrieval algorithms. In order to get access to these datasets, a fee is usually required. Some of the most used ones are the ClueWeb09 and ClueWeb12 datasets. The ClueWeb09 dataset [11] is available from 2009 and contains 1,040,809,705 web pages, in 10 languages, but there is also a simplified, TREC-B sample collection of this data containing approximately 50 million web pages. The ClueWeb12 dataset [12] is more recent, from 2012, containing 733,019,372 English web pages and there is also a TREC-B version of this dataset containing only 50 million documents. Both ClueWeb datasets do not contain topics, queries and reference ranking/judgments for these queries, but the TREC conference provides each year a list of search topics and queries together with a reference ranking of documents for these queries.

There are also free datasets available from the Kaggle platform [13], but they are not very specific for textual information retrieval tasks, nor they contain queries with judgments about relevance of a document to a query.

The aforementioned datasets can be used, together with a set of queries and accompanying document relevance judgments (like the ones provided within TREC conferences), for evaluating the efficiency of textual retrieval algorithms. One would index all the data in the collection, run the given set of queries through their own search engine and compare the returned results against the reference document relevance judgments accompanying the data collection. Our own dataset differs from the previous ones because it is focused not on general textual information retrieval algorithms, but on query suggestion algorithms for textual information retrieval. In addition, we provide it free of charge. One can use our dataset in order to assess the efficiency of a query suggestion algorithm for information retrieval against the standard MPC-based (i.e. Most Popular Completion [6]) query suggestion algorithm provided with the Google search engine. Although our dataset focuses on personalized query suggestion algorithms (algorithms that consider the previous user browsing history or query history when computing query suggestions), other types of query suggestion algorithms (e.g. MPC-based ones) can be evaluated using our dataset. More usage details are provided in Section V. To the best of our knowledge, there is no such dataset publicly available for evaluating query suggestion algorithms.

III. Collecting the data. The experiment

This section describes the experiment we have run and the tools we have developed in order to collect the data and build the dataset.

We have collected the data using a Chrome extension that we built. The reason why we have chosen a Chrome extension is the fact that according to different stats counters like [14], [15], Chrome has around 63% of the entire browser market share worldwide. The name of the extension is *User History Collector* and it can be installed from Chrome Web Store. After installing this extension, it will scan and parse all the web pages that the user visits, and sends the web page raw data to a server where it is stored in a database. A very important feature of this extension is the fact that it allows the user to specify which web pages he/she doesn't want to be processed. This will allow the user to protect his/her own privacy and not share personal and private data, like emails, messages, bank accounts etc. Besides the *browsing data* that the extension collects, it also collects all the queries that the users submits to Google search engine, alongside with the sub-queries and query suggestions that Google returns during the search process. We have chosen to save only Google queries because, according to different statistic tools [16], [17], [18], the Google search engine has a world wide market share of around 88%.

The entire extension was written in JavaScript and makes REST requests to a server written in Java, which stores all the data in a MySql database. Currently, the server is implemented on a single machine using threads, but for future work we want

to make it distributed using MPI [19]. In Fig. 1 we can see all the components of the extension and how the data flows from one component to another.

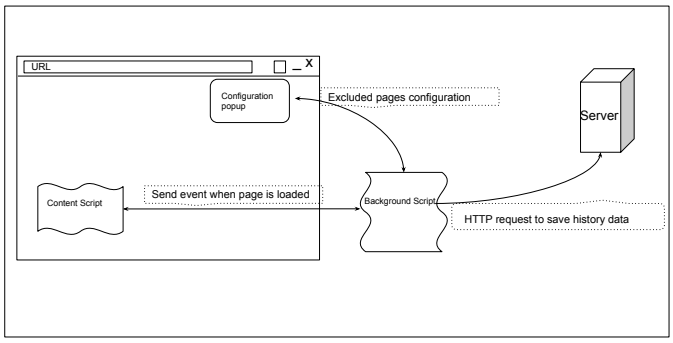


Fig. 1: User History Collector components diagram

The *background script* is the main component of the extension because it is responsible with parsing the content of visited web pages and making requests to the server in order to save the data, and it also contains information about user's preferences, like the client UUID, which is generated once when the user installs the extension, and all the web pages that the user has chosen to be ignored (i.e. not indexed) by the extension. The *configuration popup* is the user interface of the extension, where the user can manage the list of the web pages that are excluded from processing. The *content script* is the component that is injected in each web page, and waits for the page to load, checks whether the page is the Google search engine page or if it is just a normal web page, and does the following:

1) If the URL of the page does not start with *"www.google."*, it will interpret it as a new web page that was viewed and will extract the actual text from the HTML document and, alongside with page URL and page title, it will pass it to the *background script*. The *background script* will split the text in terms, will eliminate stop words and will calculate the term frequency for each unique word. After this step, it will make an Ajax HTTP request to a server which will store all the history data for later analysis.
2) If the URL of the page, does match *"www.google."*, it means the user is trying to perform a Google search. In this case:
    a) For each key pressed in Google's search input, the *content script* will extract the value of the search input and the list of suggestions provided by Google for the written subquery. This information is passed to the *background script* which will send it to the server.
    b) When the user finishes to type the desired query and submits it to Google, that particular query is passed to the *background script* which will send it to the server.

After publishing the extension to Chrome Web Store, we have asked a group of students to install the extension and continue to use the Chrome browser as they would normally do, so that we can collect data about the real user's behavior and perform offline analysis on it. We do not store any kind of personal information of the user that installed the extension and the user is at all time in control of his/her data that gets indexed (i.e. through the mechanism of excluding web sites from indexing, mechanism that is provided by the Chrome extension). All students participating in our study were volunteers. Once installed, the extension generates a UUID, which will further be used to map all the collected data to a particular user. We have started this experiment right after we published the first version of the extension, on 1st of April 2019 and had it on going for a period of 2 months. During this period of time, there were 119 unique users that had the extension installed and participated to the experiment.

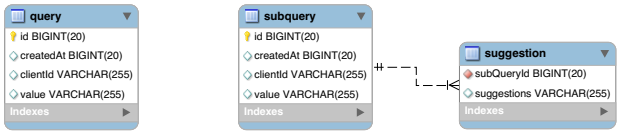
IV. The Structure of the Dataset

At the end of the 2-months experiment, the size of the MySQL database dump file grew up to 4.25GB. Our dataset is provided in the form of this MySQL database dump. The structure of the dataset (i.e. the structure of the database) is described and analyzed in the current section.

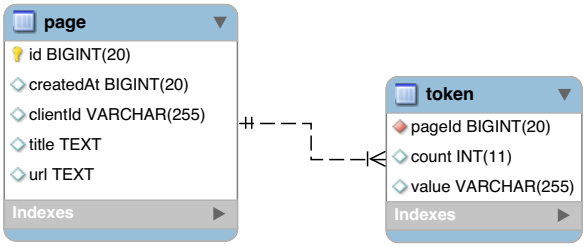
In Fig. 2 we can see the number of unique tokens from the dataset, tokens belonging to the web pages that the users have visited.

Total number of unique tokes	Total database dump file size
4770003	4.25 GB

Fig. 2: The dataset's database size



(a) Query related tables diagram



(b) Page history tables diagram

Fig. 3: Tables diagrams of our dataset's database

The table diagrams of the database are presented in Fig. 3a and Fig. 3b. First, in Fig. 3a we can see that queries made by users are stored in the *query* table, sub-queries (i.e. prefix

Time period	Total number of users	Total number of queries	Total number of collected web pages	Total number of sub-queries	Total number of query suggestions
2 months	119	54395	366417	15175	101499

Fig. 4: Dataset Dimensions

of a query or a partial query) are stored in the *subquery* table and query suggestions provided by the Google search engine are stored in the *suggestion* table. In Fig. 3b we can see the tables containing the web page history index. These two tables contain data about the web pages that the user had visited. For all this data, queries, sub-queries and visited pages, the exact date and time when that data was collected is stored in the database. This information is stored as a timestamp in the *createdAt* column which can be found in almost all the tables.

In Fig. 4 we present some dimensions of our dataset, which includes the total number of users that had the extension installed, the total number of queries that users submitted to Google search engine, the total number of sub-queries and query suggestions that Google offered at that time, and also the total number of web pages that users have visited during the 2 months period when we conducted the experiment.

Min number pages per user	178
Max number of pages per user	11042
Average number of pages per user	3079.13
Standard pages deviation	2333.30

Fig. 5: Average Visited Pages per User

Min number queries per user	18
Max number of queries per user	1799
Average number of queries per user	457.10
Standard query deviation	394.48

Fig. 6: Average Queries per User

Fig. 5 contains information from the dataset about web pages that were visited by the users of the extension. For example, the maximum number of pages that were visited by a single user in the 2 moths period of time, was 11042 web pages, whereas, the minimum number of web pages visited by a user, was 178. We have also computed the standard deviation for all the web pages per user, which is 2333.30. Fig. 6 contains the same type of statistics, but for the queries that the users used on Google search. We can observe that the minimum number of queries that were used by a particular user was 18 and the maximum number of queries was 1799 and the standard deviation is 394,38. The high deviations for both visited web pages and queries tells us the fact that the data in our dataset is very diverse.

It is commonly accepted that search queries can be divided into two main types: navigational queries and informational

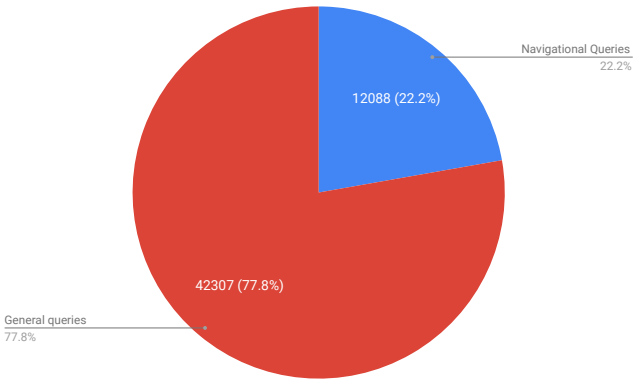


Fig. 7: Navigation and General Queries

queries. A navigational query is a search query entered by the user with the intent of finding a very particular web page. For example, a user might type ”twitter” into Google’s search input in order to find and navigate to the Twitter website, instead of directly typing the full address in the address bar. We can say that whenever a user submits a query to Google, and the URL of the first page that he navigates to contains all the terms from the query, the query is a navigational query. An informational query is a search query that can cover a very large topic, for which, the search engine can return a very large number of relevant information (the website this information comes from is not important by itself). When a user submits such a query to Google, he is looking for some information and not a particular website. In Fig. 7, which is built from the data present in our dataset, we can see that 77.8% of the queries are informational queries and 22.2% are navigational queries. We considered a query to be a navigational query if the URL of the first page, that is visited by the user, contains all query terms; all other queries that do not follow this rule are considered as information queries.

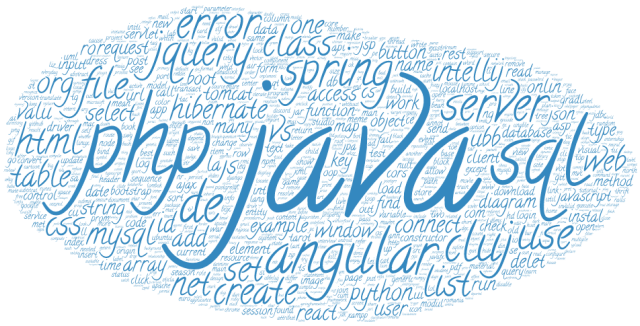


Fig. 8: Queries WordArt

In Fig. 8 we present a word art picture created on www.wordart.com, a word art that was built using query tokens from all the queries used by all the users of the Chrome extension. Given the fact that all the users are computer science students, what we can observe that this word art contains a lot of domain specific tokens like *”java”, ”php”,*

"server", "web", "html", "class", "mysql", "spring" etc.. An interactive version of this word art can be found at [20].

V. Using the Dataset

As seen in Fig. 3a and Fig. 3b, our dataset comes in the form of a relational database. The browsing history of users and the query history of users is stored in the database tables depicted in these two figures. The raw data is already indexed in the database. We can recreate query sessions of a user from the tables in Fig. 3a, using the *createdAt* field. By *query session* we refer to all the query data generated by a user from the moment he/she typed the first letter from a query in the search input box to the moment he/she types or chooses the final query that is submitted to the search engine. This includes the subqueries (i.e. query prefixes) sent by the user to the search engine together with the lists of query suggestions provided by the search engine for each subquery and also includes the final query chosen by the user and submitted to the search engine. A query session takes place in the following way: as the user starts typing characters in the search input of the search engine, the search engine returns a list of query suggestions, $Q_s(i), i = 1..10$, ordered by their relevancy to the user's information need (relevancy is computed by the search engine using a Most Popular Completion technique, $Q_s(1)$ being the most relevant suggestion according to the search engine). The user might continue typing characters and ignore the suggestions offered (i.e. sending additional subqueries to the search engine) or he/she may choose a suggestion to be the final query. In the end, he/she either chooses the final query Q from the list of suggestions provided by Google or he/she writes the final query Q completely (i.e. Q does not appear in the query suggestions list) and submits it to the server. In a *query session* we can define several *query contexts*, i.e. $(SQ, Q_s(1), ..., Q_s(10))$ tuples that are made of the subquery SQ (i.e. the string typed by the user in the search input) together with 10 suggestions offered by the search engine for the subquery SQ.

The dataset could be used for evaluating a query suggestions algorithm in a couple of different ways.

Usage 1. The query component of the dataset (i.e. tables from Fig. 3a) can be used for evaluating the output of a query suggestions algorithm. Because a *user query session* from the database contains the subqueries (i.e. query prefixes) typed by the user and also the final query Q submitted by the user to the search engine, one can evaluate if a query suggestion algorithm is able to suggest the final query Q when the user submits only a small query prefix (i.e. one or two letters of the query). For example it can check whether the customized query suggestions algorithm suggests the final query Q before the Google suggestions algorithm does. Another evaluation can be the one where a customized query suggestions algorithm reorders the query suggestions offered by Google for a subquery SQ so that it moves more relevant query suggestions to top positions in the query suggestions list. The efficiency/utility of such query suggestions reordering can then be evaluated using the MRR (Mean Reciprocal Rank) [6] or SR@k (Success Rate at top-k) [21] metrics. Such an evaluation using a similar, albeit smaller, dataset as the one presented in the current paper is demonstrated in [22].

Usage 2. The web pages component of the dataset (i.e. tables from Fig. 3b) can be used for building a personalized query suggestions algorithm like the one demonstrated in [22]. The user's recent web browsing history can be used by the query suggestion algorithm to render more personalized query suggestions or to reorder the query suggestions list provided by the search engine (i.e. Google for our dataset) so that more relevant query suggestions are moved to the front positions in the list. In addition the user's query history (contained in the tables from Fig. 3a) can also be used for personalizing the query suggestion list. Or the whole database (containing web pages history and query history of all users) can be used in order to generate domain-specific MPC-based (i.e. Most Popular Completion) query suggestions.

VI. Conclusions

We presented in this paper a dataset that can be used for testing query suggestion algorithms in the field of information retrieval. We presented the data collecting methodology, the data structure and several ways it can be used in order to evaluate a query suggestion algorithm for information retrieval. We build this dataset by collecting the web browsing history and query history from 119 volunteers in an experiment that lasted over 2 months. We think that the dataset is very useful for evaluating all sorts of query suggestion algorithms: from MPC-based ones to personalized ones based on the user's private web browsing history or on the user's query history (i.e. the queries previously submitted by this user to the search engine).

The dataset is offered free of charge at the following URL: http://www.cs.ubbcluj.ro/~ionutb/history-collector/dataset

VII. Acknowledgments

We would like to thank to the 2nd year students from Faculty of Mathematics and Computer Science, Babeş–Bolyai University, Cluj-Napoca, Romania, that participated to the experiment and had the Chrome extension installed on their own computers.

References

[1] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, "Clustering user queries of a search engine," in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, pp. 162–168. [Online]. Available: http://doi.acm.org/10.1145/371920.371974

[2] B. J. Jansen, A. Spink, and T. Saracevic, "Real life, real users, and real needs: A study and analysis of user queries on the web," *Inf. Process. Manage.*, vol. 36, no. 2, pp. 207–227, Jan. 2000. [Online]. Available: http://dx.doi.org/10.1016/S0306-4573(99)00056-4

[3] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma, "Probabilistic query expansion using query logs," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 325–332. [Online]. Available: http://doi.acm.org/10.1145/511446.511489

[4] M. Sanderson, "Ambiguous queries: Test collections need more sense," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 499–506. [Online]. Available: http://doi.acm.org/10.1145/1390334.1390420

[5] L. Li, H. Deng, A. Dong, Y. Chang, H. Zha, and R. Baeza-Yates, "Analyzing user's sequential behavior in query auto-completion via markov processes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA: ACM, 2015, pp. 123–132.

[6] Z. Bar-Yossef and N. Kraus, "Context-sensitive query auto-completion," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 107–116. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963424

[7] C. Manning, P. Raghavan, and H. Schutze, *An introduction to Information Retrieval*. Cambridge University Press, 2009.

[8] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1005345

[9] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *Proceedings of the 1st International Conference on Scalable Information Systems*, ser. InfoScale '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1146847.1146848

[10] "Text retrieval conference (trec) data," accessed: 31-May-2019. [Online]. Available: https://trec.nist.gov/data.html

[11] "The clueweb09 dataset," accessed: 31-May-2019. [Online]. Available: http://lemurproject.org/clueweb09/

[12] "The clueweb12 dataset," accessed: 31-May-2019. [Online]. Available: http://lemurproject.org/clueweb12/

[13] "Kaggle: Your home for data science," accessed: 31-May-2019. [Online]. Available: https://www.kaggle.com/

[14] "Statcounter global stats: Browser market share worldwide," accessed: 31-May-2019. [Online]. Available: http://gs.statcounter.com/browser-market-share

[15] "W3counter: Web browser market share trends," accessed: 31-May-2019. [Online]. Available: https://www.w3counter.com/trends

[16] "Statista: Search engine market share worldwide," accessed: 31-May-2019. [Online]. Available: https://www.statista.com/statistics/216573/worldwide-market-share-of-search-engines

[17] "Statcounter global stats: Desktop search engine market share worldwide," accessed: 31-May-2019. [Online]. Available: http://gs.statcounter.com/search-engine-market-share/desktop/worldwide

[18] "Search engine market share," accessed: 31-May-2019. [Online]. Available: https://netmarketshare.com/search-engine-market-share.aspx

[19] V. Niculescu, D. Bufnea, and A. Sterca, "MPI scaling up for powerlist based parallel programs," in *Proceedings of 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2019), Pavia, Italy, February 13-15, 2019*, 2019, pp. 199–204.

[20] "Wordart," accessed: 31-May-2019. [Online]. Available: https://wordart.com/6i76j7rcw0bd/word-art

[21] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng, "Learning user reformulation behavior for query auto-completion," in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, ser. SIGIR '14. New York, NY, USA: ACM, 2014, pp. 445–454. [Online]. Available: http://doi.acm.org/10.1145/2600428.2609614

[22] I. Bădărînză, A. Sterca, and F. M. Boian, "Using the users recent browsing history for personalized query suggestions," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sep. 2018, pp. 1–6.