Exploration of Product Search Intents via Clustering of Query Clusters

Omar Alonso
Microsoft
omalonso@microsoft.com

Vasileios Kandylas
Microsoft
vakandyl@microsoft.com

Serge-Eric Tremblay
Microsoft
sergetr@microsoft.com

ABSTRACT

We describe a system that organizes search results in the context of an exploratory product search session where the user is researching goods. Compared to existing approaches that use predefined categories to filter results by attributes, we organize information needs based on queries instead of documents. The idea is to organize queries around the same topic and produce a hierarchical representation of intents that describe information about a product from different perspectives. We present a prototype implementation using a real-world data set of 24M queries.

ACM Reference Format:
Omar Alonso, Vasileios Kandylas, and Serge-Eric Tremblay. 2019. Exploration of Product Search Intents via Clustering of Query Clusters. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW'19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3308560.3316606

1 INTRODUCTION

Product search continues to be a very popular activity as most products can be purchased online. Users can use a web search engine or a shopping website to locate specific merchandise. Web search engines return product information as a list of pages ranked by relevance, whereas a shopping site organizes relevant results according to categories and past purchase behavior. We propose an alternative based on the queries that users enter when they are looking for a product and how that data can be used for organizing the various aspects that are required for making a purchase decision. We hypothesize that searching for a product involves more than sifting through pages, in particular when items are somewhat pricey, or the user is not familiar with the product line.

Consider the following scenario: we are looking to purchase a new dishwasher to replace the old one. We don't know a lot about this type of appliances and we need to do a bit of research. What is the latest on this product? What is a good brand? Are there any reviews? A search on a shopping vertical shows a wide range of products with many filters that are useful when we are ready to make a purchase decision, which is not our case yet. Web search results do cover more content but, at the same time, not fully related content is included. When people are looking for a dishwasher, what are the most common things that they express?

We are interested in organizing product information for supporting exploratory search scenarios [6]. Our work centers around structure and organization and not on user interfaces. In fact, we use a standard clustering search interface to show search results.

There is previous research on systems that organize search results and how to present them in a user interface so the user can interact with the collection in different ways. The book by Hearst describes interfaces for grouping search results and collections by flat categories, hierarchical categories, and faceted categories [4].

A well-known approach for organizing content is clustering, a technique that derives groups and labels in an unsupervised fashion. A disadvantage of clustering is that the output quality is not as good as a category system. To avoid some of the problems with traditional clustering on documents, web clustering organizes items that are returned by a web search engine using as input the titles and snippets of the returned pages [3]. There is recent work on bundling search results using composite retrieval for exploratory entity search task using a question-answering data set [2].

While product search is a pillar in e-commerce, published research on this topic is sparse. A high-level overview of the techniques used by Amazon is presented in [9] where the authors identify blending of results from different categories as a big challenge for ranking. Li et al. [5] points out that the process of buying a product is different than the process of locating relevant documents in a traditional search engine and propose a system that recommends products that have the best value.

Web search engines do have a related searches feature, a short list of queries, that is usually placed at the end of the page. We are interested in expanding the notion of related queries in combination with clustering to organize topic exploration for product search. In contrast to previous work that clusters web search results or a set of documents, we pre-process a search engine query log by *clustering queries* using clicks. In the same spirit as web clustering, we search the query clusters and then cluster the search results of such queries. That is, an offline clustering step and then an on-the-fly clustering, or bundling, to organize the output of the original clusters according to the user queries. Our goal is to provide a summary of the most relevant intents that users provide in the context of a product search session. Figure 1 shows the system high level architecture.

2 METHODS

We now describe the two main techniques implemented: clustering queries based on clicks and packaging the clusters using a suffix array-based algorithm.

2.1 Clustering Queries

In order to infer semantic connections between queries from the search query log, we exploit links clicked for each keyword. This approach lets us detect non-obvious semantic associations, and it

WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3316606

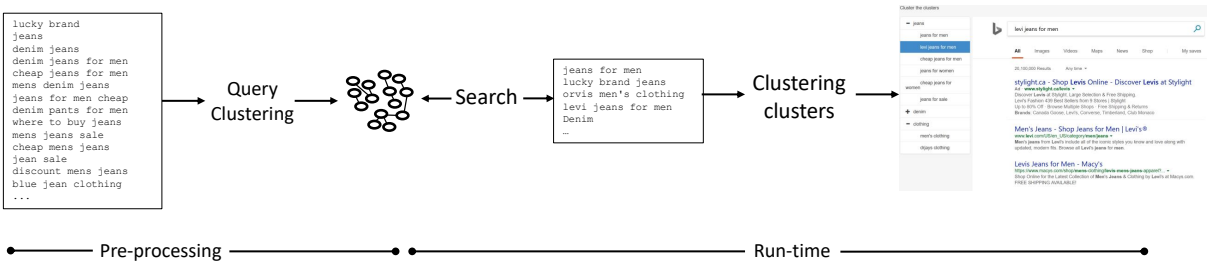

Figure 1: System architecture. The data processing phase consists of clustering a search query log using clicks and producing a data sets of clusters. At run-time, the search results for a query are clustered and presented as a hierarchical tree for the user. The user can explore the many aspects by navigating the tree that is presented on the left.

is practical to implement [1]. Consider a vector space where each dimension represents a link from the Bing search query logs. We associate each query to a vector where each component of the vector represents the number of clicks on the link. These vectors are high-dimensional but very sparse. We compute cosine similarity between the two vectors to obtain similarity. If we compute the distance between every possible pair of queries, we obtain a query similarity graph. In this weighted, undirected graph, each vertex represents a query, and the edges describe their similarity.

Once the query similarity graph is built, the next step is to create groups of related queries with community detection. The idea is to identify communities (clusters) of queries which are densely connected to each other, but loosely connected to the rest of the graph using modularity maximization [7].

Consider an undirected graph $G = (V, E)$. For the sake of presentation, we consider that this graph is not weighted, but that more than one edge can connect two nodes. Consider a set of vertices $C \in V$. The modularity measures how densely connected C is. To compute it, we count the number of edges within the set, and compare to what we would expect if the edges were drawn randomly between G's vertices, preserving the vertex degrees. The modularity is the difference between these two terms. Let m_C be the number of edges and $E[m_C]$ the expected number of edges in the set C. The modularity of C is:

$$Mod(C) = m_C - E[m_C]$$

If the vertices of the graph G are partitioned into p partitions $C_1, ..., C_p$, then the total modularity $TMod$ is the sum of the modularities of each of these partitions. To find the modularity of a set of vertices C, we can compute m_C by counting the number of edges within C. But how do we obtain the expected number $E[m_C]$? Let's draw an edge at random between two vertices of G. Let P_C describe the probability that the edge connects two vertices of C, and let m_G denote the number of edges in the graph. We obtain:

$$E[m_C] = m_G * P_C$$

We now compute the probability P_C. Let $D_G = 2 * m_G$ represent the sum of all the degrees of all the vertices of the graph, and let D_C represent the sum of all the degrees of the vertices in C. For a given edge, the probability that one of the endpoints ends up in the set C is D_C/D_G. Therefore,the probability of having both endpoints in the community is:

$$P_C = (D_C/D_G)^2$$

Putting everything together, we obtain:

$$Mod(C) = m_C - m_G * (D_C/D_G)^2$$

For clustering via modularity maximization we proceed as follows. We initialize the algorithm by assigning each vertex to its own community. Then, at each iteration, we find the two closest communities, and merge them. We stop when we cannot improve the score anymore, or when we have reached a satisfying number of communities. Given the size of the query logs, we use a custom variant of Newman's procedure presented in [8] that includes the following three steps:

(1) For each community, list all the neighbor communities. Two communities are neighbors if (a) they are connected and (b) if we union them, the total modularity increases. We obtain several neighborhoods, one for each community.
(2) The neighborhoods found in Step 1 are overlapping: one community may belong to several neighborhoods. To remediate this, take each community, list all the neighborhoods to which it belongs and keep the closest one (modularity increase is as large as possible).
(3) For each neighborhood, aggregate all the communities into one large, new community.

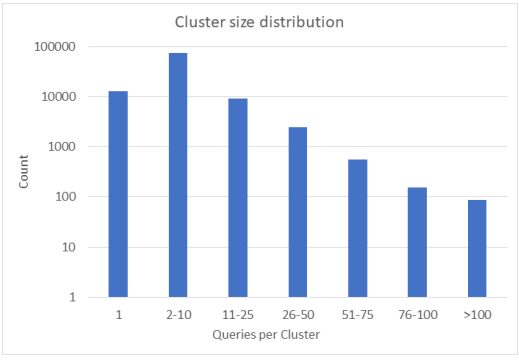

Figure 2: Cluster size distribution.

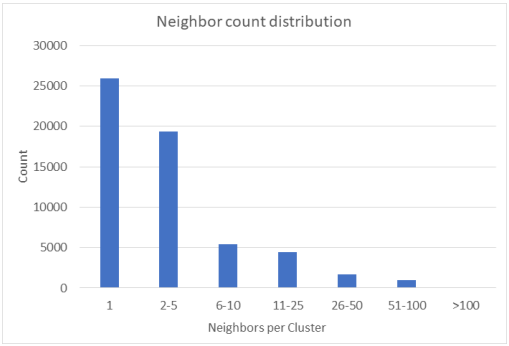

Figure 3: Cluster neighbors.

Figure 2 shows community (cluster) sizes. We observe that a large majority of communities contains between 2 and 10 queries, and that there are very few communities with more than 50 items. After the communities have been detected, we also find the neighboring communities, i.e. those communities which are close (up to a threshold) but not so close as to be merged together according to modularity maximization. Figure 3 shows the distribution of the number of neighbors where we can notice that most communities have 1 to 5 neighbors.

In our scenario, neighbors are useful because they contain information that is relevant to the main cluster in a different way that can be useful discovering relationships and serendipity. Table 1 shows a sample of clusters with their respective neighbors ranked by similarity. In the case of the `gucci` brand, the neighbors describe diverse types of products but also a competitor. In the `iphone` example, not only a competitor is mentioned but there is a neighbor that anticipates the next move (i.e., `where to buy`). The examples for `refrigerator` and `columbia` include companies that sell such products. Finally, the last example contains more diverse choices.

2.2 Packaging

As mentioned earlier, web clustering engines work as a meta search engine by issuing a query to an existing web search engine index and processing the returned titles and snippets from the search engine result page. We follow a similar approach by issuing a query to our set of pre-computed query clusters and returning the main cluster, its neighbors, and the queries per each cluster.

For packaging (a form of clustering) the cluster and its neighbors, we adopt a suffix-array as the main data structure, which is also common for this type of solutions [3], and incorporate a number of heuristics for traversing the array. A nice property of suffix arrays is that all the suffixes are close in proximity for fast lookup.

The algorithm (presented in listing 1) works in two phases as follows. We first *exploit* the retrieved cluster and its content and then *explore* the neighbors. In the first phase, given the user query, we locate all suffixes in the suffix array that are next to each other using binary search. This subset forms the base subset for traversing the suffix tree and selecting the corresponding queries (labels) as the first and most important cluster, the *base cluster*. Figure 4 shows the base cluster for "`dishwasher`" and how the tree is derived (i.e., indexes 29->28->27 produce "`18 inch dishwasher`"). The text "`18 inch dishwasher`" is selected as final because there is an equivalent query in the input data whereas entries like "`depot dishwashers`" or "`at lowes`" are not because they are likely to be part of bigger query string.

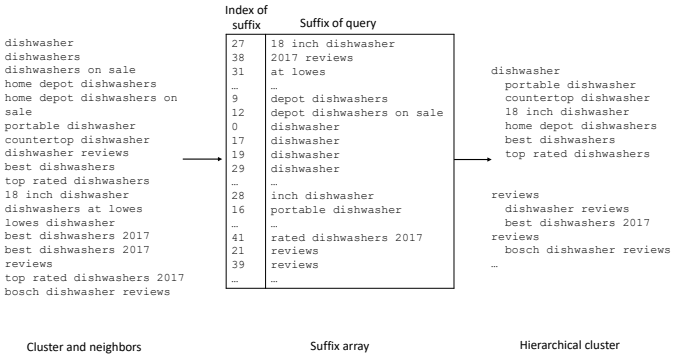

Figure 4: Example of how the suffix array is derived from the input data and how the base cluster is constructed by following the index. User query is {dishwasher}.

The base cluster is very relevant to the query but not sufficient for discovering information related to the topic. By computing term frequency (TF) for each n-gram of every suffix, we can identify high frequency terms that are candidates for exploration, that is, a second round of clustering. We iterate over a set of n-grams with high TF and produce another set of clusters. Finally, we merge the base clusters derived from the input query and the clusters produced by the n-grams with high TF into the final output. Compared to traditional clustering techniques like k-means, suffix arrays are very efficient for implementing clustering on small pieces of text.

Data: Query clusters computed by modularity
Input : User query q
Output : Clusters of queries relevant for query q

$qclusters \leftarrow$ find clusters and neighbors for q
/* parse input data, tokenization and stemming */
$sarray \leftarrow$ create suffix-array ($qclusters$)
/* exploitation */
$stree \leftarrow$ find query in $sarray$
$bcluster \leftarrow$ traverse $stree$ and extract labels
/* exploration */
$terms \leftarrow$ select terms from $sarray$ order by TF
$oclusters = \emptyset$
foreach t *in* $terms$ **do**
  $tc \leftarrow$ traverse $stree$ and extract labels
  $oclusters \leftarrow oclusters \cup tc$
end
$clusters \leftarrow$ merge($bcluster$, $oclusters$)
print $clusters$

Algorithm 1: Packaging clusters algorithm

Table 1: Sample of clusters and their respective neighbors ranked by similarity. Each cluster contains many queries.

gucci	iphone	refrigerators	columbia	dinner table
gucci glasses	iphone x	fridge	columbia clothing	round glass dining table
gucci bags	iphone 6s	home depot refrigerators	outdoor clothing for men	dining room sets
gucci sneakers	apple iphone	discount appliances	llbean sale	kitchen chairs
designer purses	iphone camera	refrigerator reviews	orvis men's clothing	farmhouse table
gucci handbags	where to buy	costco refrigerators	vantage apparel	small kitchen tables
chanel handbags	samsung phones	freezerless refrigerator	columbia jackets for men	amish tables
gold clutch	iphone review	kitchen appliance packages	coats	dining room furniture

3 RESULTS AND IMPLEMENTATION

The back-end pipeline runs on Cosmos, Microsoft's internal big data analysis platform. We use a data set of 24M queries from Bing categorized as commerce-related collected over a 1-month period. In our data set, there were 72M distinct links and the average sparseness of the query vector was 7.8 non-zero elements (clicked links per query). All these data is employed for constructing 3M query clusters using the technique described in Section 2.1. The top query within a cluster is indexed for search purposes.

At run-time, the computation works as follows. Once the user has entered a query, the system finds the most relevant query cluster and its neighbors ranked by similarity and returns a list of cluster labels and top queries from each cluster. This list is then clustered again using the method from Section 2.2 and presented as a Javascript tree explorer on a web browser where the user can explore the content. Each leaf entry triggers a query in a search engine allowing the user to examine the search results provided by the different views in the same user interface. Figures 5 and 6 show examples of our techniques in action.

The objective of the system is to group similar queries so the user can make sense of previous searches and intents when looking for a product. Instead of recommending related searches like in traditional web search engine results, we organize queries according to similar intents with the goal of providing the user the big picture when searching for products.

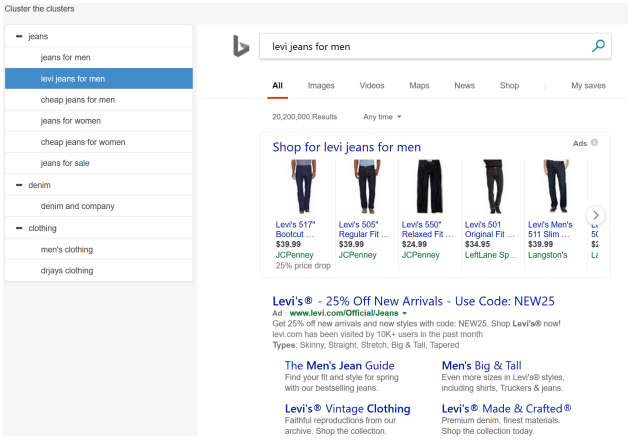

Figure 5: Search results for the query jeans.

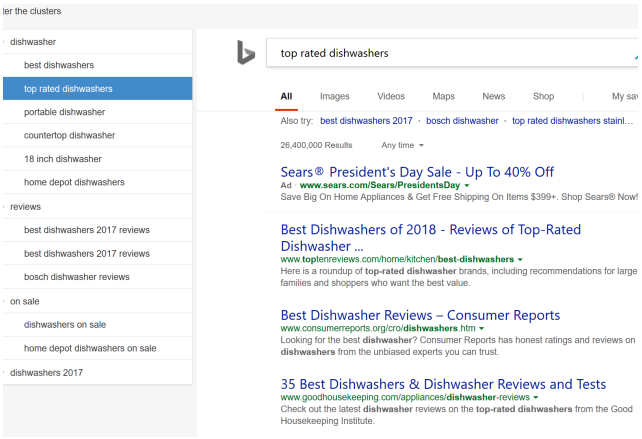

Figure 6: Search results for the dishwasher example.

4 CONCLUSION

We described a prototype that uses query logs and click data to cluster similar queries for grouping similar intents and how to cluster the output of such clusters to organize search results. The methods presented do scale and have been implemented in a working system. The demo is built only using queries instead of documents or surrogate pages like in previous systems. We also provided details on how the techniques work so they can be reproduced with other data sets. Future work includes the design of a user study for comparison against related queries.

REFERENCES

[1] Ricardo A. Baeza-Yates and Alessandro Tiberi. 2007. Extracting semantic relations from query logs. In *Proc. of SIGKDD*. 76–85.

[2] Ilaria Bordino, Mounia Lalmas, Yelena Mejova, and Olivier Van Laere. 2016. Beyond Entities: Promoting Explorative Search With Bundles. *Inf. Retr. Journal* 19, 5 (2016), 447–486.

[3] Claudio Carpineto, Stanislaw Osinski, Giovanni Romano, and Dawid Weiss. 2009. A survey of Web clustering engines. *ACM Comput. Surv.* 41, 3 (2009), 17:1–17:38.

[4] Marti A. Hearst. 2009. *Search User Interfaces* (1st ed.). Cambridge University Press.

[5] Beibei Li, Anindya Ghose, and Panagiotis G. Ipeirotis. 2011. A Demo Search Engine for Products. In *Proc. of WWW*. 233–236.

[6] Gary Marchionini. 2006. Exploratory Search: From Finding to Understanding. *Commun. ACM* 49, 4 (2006), 41–46.

[7] M. E. J. Newman. 2006. Modularity and community structure in networks. 103, 23 (2006), 8577–8582.

[8] Thibault Sellam, Martin Hentschel, Vasilis Kandylas, and Omar Alonso. 2016. e#: Sharper Expertise Detection from Microblogs. In *Proc. of EDBT*. 563–572.

[9] Daria Sorokina and Erick Cantu-Paz. 2016. Amazon Search: The Joy of Ranking Products. In *Proc. of SIGIR*. 459–460.