Query-context Search Result Clustering basing on Graphs*

Michal Meina
Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18, 87-100 Toruń

Abstract. This paper deals with search result clustering by introducing a novel approach to in-context clustering. Basing on additional information provided by the user in query we can develop custom document grouping. This work is a preliminary (extended abstract) version.

1 Introduction

Information Retrieval Systems (e.g Carrot2 Clustering Engine\(^1\) or Grouper [17]) often provides mechanisms to organize retrieved results in clusters in order to simplify process of navigating to an interesting document. In those cases search results clustering is a type of postprocessing result set. According to [8] giving a user simple mechanism to correct or change clustering can significantly improve Information Retrieval (IR) task. Although search-result clustering is currently the subject of extensive research there is lack of approaches that reformulate interaction scenario for clustering engines. We state that this ability can be crucial in narrowed dataset where differences between clusters could be very subtle. This paper introduces query-context search result clustering method based on graphs.

Our system operates on result search taking an additional query(ies) \(q_{desc}\) and makes query-context document clustering. Instead of treating document-to-cluster belonging as a document property we will allow same documents to be assigned to entirely different clusters depending on additional queries (clustering intention). Our algorithm generates a cluster description first and then assigns documents into one or more clusters matching document with description. This technique is called description-comes-first strategy [14]. Descriptions are generated in a way that takes into consideration user query which can lead to a situation where the same documents set generates different clustering. Search task introduced by our method is illustrated by example on Table 1.

<table>
<thead>
<tr>
<th>search query</th>
<th>additional queries</th>
<th>intended clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone</td>
<td>approximation of standard clustering</td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>apple</td>
<td>product announcements</td>
</tr>
<tr>
<td>microsoft</td>
<td>product comparison</td>
<td>official statements about counterpart products</td>
</tr>
</tbody>
</table>

Table 1.

For document representation we developed graph model on top of classic Vector Space Model (VSM). Idea that stands behind using graphs comes from efficiency-effectivity tradeoff. In online query processing we cannot do much computation, so we use “richer” model which is constructed offline and then we apply

---

\* This work was supported by grant N N516 077837 from the Ministry of Science and Higher Education of the Republic of Poland.

\(^1\) http://search.carrot2.org/
“faster” algorithm in query processing phase. As it was pointed in [7] semantic analysis such a Latent Semantic Indexing (LSI) make extension to VSM that captures information about co-occurrences of higher rank. In models based on graphs those co-occurrences are expressed directly in the model, which facilitates the use of this information in the clustering task.

Our clustering scenario does not focus on optimising standard quality criterion of clusters but according to [10] high score on internal measure do not necessarily result in effective information retrieval applications. Our goal is to create strong boundaries between clusters, and scatter topics across individual groups to be much “orthogonal” taking $q_{desc}$ as criterion of orthogonality. Still we can maintain acceptable high quality of clusters which are organized in context of a query.

Contributions of this paper are as follows: (1) introduction of Query-Summarize Graph - a graph structure which can be used in clustering search results in context of a user query (2) definition of similarity measurements between document and term graph.

2 Related Work

Earliest attempts to cluster-based search engines depend on clustering whole documents collection and then adding this information into results. For example in [1] interactive dendrogram was presented alongside keyword search result.

Leouski et al. [8] indicates that clustering of search results gives better user-feedback and shortends time to retrieve documents. Morover this technique gives an opportunity to a user to discover some additional knowledge about dataset. Conclusions of this work tells that giving some means to a user in order to correct or completely change classification structure (e.g. by setting cluster boundaries) can significantly improve searching.

In [6] authors introduce scatter/gather approach to present and navigate within documents. This mechanism redefine search task providing “query without typing” user interface strongly. Search process starts with presenting large clusters of whole document collection and then it allows user to select (gather) interesting clusters or items. The narrowed set is merged into one collection and clustered again (scattered). This step is repeated as long as the whole dataset is narrowed sufficiently. Mentioned paper is a good example of IR system that strongly facilitate clustering.

There is a number of papers that deal with web-search result clustering (e.g. [16] or [5]). Clustering of webpages slightly differ from regular document clustering. The problem here is in refining terms extracted from webpages since words can differ from its context or developing documents features that takes into account web environment. Gelgi et al. [4] introduces graph based approach in refining terms from a web-search result. They built a undirectet graph of terms in which edges weight corresponds to co-occurence in search results. Their variation of PageRank algorithm called TermRank was able to distinguish discriminative, ambiguous and common words in order to refine terms in documents before clustering. TermRank was significantly better than classic tf-idf weightening scheme before clustering.

There is a number of possibilites of how we can model documents as graphs. Starting with simple ones presented in [12], where authors describes six methods how we can map terms extracted from documents as graphs. Our model is very similar to simple model from this paper, but we are more focused on investigation different techniques of graph construction. In [9] authors present two algorithms that make keyword-summaries of documents. The experimental results reports gaining high f-masure of keyword extraction against summarization prepared by humans.

More sophisticated graph models are variation of Conceptual Graphs (CG’s) [13] which provides formal definition of semantic representation. Automatic construction of CG’s is a difficult problem and not necessarily easy applicable in Information Retrieval tasks. In [11] authors was using simplified CG’s that provided more semantic-oriented analysis. They slightly improved retrieval performance comparing to standard vector space model by defining documents similarity to take into consideraton graph structure. In [3] authors were able to capture domain knowledge in graph representation, stating that this is natural and very intuitively method of representing knowledge.
Finally [15] Multi-Level Association Graphs (MLAG’s) was able to subsume various IR models into one common also defining similarity measure between on-line query by Random Walk on precomputed MLAG. None of mentioned works consider construction of graph in a query processing pipeline. Moreover, none of these works did not investigate different technique of the model instantiating.

3 Graph generation

Graph generation is divided into two phases: (1) first we construct graph depending on search result and user query (2) we decompose it into number of disjoint subgraphs.

Query-Summarize Graph is a weighted, undirected graph $G = (V, E)$ whose vertices corresponds to terms ($t \in T$) in documents and edges weights are assigned according to semantic distance between terms (in our case induced from results). There is a set of distinguished nodes whose corresponding terms occurred in a query $q = (t_1, t_2, ...)$. Graph is constructed iteratively - nodes and edges are being added to graph basing on previously added nodes until stop-condition is satisfied.

Graph generation starts by adding nodes for terms extracted from query whereby edges between them are immediately created making small local clique at beginning. In $n + 1$ step of algorithm we add term that are semantically closest to previously added terms in step $n$.

Adding nodes to graph is constrained by $k$ semantically closest terms threshold for each visited node. Naive implementation could depend on choosing this parameter as fixed or depending it on fixed semantic distance threshold. Through transitivity of similarity relation the edges would be created only between nodes added in $n$ and $n + 1$ step that in the end can result in graphs that will have unsatisfactory low clustering coefficient. Moreover stop-condition could be enforced only by choosing unintuitive depth parameter that stops algorithm after specified number of steps.

Our alghoritum changes the number of terms to be added basing on the number of terms added in previous step. We follow simple strategy: if we have added previously a large number of nodes (which stand for possibly exploration of a new topic) we should add a small number of terms which are semantically very close to this one (which contribute to in-depth description of the topic). In first step we add a InitialSeed number of semantically closest terms and in every next step we add $k$ closest ones where

$$k = \frac{\text{InitialSeed}}{|\text{NodesAddedInPreviousStep}|}$$

Later we show that in our experiments this method tends to stop after relatively small number of steps outputing graph with gratifying clustering coefficient.$^2$

After constructing graph we decompose it to disjoint subgraphs by discarding edges that connects modules (generalization of connected components) in graph. We use louvain method [2] to find those modules which

---

$^2$ Our approach of tunnning $k$ parameter in this preliminary work is very simple. Nevertheless this step makes great space for research in final version, since according to [] semantic similarity distribution and similar term overlapping is strongly based on topics captured in documents.

348
is an optimisation of modularity maximization algorithm. This algorithm tries to partition the graph into modules that have strong internal connection and sparse connectivity between modules. Modularity is a benefit function that in this case measures quality of particular division according to information how module could be connected at random.

In search task our system is using those subgraphs as structural cluster descriptions.

4 Query-Summarize Graphs structural analysis

For the needs of our experiments we crawled web for newswire text from four different sources. For data preparation we apply small hand-made stop words list and stemming. Since articles from newswires have large number of named entities we developed heuristic that identifies and normalizes those basing on n-gram model. As semantic similarity we have used cosine distance on tfidf model

\[ \text{sim}(t_i, t_j) = \frac{V(t_i) \cdot V(t_j)}{|V(t_i)| |V(t_j)|} \]

where \( V(t_i) \) is term column in tf-idf weighted document-term matrix.

Search results were divided into two groups using keyword search and time span search. Graphs was generated by set of human-choosed queries \( q_i \) using first 100, 200, 500, 1000 documents from search results. On figures and tables \( q_i - ts \) and \( q_i - q_j \) stands for graph generation by query \( q_i \) respectively on time span search result and keyword search results by query \( q_j \).

![Fig. 1. Nodes count (y-axis) and number of steps of algorithm (x-axis)](image)

For large set of randomly chosen queries graph generation stops in not more than fifty steps. Most of the queries picked randomly were having large \( tf \) and small \( idf \) value and those terms generates graphs where node count was around \( \text{InitialSeed} \) number. That indicates when we are querying with terms that make small contribution in any topic we end with undescriptive graphs. On the other hand querying with highly
Descriptive words (e.g., named entities) generate graphs with a large number of topics. Information about the number of nodes in the Q-S Graph on every step of the algorithm shows steady growth with few rapid increases. This follows our intuition in tuning the $k$ parameter - nodes added when $k$ parameter was low tend to make strong bounds with parts of the graph that in the future become modules than with other parts of the graph.

Another promising result is that graph size tends to have higher correlation with a query than with the size of the result set.

After generating Q-S graphs, we need to decompose it into disjoint subgraphs. Information about modularity of Q-S Graphs is measured by the Average Clustering Coefficient. This measure indicates how nodes in graphs tend to cluster together. Assuming that $N_v \subseteq V$ is the set of immediate neighbors of node $v$, the Clustering Coefficient is defined as follows:

$$C_v = \frac{2|\{e_{jk}\}|}{|N_v||N_v| - 1} : v_j, v_k \in N_v, e_{jk} \in E$$

$$C = \frac{1}{n} \sum_{v \in V} C_v, \text{ where}$$

<table>
<thead>
<tr>
<th>DocNo</th>
<th>q1-ts</th>
<th>q2-ts</th>
<th>q3-ts</th>
<th>q1-q1 random</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.5155</td>
<td>0.8882</td>
<td>0.6303</td>
<td>0.6395 0.1759</td>
</tr>
<tr>
<td>200</td>
<td>0.5161</td>
<td>0.6241</td>
<td>0.6868</td>
<td>0.5741 0.0987</td>
</tr>
<tr>
<td>500</td>
<td>0.6712</td>
<td>0.4949</td>
<td>0.8483</td>
<td>0.4725 0.0675</td>
</tr>
<tr>
<td>1000</td>
<td>0.5172</td>
<td>0.4378</td>
<td>0.7991</td>
<td>0.7230 0.0797</td>
</tr>
</tbody>
</table>

Table 2. Average Clustering Coefficient for Q-S Graphs and Alberti Barabasi random graph model

We see that Average Clustering Coefficient was significantly larger than random graph of similar size. That indicates that Q-S Graph generated with tuning strategy of $k$ parameter tends to have nodes arranged in highly coupled groups.

Most interesting point is how cluster descriptions look like if we query for semantically close terms. In order to evaluate clustering features in this preliminary work, we take a case-driver evaluation choosing another three queries $q_a, q_b, q_c$. But in this case $q_a$ and $q_b$ was synonyms and $q_c$ was the generalization of two others, and we generated graphs against time span search results. As we expected Q-S Graphs for queries that were synonyms were having similar size (respectively 143 nodes and 279 edges to 158 nodes and 298 edges). Graph from the last query was significantly larger (176 nodes and 402 edges), capturing more terms. Number of clusters was respectively 14, 19, 17. It was not expected that number of clusters will be different for synonyms.

Terms for every cluster description were subjected to a manual evaluation, and results were encouraging. For synonyms the largest modules were almost identical. More than 30% of cluster descriptions were having large number of common words and the differences come from different context in which each term tends to be used in newswire texts. Other cluster descriptions and their difference was very hard to interpret without examining documents. Largest clusters in graph for query $q_c$ were very similar to the largest clusters for $q_a$ and $q_b$, but were enriched by new words. New words were covering a larger semantic field of the generalized query.

Our preliminary experimental results show that our algorithm stops outputing graph with high clustering tendency. That allows us to decompose it into a number of disjoint subgraphs $G'_1, G'_2, ..., G'_k$, which we later treat as cluster-descriptions. After examining words in decomposed graphs, we encountered that distribution of their features follows our intuition in query-context search.

5 Application in Information Retrieval System

Q-S Graphs can be employed easily in search result clustering based on user interaction. After retrieving results, documents could be grouped according to an initial or additional query (which can be applied
repeatedly). We propose score function that will classify every search result according to cluster descriptions. We assume that document is represented as a vector of extracted terms \( d = [t_1, ..., t_n] : t_i \in T \), cluster description is subgraph of \( G \) that \( G' = (E' \subset E, V' \subset V) \), with weight function \( w \).

First we need to define measure of correspondence of term \( x \) to graph \( G' \) in context of a term \( t \):

\[
P_{G',t}(x) = \sum_{y \in N(x)} \text{sim}(t, y) w(x, y) \frac{P_{G',t}(y)}{|N(x)|}
\]

At this point we can define two score measures that indicates document to cluster belonging. Strong score (which filters documents whose nodes are presented in graph \( G' \))

\[
\text{score}_{G'}(d) = \sum_{t \in d \land t \in V'} P_{G',t}(t)
\]

Strong score can significantly narrow document collection depending it on the size of graph \( G' \). In order to classify every document we can define smooth score, as:

\[
F(d) = \{t' : \text{sim}(t', y) = \max(\{\text{sim}(t, y) : y \in V'\})\}
\]

One drawback of constructing Q-S graphs in on-line query processing is that we need to construct tf-idf weighted document-term matrix for retrieved document set and make \(|V||T|\) computation of similarity distance. Beside simple optimisation that we can use in sparse vectors computation we can redefine semantic distance function \( \text{sim} \) as for example number of co-occurences two terms in documents like in [4] \( \text{sim}(t_i, t_j) = |\{d \in D : t_i \in d \land t_j \in d\}| \)\). Other possibility is using it as precomputed from whole or fraction of dataset or even taken from elsewhere in the first place. This can lead us to application in another IR technique called Query Expansion\(^3\). After construction Q-S graph for query \( q = (t_1, t_2,...) \) we can discover (e.g TermRank) from it additional terms \( t'_1, t'_2 \)... which can contribute to fast inverted index scan (keyword search) by query \( q' = (t_1, t_2,..., t'_1, t'_2) \), retrieving more potentially interested documents.

References


\(^3\) the process of reformulating a seed query to improve retrieval performance