Web Experience Enhancer Based on a Fast Hierarchical Document Clustering Approach

Alexandru Ionut Dospinescu[1], Florin Pop[1], Valentin Cristea[1]

University *Politehnica* of Bucharest, Faculty of Automatic Control and Computers, Computer Science Department, Romania
alexandru.dospinescu@cti.pub.ro, florin.pop@cs.pub.ro.
valentin.cristea@cs.pub.ro

Abstract. As the importance of the Internet has increased rapidly to new heights especially with wide adoption of web services and advent of cloud computing, the amount of sheer information that can be obtained from this global information service center leads to major challenges concerning finding relevant information for a casual user or a group of users. On the other hand, even after finding useful documents, it is hard to keep track of them as the amount of useful findings (bookmarks) increases. Also, one has to manually bookmark each time a page is even remotely interesting in order to make it easier to return to that page. In this paper we formulate a new approach to tackling these issues through a model that takes into account not just a single user's experience and intentions, but those of an entire group of users as well. We propose an application in the shape of a browser plugin/extension powered by a dedicated and highly responsive web server that can help the user keep track of his web experience in a highly organized and easy to navigate fashion while also enabling the user to share some of that experience. The clustering of the data is done with a customized algorithm that allows overlapping while showing improved speed and readability.

Keywords: web document clustering, closed frequent itemsets, social web, user interaction, web application

1 Introduction

With the ongoing trends of web services and the advent of cloud computing, the users are brought closer and closer to the Internet. Its importance in the functioning of the civilized society has never been so staggering. Still, the ever growing amount of information can overwhelm even the best web search engine in finding relevant information for a given user, especially when faced with the trends of applying Search Engine Optimization techniques such as stuffing web sites with a high amount of low-quality links, with little or no topical connection to the host site, in an attempt to gain a higher search engine results page.

A highly adopted solution to the problem of finding relevant information relative to the intention of the user is search result clustering by which documents

from the results list returned are grouped and labeled with relevant tags [11]. Such is the case with the commercial solutions Carrot or Vivisimo [3]. Still, even with such an organized structure it is possible that the information under a group may be either too encompassing for what the user seeks. Also, this clustered information retrieval is exclusively dependent upon the results returned by the search engine which, in turn is highly dependent on the search query. For users that lack the knowledge of the proper terminology for building the appropriate queries to supply to the search engine, this is a great issue.

Even after finding useful documents, the vast amount of information can create obstacles in keeping track of them. While bookmarks are a temporary solution, as the size of bookmarks increases, they soon lose relevance. Also, there is another limitation with bookmarks which is reflected in the user experience. Each time a user finds a page to be remotely interesting, that user must manually bookmark that page. This is highly unwanted and a more unobtrusive approach would be preferred to collecting and labeling references by hand.

In order to alleviate some of the problems encountered by casual as well as experienced users when searching and keeping track of relevant information we propose an application that would keep track of a user's web experience (visited web pages) in a seamless way, would be able to present it to the user in a highly organized and easy to navigate fashion and would enable users to share their experience with other users that use the application. Also, it would provide a way for a user to filter through his web experience.

Practically we suggest a browser plugin/extension (our first implemented version is in the form of a Google Chrome extension) that is backed by a dedicated web server running on a cloud platform that does that by meeting the following requirements:

- the user's experience is seamlessly recorded and stored in an efficient manner as the user accesses various web documents;
- when the user wants to view this experience through the application, the web server should be able to supply the extension with a hierarchical structure representing his experience;
- a user should be able to filter through his web experiences based on time, popularity or a feedback indicator;
- a user should be able to share experience as well as make use of others' shared experiences;
- the application should be very responsive and scalable.

For the last requirement we have implemented a custom version of the Frequent Itemset Hierarchical Clustering (FIHC) [5] algorithm. We present in this paper the architecture and technologies used for its development, the advantages and novelty by referring to others algorithms. We evaluate of the proposed application by comparison of cluster quality and clustering speed between FIHC and iFIHC (improved FIHC).

The rest of the paper is structured as follows. In Section 2 we will briefly overview some of the most relevant related work. In Section 3 we will describe the problem at hand and our solution and architecture. In Section 4 we will

present some of our experimental results and finally in Section 5 we will draw our conclusions and future directions for our application.

2 Related work

In this section we will briefly overview document clustering while focusing on FIHC and frequent itemset mining algorithms.

2.1 Document Clustering

Document clustering can be seen as being divided into: flat/partitioning, hierarchical, density-based, probabilistic, graph-based and model based. Among these the most common approaches have been the hierarchical and partitioning ones.

Regarding the partitioning clustering algorithms, the ones that are mostly used for web document clustering are flat medoid-based. This class of algorithms results in partitions of the document collection called k-means. Each partition will be represented by an object within it. The main scope of this class is to find medoids that minimize the dissimilarities inside their cluster. Some of the more promising algorithms from this class are K-Means, K-Medoids, CLARA, CLARANS [8]. Though these algorithms are very resilient to outliers and are faster than hierarchical algorithms, they do not allow overlapping which is something truly required for our approach.

Still, there is another type of partitioning cluster algorithms, namely fuzzy algorithms. This class allows for overlapping. One of the most representative such algorithm is FCM [2]. The basic idea behind it is that it tries to minimize an objective function normally describing the distance between the medoids of all the clusters an object might belong to. It starts with randomly selecting initial medoids and calculating the membership of all the objects in the dataset to those medoids. At each iteration, new medoids are computed based on the current memberships and new memberships are then assigned. This algorithm would have sufficed only if it didn't require a predefined number of clusters which cannot be known given the dynamic nature of our problem.

Concerning the class of hierarchical clustering algorithms one that produces really good results is UPGMA [6]. It uses a measure determined by the cosine similarity between clusters divided by the size of the clusters being evaluated. But since it requires a precomputed similarity matrix, it is expensive and inflexible when faced with an expanding collection of documents.

Another popular approach for this class of algorithms is taken by those based on frequent itemset methods. This subclass of algorithms is based on clusters being represented by labels made by itemsets that are very frequent inside the representative cluster and very rarely in others. Such labeling provides important information regarding the structure of the cluster contents.

One of the more scalable methods belonging to this class is FIHC that, based on the frequent itemsets found by employing the Apriori algorithm [1], organizes clusters into a topic hierarchy. Some of the features and advantages of this approach are:

- reduced dimensionality of the vector space;
- a very high clustering accuracy and speed;
- does not require a predefined number of final clusters as input;
- easy to navigate with meaningful descriptions.

Overall, this algorithm can be divided into four independent phases:

1. *Extracting the feature vectors* - stop word removal, stemming and vectorization;
2. *Determining the global frequent itemsets* - all sets of terms that are present in at least a minimum global fraction of the document set are determined. The terms are tested for global frequency by their presence and not by TF (term frequency) or TFxIDF (term frequency x inverse document frequency);
3. *Clustering* - generates the initial clusters from the global frequent itemsets and then makes the clusters disjoint based on a score function that represents how well a document belongs to a certain cluster;
4. *Constructing the cluster tree* - entails the construction of a hierarchical cluster tree. In this tree, every cluster has exactly one parent that has a more general label than that of the child. This step also consists of tree pruning through which similar clusters (siblings or children) are merged in order to increase accuracy.

There are a couple of variations of this algorithm that are worth mentioning namely TDC [20] and the one presented by Vikram et al. [17].

The major difference between TDC and FIHC is that TDC uses only closed frequent termsets for building the initial clusters. The reasoning behind this change is that closed termsets can dramatically reduce the size of the final termsets while managing to retain completeness since all non-closed termsets are always included in the closed ones. Also, another important change is that the determination of the minimum support parameter used in obtaining the initial frequent termsets is determined automatically so that it is maximized while covering the entire document set.

In [17], the hierarchical clustering algorithm works similar to TDC only that they use generalized closed frequent itemsets and use Wikipedia as an ontology for improving the document representation. Though this gives better clustering results it is slower than the score based approach.

2.2 Frequent Itemset Mining

Instead of using Apriori to mine frequent itemsets, for a faster implementation of FIHC a better solution seems to be using a closed frequent itemsets mining algorithm such as CloStream[18] that can also handle stream data (a sequence of transactions that arrives in a timely manner).

CloStream is a frequent itemset mining algorithm that efficiently addresses some of the challenges posed by stream data such as speed of processing a new transaction, the support for incremental mining. Moreover, CloStream only mines closed frequent itemsets which have been shown to be more relevant for

analysis [15] and they do not lead to the loss of information [7]. A closed itemset represents a frequent itemset for which there is no frequent itemset that is a superset of the original itemset having the same support.

In order to efficiently store mining information from previous transactions which can then be used to update the closed frequent itemsets as a new transaction is added or removed, the algorithm uses two data structures stored in memory, namely a $ClosedTable$ and a $CidList$, but also a temporary hash table generated and used on an update over the transactions. The $ClosedTable$ contains information about the found closed itemsets. Each entry consists of three fields: Cid - uniquely identifies a closed itemset, CI - contains the itemset and SC - records the support of that closed itemset. The $CidList$ is used to maintain an association between items and closed itemsets where they are present. It consists of two fields namely Item field which identifies the item and a CidSet field which contains a set of the id's of all the associated itemset from the ClosedTable. Also, the algorithm uses two methods for handling what happens when a transaction is added or removed [19]: CloStream+ and CloStream-.

While there is a great advantage with CloStream since it maintains the closed frequent itemsets without rescanning any of the original transactions, the size of the $ClosedTable$ increases exponentialy as it stores every itemset derived from the document set (not just frequent or closed ones).

As a different approach *DCI Closed* [10] is a non-incremental closed frequent itemset miner that boasts a reduced memory footprint and excellent execution times compared to its peers. The algorithm is based on a divide-et-impera approach that makes use of a bitwise vertical representation of the database. Due to its efficient utilization of memory space, it can handle very low support thresholds well. After two passes over the dataset, it finds frequent singletons and generates the vertical bitmap of those singletons relative to the transactions they appear in so that for each item $singleton_i \in setOfAllFrequentSingletons$ a binary array is created where a bit bit_j is only if $singleton_i$ is present in transaction j. Because of this, every closed frequent itemset found can easily be associated with all the transactions that contain it. The detection of closed frequent itemsets is done by closure climbing in that the algorithm tries to find unique order preserving generators, determine their closure and then try to find other generators as supersets of that closure.

3 Problem Description and Proposed Solution

We seek to improve a casual user's informational gain from navigating the web by making use of that user's experience coupled with that of its peers. Practically, we are interested in designing an application that can keep track of the documents the user visits in a seamless way, can organize that information and present it to the user in a helpful and easy to navigate fashion while also enabling that user to share his experience and use the shared experience of others.

In a way, we seek to stimulate cultural evolution between users while also improving for a user the efficiency of using his past experience.

Another aspect to consider as part of our problem is a way to automatically differentiate without receiving explicit input from the user to whether a visited web document contains information with any relevance or not. For example linking pages, or spam pages should not be considered important.

3.1 Problem Requirements

Since there is an increasing trend of migrating from local space to location independent space such as it is the case with Storage-as-a-Service services such as DropBox or OneDrive, our application should be able to be location independent in that every user would have a profile and for that profile all information which the user wants to have organized or be shareable should be stored online. This would also alleviate the need for storage and processing resources from the user to the application's system. Moreover, it would enable the possibility of sharing user experiences since all the information would be centralized.

Considering we are interested in being able to give the user's the best of their web experiences, the application needs to be able to filter out less important experiences. This implies having a feedback measure associated with each document in the document set that make up a particular experience, which can be automatically determined by the application.

Still, the biggest requirement of our application revolves around the way it handles web document clusterization. First of all, the application should be able to extract the valid text content from, but not limited to, any given HTML or PDF document; support for formats such as DOC, DOCX, RTF and other could be included in a future version. Then it should be able to generate a summary for that text which would be used for really fast results. So based on the user preference of allotted time for generating the cluster hierarchies, either the original text or the summary will be used. Secondly, it would be ideal if the application would be able to find relevant mining information related to the terms in each document in a timely fashion. Thirdly, the clustering algorithm should not expect any user supplied parameters such as number of clusters or minimum support.

Concerning the algorithm, it should also be fast, scalable to the size of the document set and as accurate as possible while also allowing overlapping. The reason we want to allow overlapping is because some documents should inherently belong to multiple clusters. For example "frozen" may belong to phase transitions, movies, music, digital animation etc. Lastly, the algorithm should be able to supply meaningful and readable labels for its clusters. Stemmed terms like those used in FIHC or TDC are not entirely qualified as readable.

3.2 Proposed Solution

For our initial version of the application we present a Google Chrome extension that allows the user to create an account and authenticate, thus associating a profile with his web experience. The user can manage via the extension his web experiences as sessions. For example, a session might be called linguistics and

the user would make it the current session while navigating the web in search of relevant documents related to language and language theory. Then he might switch to a session called casual which would track documents related to casual browsing such as meme pages.

The tracking of the documents is done seamlessly while the extension is active, by sending a notification to the server with the session name and URL of the web documents loaded by the user. When the user wishes to see his web experiences organized he opens the search page of the extension and he can select the session for which he wants his experiences as well as various filters for that experience such as time interval or based on feedback measurements.

The application should also be able to facilitate the sharing of web experiences between willing users. This is basically done by having the option of creating or subscribing to an existing community. While the user browses under a community, all his experiences are shared with that community.

Architecture Our application has two components: the client application which is the Google Chrome extension and the web server powering the extension.

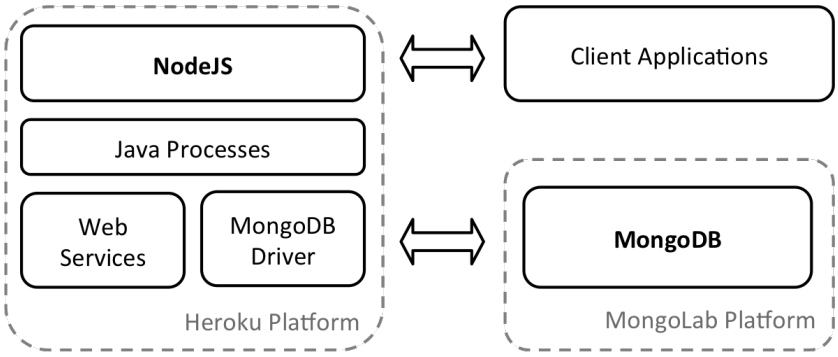

Fig. 1. Web Experience Enhancer Application Architecture

The client application is in the form of a Google Chrome extension that makes use of the Chrome API to track the web documents the user accesses as well as the time he spends on such a document in a browsing session.

The extension is implemented entirely with HTML5, JavaScript and CSS3. This brings about various advantages compared to developing a native application such as:

- perfect integration with the browser technologies;
- the ability to create a rich and easy to use interface comparable to Flash like interfaces;
- the ability to use multiple threads thorough web workers so as to maintain the browser responsiveness while generating the output from the clustering results;

- can easily parse JSON messages with minimum overhead which is most useful for the communication between our client and server components.

Regarding the web server, since we wish for our application to be highly accessible in that it will be location and user machine independent (so it would not matter if the user is using a thin client or not) while also being elastic enough to provide excellent performance and make use of resources as efficiently as possible, seems that using a cloud platform for deploying our application's web server is a reasonable choice, especially since this component is the real core of the application. The web server's importance can be easily explained given the fact that it handles almost everything (creation and storage of user profiles, user sessions, communities and web experiences the database, clustering, extraction to text content, document pre-processing, summarization and pattern mining).

Having considered these, we have chosen Heroku [12] and MongoLab [9] as our initial deployment solutions. Regarding the technologies used for implementing the web server, we decided in using NodeJS coupled with a few Java processes and MongoDB as a database solution (see Figure 1).

Concerning NodeJS, it is a different approach to the stateless request / response paradigm that allows for the developing of real-time web applications employing push technology over web sockets (it makes possible for two-way connections). Also it is innovative in the sense that uses a non-blocking, event-driven I/O model which makes it possible to remain lightweight and efficient when dealing with data-intensive applications and it is capable of dealing with a high number of simultaneous connections with high throughput (highly scalable). Also, since it is based on JS, we can adopt JS across the entire stack which means unifying the development language as well as the data exchange format since both NodeJS and client JS can natively parse JSON messages.

The reason for also employing Java is due to NodeJS's weakness relating to CPU intensive tasks which could impair the very advantages NodeJS can provide. That is why NodeJS will execute Java processes for each of the CPU bound tasks such as clustering, closed frequent termset mining, summarization and text processing.

Regarding MongoDB, we choose it since it is one of the leading NoSQL databases designed for handling big data that features among others document-oriented storage with documents stored in BSON format (can be seen as a giant array of JSON objects), schema free (which is a great advantage in iterative development), replication, auto-sharding, map/reduce support.

Still, the really great thing about using NodeJS with MongoDB and JS for the client component is that JSON is the message exchange format all around which alleviates the need for data conversion and constraint validation.

Filters and feedback measure The application allows for the retrieval of a hierarchical cluster created only from documents of the user session that satisfy some given filters. The filters we proposed for such cases are temporal, related to popularity (the number of visits for a document) and user feedback value.

The latter is a measure of positive feedback from the user interacting with the document over several visits. According to [14], pages that get allocated 2 or more minutes are usually "good pages" and pages that are closed in just a few seconds are "bad", with pages usually being useful if more than 30 seconds pass (see Figure 2).

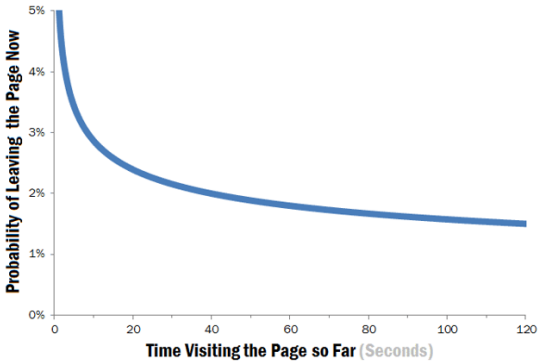

Fig. 2. Probability of leaving a page in relation to elapsed time [14].

As such the extension will call the server once when a new visit has occurred, and if the user stays more than 30 seconds on the page, it will call the server again so as to register a positive feedback for the page. The feedback value itself is a binary value where 0 means not received positive.

When a clustering request occurs the feedback of a document is calculated according to the following function:

$$Feedback(doc_j) = \frac{\sum_{v_i \in V_j} f_{v_i}}{N_{V_j}} \tag{1}$$

where f_{v_i} is the feedback value for a visit v_i of doc_j, V_j represents all the visits for doc_j and N_{v_j} is the total number of visits for that document.

Processes and services used by the web server Besides the main node process which runs the server and listens for incoming requests, the application makes use of four Java processes and a commercial web service for extracting text content from an HTML document. In our first implementation we made use of the free web service provided by AlchemyAPI which specializes in analyzing unstructured content. Regarding the four emplyed Java processes these are:

1. PDF Text Extractor - makes use of the PDFBox library from Apache [16];
2. Text Summarizer - extracts a sentence based summary using a specialized algorithm such as LexRank [4];
3. Text Processor - extracts an "improved" feature vector from the document;

4. Clustering Processor - responsible for closed frequent termset ming and hierarchical document clustering.

3.3 iFIHC - improved Frequent Itemset Hierarchical Clustering

Improved preprocessing of the text documents The preprocessing stage of any document clustering algorithm is extremely important and can profoundly affect the result of the clustering. All preprocessing and selection steps in document clustering try to eliminate "noise" from the texts. Basically they try to minimize the existence of particles of text that that do not carry useful information while trying to reduce the dimensionality of the input data by finding and eliminating informational redundancy. This is usually done though stop word removal, stemming and vectorization.

Though these steps are standard with most document clustering algorithms, we suggest modifying the part related to stemming with a combination of lemmatization and stemming especially since our objective is to obtain a friendly and easy to navigate hierarchical tree where each node has as label a series of terms.

Though stemming and lemmatization are both cases of word normalization which try to reduce inflectional and related forms of a word to a common base form (cars => car), they differ quite a lot. Stemming uses a heuristic for chopping off the end of the words and often includes the removal of derivational affixes. On the other hand, lemmatization makes use of a dictionary and the morphological analysis of the words in hope of returning the dictionary form of a word which is also known as a lemma.

Stemming reduces words that are related derivationally while lemmatization only reduces the inflectional forms of a given lemma. In other words stemming would reduce something like "evolved" and "ponies" to "evolv" and "poni" while lemmatization would result in evolve and pony (see Figure 3).

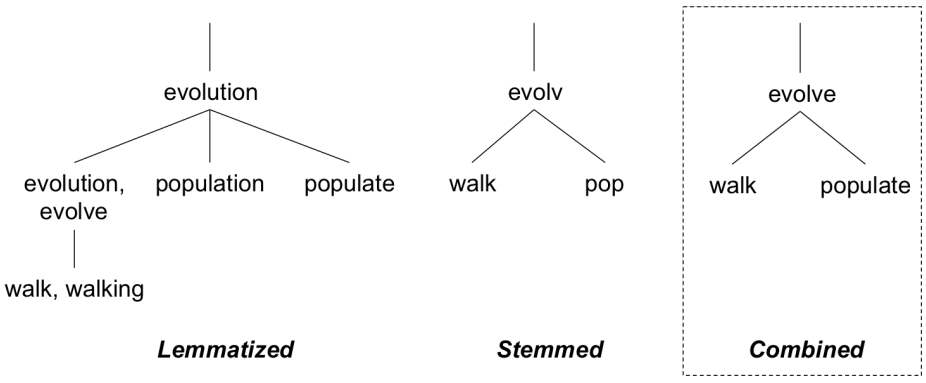

Fig. 3. Difference between lemmatization, stemming and combining them

Still, using a lemmatizer usually doesn't result in much dimensionality reduction since it would keep the verb "cook" and the nominalized noun "cooking" distinct whereas they could be reduced to the same base form namely "cook" which is the shortest. That is why we propose to reduce all the words to their lemma form while associating them with their lemma stemmed form. This way we can reduce words to a unique valid root form. The lemmatization is done based on the WordNet dictionary for the English language [13].

Custom FIHC There were a lot of drawbacks with using the clustering steps in the original algorithm considering our application requirements (some degree of overlapping, accuracy, improved readability, improved speed, etc). That is why we propose using a modified version we call iFIHC that uses *DCI Closed* instead of Apriori and our combined stemmer and lemmatizer approach.

Cluster Construction. The first part is similar to FIHC with a slight difference in that the documents that do not meet any filters supplied by the user such as popularity, visiting time or feedback value are removed along with any closed termset that remains without any associated documents.

Concerning the disjoint step our custom implementation, instead of using the same Score function that needed the cluster support parameter we decided to use the equation from [17], namely:

$$Score(C_i, doc_j) = \sum_{t_k \in C_i} \frac{doc_j \times tf(t_k, doc_j)}{length(C_i)}, \tag{2}$$

where tf is an augmented term frequency used for preventing a bias towards larger documents and is calculated based on the function:

$$tf(t_k, doc_j) = 0.5 + \frac{0.5 \times freq(t_k, doc_j)}{max\{freq(w, doc_j), w \in doc_j\}}. \tag{3}$$

Based on the values of the new Score function, we only keep the duplicates that are above or equal to the mean of all scores for doc_j. After removing the less relevant duplicate documents and all empty clusters, we try to improve the quality of the cluster labels by removing all irrelevant terms.

We resort to this in order to further eliminate noise since the closed frequent itemsets are mined by presence and not TF or TFxIDF. As such for every cluster we calculate the mean of the augmented term frequency values and eliminate all those below it. This is done by repeatedly eliminating the terms below the mean, until the difference between the maximum value and the mean is smaller than δ standard deviations. In our experiments setting δ to 1.2 yielded good results. This step could also be applied before duplicate removal as well.

Tree Building. Our implementation works in a top-down fashion starting from the first level with clusters containing the smallest number of terms and adding new nodes as the length of the cluster label increases.

A very important difference between this phase and the one in the original FIHC is that when for a cluster we find a parent, a copy of that cluster will be assigned to the parent. This is done so we really have overlapping not just for documents but also for entire clusters. After the cluster tree has been constructed each cluster obtains the document indexes of the children. These indexes will be used with our experiments concerning the quality of the clustering processes.

4 Experimental results

For testing the performances of the algorithms used, we ran them on local platform based on an Intel i5 430M processor with 8GB of RAM and a 64-bit Windows 7 operating system.

Regarding the minimum support for mining the closed frequent termsets, we choose a minimum support of 6% for our tests.

In evaluating the cluster quality we used as comparison the values obtained by [17] since they are the most recent. As such we have tested the algorithm out on the same Re0, Wap, and Ohscal datasets[1].

Regarding the way we can evaluate clustering quality for a hierarchical document clustering algorithm that allows overlapping, we use the same measurement functions as for the original FIHC [5], more specifically by overall F-measure given by the following equations:

$$Recall(K_i, C_j) = \frac{n_{ij}}{|K_i|} \tag{4}$$

$$Precision(K_i, C_j) = \frac{n_{ij}}{|C_j|} \tag{5}$$

$$F(K_i, C_j) = \frac{2 \times Recall(K_i, C_j) \times Precision(K_i, C_j)}{Recall(K_i, C_j) + Precision(K_i, C_j)} \tag{6}$$

$$F(C) = \sum_{K_i \in K} \frac{|K_i|}{|D|} \max_{C_j \in C} \{F(K_i, C_j)\} \tag{7}$$

where n_{ij} is the number of documents assigned to cluster C_j that are also present in the natural class K_i. The function $F(C)$ represents the overall F-measure of the final cluster tree C, as the weighted sum of each of the maximum F-measures for each natural class and cluster.

As it can be seen, the iFIHC algorithm behaves far better than FIHC in terms of accuracy while improving the readability of the end result. For testing the execution speed we experimented with a scaled up $Re0$ dataset and we only evaluated the clustering and tree building steps (see Table 1).

It seems iFIHC scales much better and is a lot faster than the original FIHC which is what was needed by our application (see Figure 4). This is mainly due to the reduced dimensionality of the initial cluster set generated from the closed frequent term sets as well as due to the truncation and elimination of clusters and cluster terms during the cluster construction phase.

[1] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download

Dataset	FIHC	iFIHC
Re0	0.529	0.586
Ohscal	0.325	0.461
Wap	0.391	0.516

Table 1. Cluster quality comparison.

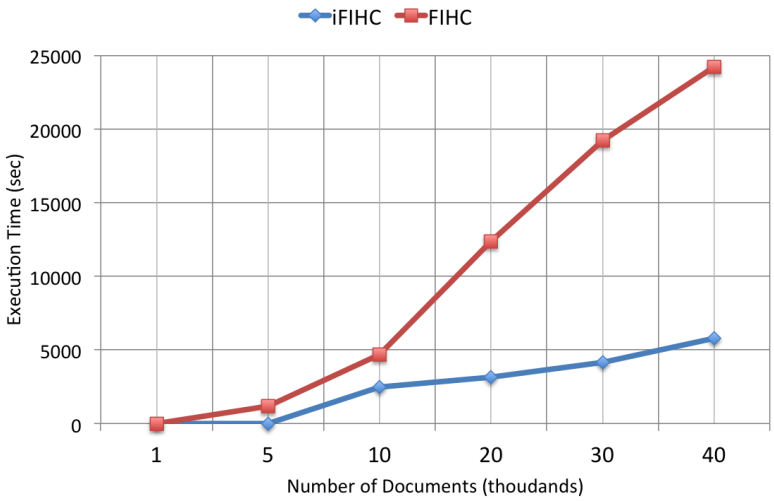

Fig. 4. Clustering speed comparison between FIHC and iFIHC.

5 Conclusions

Given the high need to filter and manage all the user relevant information available on the web we proposed a useful application in the form of a browser extension that can help the user by keeping track of his web experiences, organizing them in an efficient manner and filtering them out according to user specified filters. Moreover the application allows for the enabling of the users to share their web experiences stimulating collaboration where and each of their experiences with the web documents are seamlessly rated by the extension accordingly in relation to the interest they show.

We have provided a viable architecture for the application that takes advantage of cloud computing and available web services in an efficient way (the application is location and machine independent) and uses a custom approach to hierarchical document clustering.

Our approach improves the readability of the cluster labels by combining stemming and lemmatization and improves the overall clustering process so that:

- there is no need for a predefined number of clusters;

- the clustering process is really fast and scalable compared to previous approaches while maintaining accuracy;
- allows a larger degree of relevant overlapping;
- provides more meaningful labels with less redundant terms.

In order to achieve the qualities mentioned we modified the original Frequent Itemset Hierarchical Clustering algorithm by replacing Apriori with an efficient and suitable closed frequent itemset mining algorithm such as *DCI Closed* and by implementing an improved version of the clustering steps called iFIHC that is comparatively faster and at least as accurate as the original.

As future prospects, we will focus on incremental clustering, the usage of sessions, in particular search queries that lead to the visited web pages as additional information, the usage of a reference ontology, and to match pages with it, for a better understanding of the content. The application can be further extended in the sense of web experience sharing as well as in generating even more useful and readable labels for the clusters by replacing them with the greatest common set. Practical application of the work will be analyzed and extended from a third party perspective (e.g., e-commerce vendors, search engine providers, etc.).

Acknowledgment

The research presented in this paper is supported by the following projects: "*KEYSTONE - semantic KEYword-based Search on sTructured data sOurcEs* (Cost Action IC1302)"; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms - PN-II-PT-PCCA-2013-4-0870; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321.

References

1. Agrawal, R., Srikant, R.: *Fast algorithm for mining association rules.* In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pp. 487499. Morgan Kaufmann (1994)
2. Bezdek, J.C.: *Pattern Recognition with Fuzzy Objective function Algorithms.* Plenum Press, New York (1981)
3. Carpineto, C., Osinski, S., Romano, G., Weiss, D.: *A survey of Web Clustering Engines.* ACM Computing Surveys, Vol. 41, No. 3, Article 17 (2009)
4. Erkan, G., Radev, D.R.: *LexRank: graph-based lexical centrality as salience in text summarization.* J. Artif. Int. Res. 22, pp. 457-479, 1 December (2004)
5. Fung, B.C.M., Wang, K., Ester, E.: *Hierarchical document clustering using frequent itemsets.* International Conference On Data Mining (2003)
6. Jain, A.K., Dubes, R.C.: *Algorithms for clustering data.* Prentice Hall (1988)

7. Ji, L., Tan, K., Tung, A.: *Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets*. IEEE Trans. on Knowledge and Data Engineering 19(9), pp. 11751187 (2007)
8. Kwale, F.M.: *A Critical Review of K Means Text Clustering Algorithms*. International Journal of Advanced Research in Computer Science, Vol. 4, No. 9, July-August (2013)
9. Leonard, A.: *Migrate MongoDB Database to Cloud*. In Pro Hibernate and MongoDB, pp. 283-296. Apress, (2013).
10. Lucchese, C., Orlando, S., Perego, R.: *DCI Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets*. Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1 (2004)
11. Marin, A; Pop, F.: *Intelligent Web-History Based on a Hybrid Clustering Algorithm for Future-Internet Systems*, Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2011 13th International Symposium on, pp. 145-152, 26-29 Sept. (2011)
12. Middleton, N. and Schneeman, R.: *Heroku: Up and Running* (1st ed.). O'Reilly Media, Inc, (2013)
13. Miller, G. A.: *WordNet: a lexical database for English*. Communications of the ACM, 38(11), 39-41, (1995).
14. Nielsen, J.: *How Long Do Users Stay on Web Pages*. Web Document. 15 June 2014. http://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/
15. Pasquier, N., Bastide, T., Taouil, R., Lakhal, L.: *Discovering Frequent Closed Itemsets for Association Rules*. Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel, pp. 398416 (1999)
16. PDFBox, Java PDF. "processing Library." Link: http://www. pdfbox. org, (2014).
17. Shankar, R., Kiran, G., Vikram, P.: *Evolutionary Clustering using Frequent Itemsets*. In Proceedings of StreamKDD'10, July (2010)
18. Show-Jane, Y., Yue-Shi, L., Cheng-Wei, W., Chin-Lin, L.: *An Efficient Algorithm for Maintaining Frequent Closed Itemsets over Data Stream. Next-Generation Applied Intelligence*, Lecture Notes in Computer Science Volume 5579, pp 767-776 (2009)
19. Show-Jane, Y., Cheng-Wei, W., Yue-Shi, L., Tseng, V.S., Chaur-Heh H.: *A fast algorithm for mining frequent closed itemsets over stream sliding window*. IEEE International Conference on Fuzzy Systems (FUZZ), pp. 996-1002, 27-30 June (2011)
20. Yu, H., Searsmith, D., Li, X., Han, J.: *Scalable Construction of Topic Directory with Nonparametric Closed Termset Mining*, In Proc. of Fourth IEEE Intl. Conf. on Data Mining (2004)