An MDL-Based Frequent Itemset Hierarchical Clustering Technique to Improve Query Search Results of an Individual Search Engine

Diyah Puspitaningrum[1(✉)], Fauzi[1], Boko Susilo[1], Jeri Apriansyah Pagua[1], Aan Erlansari[1], Desi Andreswari[1], Rusdi Efendi[1], and I.S.W.B. Prasetya[2]

[1] Department of Computer Science, University of Bengkulu (UNIB), Bengkulu, Indonesia
diyahpuspitaningrum@gmail.com, {fauzi.faisal, boko.susilo, jeri.apriansyahpagua, aan.erlansari, desi.andreswari, rusdi.efendi}@unib.ac.id

[2] Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
s.w.b.prasetya@cs.uu.nl

Abstract. In this research we propose a technique of frequent itemset hierarchical clustering (FIHC) using an MDL-based algorithm, viz KRIMP. Different from the FIHC technique, in this proposed method we define clustering as a rank sequence problem of the top-3 ranked list of each itemsets-of-keywords clusters in web documents search results of a given query to a search engine. The key idea of an MDL compression based approach is the code table. Only frequent and representative keywords as those in a KRIMP code table can be used as candidates, instead of using all important keywords from keywords extractor such as RAKE. To simulate information needs in the real world, the web documents are originated from the search results of a multi domain query. By starting in a meta-search engine environment to grab many relevant documents, we set up k = {50, 100, 200} for k-toplist retrieved documents of each search engine to build a dataset for automatic relevance judgement. We implement a clustering technique to the best individual search engine the MDL-based FIHC algorithm with setting of k = {50, 100, 200} for k-toplist of retrieved documents of each search engine, minimum support = 5 for itemset KRIMP compression, and minimum cluster support = 0.1 for FIHC clustering. Our results show that the MDL-based FIHC clustering can improve the relevance scores of web search results on an individual search engine significantly (until 39.2 % at precision P@10, k-toplist = 50).

Keywords: MDL-based FIHC · Frequent itemset hierarchical clustering · KRIMP · Search engine · Relevance score

G. Zuccon et al. (Eds.): AIRS 2015, LNCS 9460, pp. 279–291, 2015.
DOI: 10.1007/978-3-319-28940-3_22

1 Introduction

The World Wide Web contains huge amount of information from around the world. Given a query, existing search engines such as Google, Lycos, Bing, Exalead, and Ask.com return different lists of web search results, ranked by their relevance to the given query. This difference corresponds to different rank algorithms employed by those search engines. Looking for a search engine that has high relevance score then become a need. To overcome this problem one possible solution is by clustering web document to many different groups of topics so a user can match user information need by directly traversing along the intended topic. By clustering web search results of a search engine, the user will quickly find high precision documents. This is the idea behind the cluster utilization in a search engine results [1, 2, 3, 4, 5]. We also know that the major advantage of a meta-search engines is their coverage of multiple search engines [6] hence we can use this to develop a high relevance data set using the Condorcet method [7] as a gold standard for a multi domain query. Furthermore, while FIHC outperforms best existing methods in terms of both clustering accuracy and scalability [8], KRIMP, an MDL-based algorithm, models the database very well [12]. As a result one can use KRIMP to generate only keywords that best represent web documents search results of a given query, and then pass it to FIHC clusters to obtain more relevant results of the query search results.

In this paper, we present an MDL-based approach to web search result clustering (FIHC) that captures associations among keywords of sets of documents extracted from titles and snippets, given a query. Each document URL is then mapped to the most appropriate cluster and the cluster results of triples URL-title-snippet is returned.

This paper provides one main contribution: it shows that an MDL-based FIHC can be used to significantly improve the relevance score of an individual search engine.

2 Related Works

The Minimum Description Length (MDL) principle is a method for inductive inference, or better, for the model selection problem. The basic idea of MDL is to try to find regularity in data. 'Regularity' here may be identified with the 'ability to compress' or viewing learning as data compression. Learning here means finding frequent keywords among web documents that usually go together. For a given set of hypotheses H and dataset D, one should try to find the hypothesis in H that compresses D most.

MDL (Minimum Description Length) is closely related to MML (Minimum Message Length) [9] and also to Kolmogorov Complexity [10]; in fact, one could see MML as fully *Bayesian* MDL. All three embrace the slogan *Induction by Compression*. Below is a brief description of MDL principle.

Given a set of models H, the best model $\mathrm{H} \in H$ is the one that minimises

$$\mathrm{L(H)} + \mathrm{L(D|H)}$$

in which L(H) is the length, in bits, of the description of H; and L(D|H) is the length, in bits, of the description of the data when encoded with H. This is called two-part MDL

or *crude* MDL, as opposed to *refined* MDL, where model and data are encoded together [11]. Refined MDL has a major weakness: it cannot be computed except for some special cases [11]. Hence, for modeling a database we use *crude* MDL, that is for compression purpose. MDL finds the set of frequent itemsets that yields the best compression. In this research a database is a model of a database of frequent itemsets-of-keywords that originated from retrieved web documents search results given a query to a search engine.

Central in an MDL-based approach, viz KRIMP algorithm, is the notion of a *code table*. A code table is a simple two-column translation table that has itemsets on the left-hand side and a code for each itemset on its right-hand side. Using such a table we can encode and decode databases. This is where MDL comes in: we search for the code table that compresses the data best. For further information about code tables in KRIMP see [12].

The approach presented in this paper is an MDL-based frequent itemset hierarchical clustering that is implemented on search results of an individual search engine. It differs from TDPM for evolutionary clustering that divide data into epochs where all data points inside the same epoch are exchangeable and the temporal order is maintained across epochs [13]. Even though the number of clusters produced by an MDL-based FIHC is also unbounded like in TPDM, but we do not divide data into epochs because we are focus in the offline clustering, that is a clustering stage before go to the incremental clustering stage and then to the realtime clustering stage as suggested by Vadrevu et al. [14] to build a scalable clustering system of large collections of documents. Our approach is more inspired by FIHC [8], a robust method in hierarchical clustering, that is more suitable for clustering the web search results rather than the parametric clustering techniques that use a fixed upper bound on the number of clusters.

Our approach also different from a web search clustering technique with richer representation of snippets by discover hidden topics from a very large external data collection such as proposed by Nguyen et al. [15]. Based on Nguyen et al.'s work, there is a need to collect web pages from huge resources and the data must cover many useful topics, hence in our work the focus is more to find associations among important keywords extracted from title and snippet. The MDL-based frequent itemset mining is a suitable technique for finding the association by mining only frequent itemsets that compresses the data best. Then the output of MDL-based technique is clustered using FIHC. We will use the produced clusters to create a list of high relevance retrieved documents.

3 Problem Formalization and Algorithm

3.1 Problem Formulation

We define the general steps of web search clustering of a search engine in two steps:

1. Given a query q, a search engine is used to retrieve a list of results, database $D = (r_1, \ldots, r_n)$; where each r_i is a triple URL-title-snippet with n is the total number of retrieved documents.

2. A clustering C = (C_1, …, C_m) of the results in D is obtained by means of a clustering algorithm.

Consider the requirement of relevance; the problem formulation is:

Given q, produce a set of relevant documents returned by a search engine:

$$\bigcup_{j=1}^{m}\bigcup_{i=1}^{\ell} r_{i,C_j}, \forall C_j, C_{j=1,..,m} \in D$$

where r_iC_j is a triple URL-title-snippet of cluster C_j at rank i-th; ℓ is the maximum number of documents retrieved from cluster C_j; and m is the maximum number of clusters used as search results.

or we paraphrase it as:

given a query q, assume C is the MDL-based FIHC clusters formed from a database D, and D is a list of a search engine search results, then let the system presents a high relevance documents to the user by merge a list of top-ℓ search results documents from m clusters, ordered by first found cluster first out.

3.2 MDL-Based FIHC Steps for Clustering Individual Search Engine

Our MDL-based FIHC algorithm is composed of four steps:

1. *Search result fetching*: Given a query q to a search engine (e.g. Google) we build a database D of web search results returned by the search engine.
2. *Document parsing and keywords extractor*: The obtained webpages' titles and snippets are analyzed by an HTML parser and then passed to the RAKE important keyword extractor [16]. Titles and snippets are informative enough to represent most relevant contents for a given query [14]. The RAKE extractor results in keywords of one or two words; it detects array of words separated by words such as and, the, and of, as its key candidates. Several metrics are used in RAKE: (1) word frequency ($freq(w)$), (2) word degree ($deg(w)$), and (3) the ratio of the frequency and the degree ($deg(w)/freq(w)$), where w is a keyword. For further discussion on RAKE see [16].
3. *MDL-based FIHC clustering*: The database of RAKE keywords named DB, ($DB \subset D$), is then compressed by KRIMP to produce a code table $CT(DB, q)$. Only keywords that exist in CT will be used for clustering, since they are frequent and compress the database DB very well, which means that the resulting CT is a good representation of the keywords of all retrieved documents in a search engine's results of a given query q. As a $CT(DB, q)$ exists we then use it to form clusters using FIHC (see more detail of MDL-based FIHC clustering in Sect. 4). All the formed clusters is written to the *final table*. For simplicity, there is no ranking within clusters.
4. *Post-processing*: Once a final table is created, the post-processing involves a merging of top-ℓ search results documents from m clusters, ordered by first found cluster first out. The merging results is the final search results of the search engine.

4 MDL-Based FIHC Clustering

The KRIMP algorithm [12] uses a code table $CT(DB, q)$ that best describes the database DB, ($DB \subset D$). To do this, one should find the minimal coding set [15].

Let I be a set of items and let D be a dataset over I, *cover* a cover function, and F a candidate set. *Cover* is required to identify which elements of CT are used to encode keywords of a web document given a query q. The result of a cover function is a disjoint set of elements of CT that cover the keywords over q. Minimal coding set is found by seeking the smallest coding set $CS \subseteq F$ such that the corresponding code table CT has a minimal total encoded length $L(D|CT)$. The search space for seeking the optimal code table is far too large, therefore the KRIMP algorithm [12] uses a heuristic, a simple greedy search strategy:

1. Start with a standard code table ST, containing the singletons only of itemsets $X \in I$.
2. Add the itemsets from F one by one. If the resulting codes lead to a better compressed size, keep the code. Otherwise, discard the code.

See further in [12].

Figure 1 presents the MDL-based FIHC clustering algorithm in action when mining "all interesting" itemsets. The algorithm starts with mining large 1-itemsets and using a very low minimum support obtained by trial i.e. *minsup* = 5, KRIMP results in best quality itemsets. Mining stops when "all interesting" itemsets are found. All interesting frequent itemsets are then saved in a code table. We propose the MDL-based frequent itemset mining method (KRIMP) to produce inputs to the FIHC hierarchical clustering because we want to show that KRIMP is also represents the web search results' database best. KRIMP produces best quality frequent itemsets by rejecting frequent itemsets-of-keywords with minimum support lower than *minsup*.

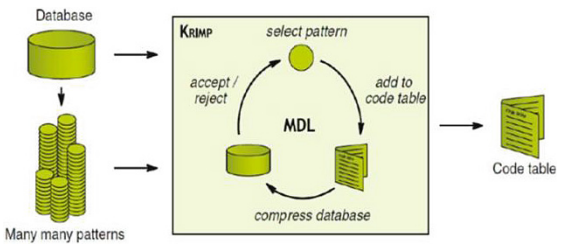

Fig. 1. An MDL-based FIHC clustering algorithm (KRIMP algorithm) [12]

The method for producing clusters, the FIHC technique [8], is "cluster-centered" in that the "cohesiveness" of a cluster is measured directly using frequent itemsets. Since the documents under a cluster contains the same topic, they are expected to share more common itemsets than those under different cluster.

Here we describe the FIHC process after KRIMP produces the code table of many frequent itemsets of important keywords produced by RAKE. Once a code table CT (DB, q) exists, we then use it to form clusters using FIHC [8]. In FIHC there are 3 main

processes: clustering, tree building, and tree pruning. There are two steps to construct clusters: first, constructing initial clusters, then, second, making initial clusters disjoint. An initial cluster is constructed for each global frequent itemset to contain all documents contain the itemset. FIHC only allow any clusters that have cluster frequent items with cluster supports more than the minimum cluster support. In second step, FIHC identify the "best" initial cluster and keep a document only in the best initial cluster by measuring the goodness of a cluster for a document. In tree building process, cluster tree is built bottom-up by choose the "best" parent among such potential parents. The tree pruning criterion is based on the inter-cluster similarity between a parent and its child. A child will be pruned only if the child (the subtopic) is similar enough to its parent topic. Once final clusters are formed after tree pruning process, the next step is creating a set of *relevant* documents as final answer to the given q. This is done by merge top-ℓ search results documents from m clusters.

5 Experiments

We performed experiments in two steps: first, we build a gold standard, viz., the relevant search results; and second, we build a list of documents with high score of relevance. To do the first step, a meta-search engine environment is used to build 4 data fusions that then elected using Condorcet, an automatic relevance judgement suggested by Nuray and Can [7]. For the second step, extensive experiments of the MDL-based FIHC technique are performed on an individual search engine.

Table 1. The multi domain queries [18]

Two terms queries	Three terms queries
Database overlap	Comparative education methodology
Multilingual OPACs	Java applet programming
Programming algorithm	Indexing AND digital libraries
Road-map plan	Geographical stroke incidence
Adolescent alcoholism	Culturally responsive teaching

Table 1 shows queries taken from Mohamed [18] that we use for all of our experiments including both of the two steps above. According to Jansen et al. [19], 97 % of internet queries consist of less than 6 *terms*. Even more, the average length of those queries is 2.5 *terms*. For our experiment, we use queries with length of 2 and 3 *terms* and expand them using operator AND/OR. Therefore using, for example, a three terms query "culturally responsive teaching" will result several combinations: "culturally AND responsive AND teaching", "culturally AND responsive OR teaching", "culturally OR responsive AND teaching", and "culturally OR responsive OR teaching".

5.1 Building the Gold Standards

A meta-search engine allows us to search multiple search engines fast at once, returning more comprehensive and relevant results [20]. We use this ability to build different ideal relevant search results (the data fusions). The data fusions then are used to build gold standard for each query using an election technique namely *Condorcet* (Nuray and Can [7]). Our meta-search engine uses 3 out of 5 popular search engines: Google, Bing, AskJeeves, Lycos, and Exalead.

In the Condorcet method [7], voters rank the candidates in the order of preference. The vote counting procedure then takes into account each preference of each voter for one candidate over another. The Condorcet voting method specifies the winner as the candidate, which beats each of the other candidates in a pair wise comparison. The Condorcet method is chosen as an automatic relevance judgement technique in our experiments since for data fusion the Condorcet method provides the best performance in terms of automatic ranking compares to the random selection method or RS, and the reference count method or RC. According to [7], the mean correlation values for Condorcet is 0.560, compares to 0.506 for RC and 0.493 for RS.

For our Condorcet gold standard [21], we built the gold standard dataset using 4 data fusions. Each data fusion is built in a meta-search engine setting using all combinations of 3 out of 5 component engines using a rank fusion algorithms. The four rank fusion algorithms used are: KE [17], two variants of Weight Borda-Fuse [22, 23], and Count Function algorithm [24]. The choise of k-toplist search results (k = 50, 100, 200) that is used by 3 component engines will produce different ranking of retrieved documents in the data fusion. The KE, the two variants of Weight Borda-Fuse (WBF), and the Count Function algorithms act as "the voters", and their union of 10-toplist documents act as "the candidates" in Condorcet. As a gold standard dataset we only take the 10-toplist documents of the Condorcet results.

5.2 The Unclustered Web Search Result Experiments

Google is a good sample of individual search engines. From Table 2, the Google search engine in average above all queries shows best relevant search results.

Table 2. Relevance of web search results (averaged over all queries)

Search engine	Evaluation metrics					
	k-toplist	P@1	P@3	P@5	P@10	MRR
Google	50	0.6003	0.5353	0.4584	0.3203	0.3806
Bing	50	0.3418	0.2476	0.2203	0.1734	0.29
Ask.com	50	0.1591	0.245	0.2387	0.1972	0.3125
Lycos	50	0.5312	0.4403	0.3985	0.2869	0.3807
Exalead	50	0.2683	0.1343	0.1116	0.0679	0.1183

To measure the relevance of search results, we adopted the Precision at rank n (P@n) metric and the Mean Reciprocal Rank (MRR) as in [25]. Precision at rank n is defined as the proportion of retrieved documents that is relevant with the gold standard,

averaged over all documents. MRR measures where in the ranking the first relevant document (with the gold standard) is returned by the system, averaged over all the web documents. This measure provides insight in the ability of the system to return a relevant document at the top of the ranking.

As an example the query is "adolescent AND alcoholism" built using Google, Bing, and Lycos. Table 3 shows an evaluation result using Condorcet (*50*-toplist), for each original Google, Bing and Lycos web search results; these are before a clustering technique is applied to the individual results of the search engines. In this example Google outperforms other search engines.

Table 3. Relevance of web search results (query "adolescent AND alcoholism", 50-toplist)

Search engine	Evaluation metrics					
	k-toplist	P@1	P@3	P@5	P@10	MRR
Google	50	1.0	0.33	0.6	0.5	1
Bing	50	0.0	0.67	0.4	0.3	0.5
Lycos	50	0.0	0.0	0.2	0.4	0.2

5.3 The MDL-Based FIHC Algorithm Experiments

In our experiments, we cluster Google's search results using *k-means* or the MDL-based FIHC with different number of clusters. For MDL-based FIHC we set $\ell = 1$ and $m = 30$, while for *k-means* $\ell = 3$ and $m = 10$. Number of clusters in *k-means* is set to 10 clusters per query search due to we have enough documents from *k*-toplist a search engine's search results $\{k = 50, 100, 200\}$. Number of clusters in FIHC is not defined by the user but it is generated automatically by the hierarchical FIHC clustering by measuring "cohesiveness" of frequent itemsets to a cluster. As an example is for a search results of query "adolescent AND alcoholism" from a search engine with *k*-toplist ($k = 200$), minimum cluster support = 0.1 and 1533 keywords, from 200 retrieved documents at initial stage of the FIHC we have 1535 clusters that then they drop to 200 clusters at final stage. From hundreds of clusters we take only the 30 first clusters of the hierarchical FIHC final result since when we observed data the number of document in each cluster almost all is 1. For relevance evaluation we test the FIHC final result using only the 10 first clusters due to the Condorcet dataset is also 10 documents. Our search engine prototype displays only 10-toplist of search results.

In final result of hierarchical FIHC we have 30 documents. In *k-means*, since the number of clusters is constant equal to 10 clusters, then we must generate the same number of documents (30 documents) as the final result by picking 3-toplist of documents from each clusters.

For all of our experiments, relevance is measured using only the 10-toplist of a final result's documents of either *k-means* or hierarchical FIHC against the Condorcet dataset as the gold standard.

Here is how we set up parameters in the MDL-based FIHC. In the MDL-based FIHC clustering stage, a parameter for KRIMP we must set up is: the candidate type of itemset collection. For any candidate type of itemset collection we set all itemsets to

minimum support equal to 5 ("*all-5d-pop*"). The choice of "*all*", and not "*closed*" frequent itemsets, is because we want to mine all interesting itemsets. We use a small value for the minimum support due to our web search results database is a sparse database, contains important keywords generated from RAKE. In the FIHC clustering stage, we must define two parameters: the *minimum global support* and the *minimum cluster support*. The minimum global support is 0 (ignored) because, in general, majority of the data are itemsets with global frequent itemsets' global support is below 1 %. If for example we set the minimum global support to 1 we will lose so many global frequent itemsets data and it is not good for further clustering process. The minimum cluster support is set to 0.1. The *cluster support* of an item in C_j is the percentage of the documents in C_j that contain the item. The minimum cluster support value must be set properly depend on the data. In our case, by setting minimum cluster support to 0.1 we still have enough cluster frequent items with cluster supports ≥10 % in initial clusters while at the same time remove many unimportant frequent items with cluster supports below 10 %.

We evaluate the performance of all clustered search results of an individual search engine to the gold standard (Condorcet dataset). See Table 4 and Fig. 2.

Table 4. Relevance of web search results on Google (averaged over all queries)

Search engine	Evaluation metrics					
	k-toplist	P@1	P@3	P@5	P@10	MRR
Google	50	0.5567	0.4822	0.3993	0.2933	0.2949
k-means	50	0.4778	0.4167	0.3456	0.2656	0.6613
Hierarchical	50	0.7840	0.6605	0.5420	0.3920	0.8610
Google	100	0.5300	0.4589	0.3867	0.2863	0.3015
k-means	100	0.5278	0.3519	0.3456	0.2461	0.6682
Hierarchical	100	0.7368	0.6228	0.5237	0.3908	0.8328
Google	200	0.5200	0.4378	0.3673	0.2743	0.3116
k-*means*	200	0.3111	0.2537	0.2400	0.1794	0.5033
Hierarchical	200	0.7143	0.6667	0.4571	0.3286	0.8143

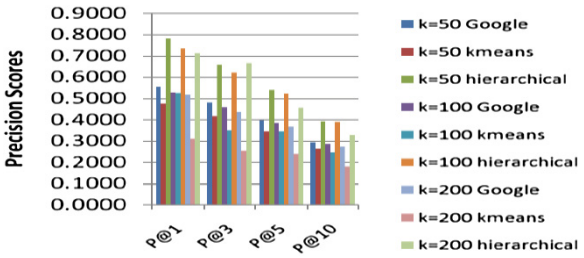

Fig. 2. Performance comparison for different web search results settings of Google. Figure 2 is taken from Table 4.

The "Google" rows show the relevance of the original (and unclustered) search results obtained directly from the Google search engine, The "*k-means*" rows show the results' relevance when the *k-means* clustering is applied, and the "hierarchical" rows show what we get when the MDL-based FIHC clustering is used.

From Fig. 2 and Table 4, we have the following observations.

1. By P@x evaluation, the MDL-based FIHC clustering outperforms other standard clustering algorithms.

Clustering web search results using *k-means* are not recommended since it performs worse than the original search results version of the search engine.

2. By P@x evaluation, the MDL-based FIHC clustering always performs better than the unclustered version, the original web search results of individual search engine.

Even at P@10 the MDL-based FIHC clustering shows significant improvement in relevance or precision than the original one with best setting k-toplist at $k = 50$ of web search results (improvement up to 33.65 % of original Google results). Adding more documents (k-toplist, $k = \{100, 200\}$) are not necessary in improving relevance.

3. By MRR evaluation, the first found relevant document rank position was significantly improved using the MDL-based FIHC clustering.

In this case the MDL-based FIHC clustering outperforms *k-means* in first relevant position is found. This also shows that there is no correlation between improvement in MRR and improvement in precisions of web documents (P@x).

Example results of the gold standard (*Condorcet*), Google, the *k-means* clustering on Google, and the MDL-based FIHC clustering on Google, can be found in the Appendix at https://www.dropbox.com/s/vfc4xobyw83yaax/Appendix.docx?dl=0.

Several statistical Welch two sample t-tests are done on the lists of P@10 values of Google, *k-means*, and hierarchical (the MDL-based FIHC). The aim is to show that there is significant difference between final search result of an individual search engine before and after it is clustered (Table 5). All tests uses confidence level = 0.95 are performed on a hierarchical clustered Google against its corresponding unclustered Google (hierarchical-vs-Google) and done also on a *k-means* clustered Google against its corresponding unclustered Google (*k-means*-vs-Google); all tests are performed 3 times (3 k-toplist, $k = \{50, 100, 200\}$). The detail lists of P@10 of experiments in Table 4 are not shown here due to space limitations.

See Table 5, test results of an individual search engine (clustered vs unclustered search results). Let μ_1 be the mean of a clustered individual search engine's search results and μ_2 the mean of the original unclustered search engine's search results. The hypotheses of interest are expressed as: Hypothesis 1: $\mu_1 - \mu_2 = 0$, Hypothesis 2: $\mu_1 - \mu_2 \neq 0$. The t-tests show there is significant difference between final search result of an individual search engine before and after it is clustered (Table 5, part A and B). To be more specific, by modify the Hypothesis 2 to greater or less to 0, the tests results show hierarchical-vs-Google has evidence that $\mu_1 - \mu_2 > 0$ while *k-means*-vs-Google has evidence of $\mu_1 - \mu_2 < 0$. Since Table 5 part C and part D show the p-value is very low, then we reject the Hypothesis 1. From Table 5 part C, for hierarchical-vs-Google the results show that there is strong evidence of a mean increase in P@10 between

Table 5. The Welch two sample t-tests on P@10 of an individual search engine

A. hierarchical-vs-Google *Hypothesis 1:* $\mu_1 - \mu_2 = 0$; *Hypothesis 2:* $\mu_1 - \mu_2 \neq 0$						
k-toplist	*T*	*df*	*p-value*	*Mean(X)*	*Mean (Y)*	*|Mean (X)-Mean (Y)|*
50	6.34	407	6.22e-10	0.392	0.293	0.099
100	6.69	387	7.87e-11	0.391	0.286	0.105
200	1.45	7	4.21e-13	0.329	0.274	0.055
B. *k-means* -vs-Google Hypothesis 1: $\mu_1 - \mu_2 = 0$; Hypothesis 2: $\mu_1 - \mu_2 \neq 0$						
50	-1.96	473	0.05104	0.266	0.293	|-0.027|
100	-2.64	447	0.008525	0.246	0.286	|-0.04|
200	-7.46	474	4.21e-13	0.179	0.274	|-0.095|
C. hierarchical-vs-Google Hypothesis 1: $\mu_1 - \mu_2 = 0$; Hypothesis 2: $\mu_1 - \mu_2 > 0$						
50	6.34	407	3.11e-10	0.392	0.293	0.099
100	6.69	387	3.94e-11	0.391	0.286	0.105
200	1.45	7	0.09514	0.329	0.274	0.055
D. *k-means*-vs-Google Hypothesis 1: $\mu_1 - \mu_2 = 0$; Hypothesis 2: $\mu_1 - \mu_2 < 0$						
50	-1.96	473	0.02552	0.266	0.293	|-0.027|
100	-2.64	447	0.004262	0.246	0.286	|-0.04|
200	-7.46	474	2.10e-13	0.179	0.274	|-0.095|

search results of the hierarchical clustered Google and the original unclustered Google. It is contrary to the fact to the *k-means*-vs-Google that show a mean decrease (Table 5, part D).

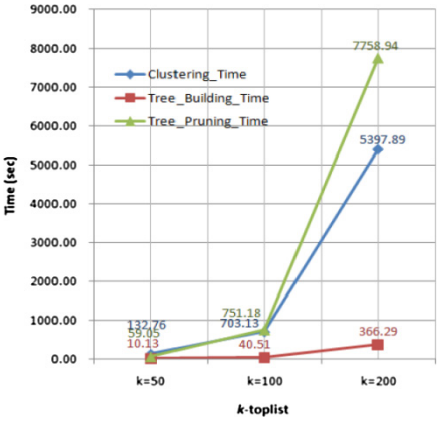

Fig. 3. Scalability of the MDL-based FIHC clustering with the scale-up *Google* web documents search results (minimum cluster support = 0.1)

Figure 3 depicts the runtime of experiments in Table 4 with respect to the number of documents (*k*-toplist, $k = \{50, 100, 200\}$). The whole process completes within 129 min tested on an Asus K45VD laptop machine with processor Intel Pentium

dualcore 2.4 GHz, RAM 4 GB, and hardisk 500 GB. Implementation of the system was built in Python v2.6. The system uses very low minimum cluster support 0.1 since the system is not intended for specific domain only but multi domain that in real situation leads to many very low cluster frequent items and cluster supports (CS). As a consequence it filters them at initial clusters and processes only cluster (label) with CS >= 10 %. It demonstrates that the MDL-based FIHC clustering technique is a scalable method.

Figure 3 also shows that tree pruning and clustering are the most time consuming stages in the MDL-based FIHC. This is different with runtime of tree building that completes in 6.1 min. In the clustering stage, most time is spent on constructing initial clusters while in tree pruning there is an indication of many subtopics are very similar to their parents' topics. The tree pruning scan the tree in the bottom-up oder and calculates *Inter_Sim* similarity between the node and each of its children. If *Inter_Sim* is above 1 then the system prunes the child cluster.

Both in the tree pruning and in the clustering stages, their runtimes are linear with respect to the number of web search results that is handled (k-toplist, $k = \{50, 100, 200\}$ for each settings of individual search engines).

Expensive clustering time as shown in Fig. 3 is due to we performed the MDL-based FIHC clustering viz. KRIMP in sequential. If we split a database of web documents into several smaller databases and feed them into KRIMP, the problem then is about how to merge different results of frequent keywords of each databases in KRIMP code tables into best representative of frequent keywords of the original web documents' database. MapReduce technique is suitable to be implemented in this MDL-based FIHC clustering algorithm. Tree pruning time in future also can be improved by parallelizing processes of computing many *Inter_Sims*. We leave these problems for further research and discussion.

6 Conclusions

We introduced a new approach for frequent itemset-based hierarchical clustering to address the issue of improving the relevance of search results of an individual search engine. The novelty of this research is that it exploits frequent itemsets using an MDL-based algorithm, viz. KRIMP, for defining clusters to generate more relevant retrieved documents from a search engine. Evaluated in a meta-search engine environment datasets, the experimental results show that our approach outperforms other clustering algorithms in terms of relevancy as well as shows significant relevance improvement of web search results of an individual search engine.

References

1. Di Marco, A., Navigli, R.: Clustering and diversifying web search results with graph-based word sense induction. J. Comput. Linguist. **39**, 709–754 (2013)
2. Carpineto, C., Osinski, S., Romano, G., Weiss, D.: A survey of web clustering engines. J. ACM Comput. Surv. **41**, 1–38 (2009)
3. Zamir, O., Etzioni, O.: Grouper: a dynamic clustering interface to web search results. In: 8th International World Wide Web Conference (WWW8), pp. 1361–1374. Toronto (1999)

4. Hearst, M.A., Pedersen, J.O.: Reexamining the cluster hypothesis: scatter/gather on retrieval results. In: 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1996). Zurich (1996)
5. Leuski, A., Allan, J.: Improving interactive retrieval by combining ranked lists and clustering. In: RIAO, pp. 665–681. France (2000)
6. Jadidoleslamy, H.: Search result merging and ranking strategies in meta-search engines: a survey. Int. J. Comput. Sci. **9**, 239–251 (2012)
7. Nuray, R., Can, F.: Automatic ranking of information retrieval systems using data fusion. J. Inf. Process. Manag. **42**, 595–614 (2006)
8. Fung, B.C.M., Wang, K., Ester, M.: Hierarchical document clustering using frequent itemsets. In: SIAM International Conference on Data Mining (SDM 2003), pp. 59–70 (2003)
9. Wallace, C.S.: Statistical and Inductive Inference by Minimum Message Length. Springer, New York (2005)
10. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, New York (1997)
11. Grünwald, P.D.: The Minimum Description Length Principle. MIT Press, Cambridge (2007)
12. Vreeken, J., van Leuwen, M., Siebes, A.: KRIMP: mining itemsets that compress. J. Data Min. Knowl. Discov. **23**, 169–214 (2011)
13. Ahmed, A., Xing, E.: Dynamic non-parametric mixture models and the recurrent Chinese restaurant process: with applications to evolutionary clustering. In: SDM, pp. 219–230 (2008)
14. Vadrevu, S., Teo, C.H., Rajan, S., Punera, K., Dom, B., Smola, A., Chang, Y., Zheng, Z.: Scalable clustering of news search results. In: WSDM 2011. Hong Kong (2011)
15. Nguyen, C.-T., Phan, X.-H., Horiguchi, S., Nguyen, T.-T., Ha, Q.-T.: Web search clustering and labeling with hidden topics. ACM Trans. Asian Lang. Inf. Process. **8**, 12 (2009)
16. Rose, S., Engel, D., Cramer, N., Cowley, W.: Automatic keyword extraction from individual documents. In: Berry, M.W., Kogan, J. (eds.) Text Mining: Applications and Theory, pp. 3–20. John Wiley & Sons, Chichester (2010)
17. Akritidis, L., Voutsakelis, G., Katsaros, D., Bozanis, P.: QuadSearch: a novel metasearch engine. In: 11th Panhellenic Conference on Informatics (PCI 2007), pp. 453–466. Patras (2007)
18. Mohamed, K.A-E-F.: Merging multiple search results approach for meta-search engines. Ph. D. thesis, University of Pittsburgh (2004)
19. Jansen, B.J., Spink, A., Bateman, J., Saracevic, T.: Real life information retrieval: a study of user queries on the web. ACM SIGIR Forum **32**, 5–17 (1998)
20. Meng, W.: Metasearch engines. Technical report, Department of Computer Science, State University of New York at Binghamton (2008)
21. Puspitaningrum, D., Pagua, J.A., Erlansari, A., Fauzi, F., Efendi, R., Andreswari, D., Prasetya, I.S.W.B.: The analysis of rank fusion techniques to improve query relevance. J. Telkomnika **13**(4) (2015)
22. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top-k lists. SIAM J. Discrete Math. **17**, 134–160 (2003)
23. Dorn, J., Naz, T.: Structuring meta-search research by design patterns. In: The International Computer Science and Technology Conference (ICSTC). San Diego, California (2008)
24. Patel, B., Shah, D.: Ranking algorithm for meta search engine. Int. J. Adv. Eng. Res. Stud. (IJAERS) **2**, 39–40 (2012)
25. Sigurbjörnsson, B., van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: WWW, pp. 327–336. Beijing (2008)