A Personalized Search Engine Based on Web-Snippet Hierarchical Clustering

Paolo Ferragina
Dipartimento di Informatica, Pisa
ferragina@di.unipi.it

Antonio Gulli
Dipartimento di Informatica, Pisa
gulli@di.unipi.it

ABSTRACT

In this paper we propose a hierarchical clustering engine, called SnakeT, that is able to organize on-the-fly the search results drawn from 16 commodity search engines into a hierarchy of labeled folders. The hierarchy offers a complementary view to the flat-ranked list of results returned by current search engines. Users can navigate through the hierarchy driven by their search needs. This is especially useful for informative, polysemous and poor queries.

SnakeT is the first complete and open-source system in the literature that offers both hierarchical clustering and folder labeling with variable-length sentences. We extensively test SnakeT against all available web-snippet clustering engines, and show that it achieves efficiency and efficacy performance close to the best known engine Vivisimo.com.

Recently, personalized search engines have been introduced with the aim of improving search results by focusing on the users, rather than on their submitted queries. We show how to plug SnakeT on top of any (un-personalized) search engine in order to obtain a form of personalization that is fully adaptive, privacy preserving, scalable, and non intrusive for underlying search engines.

SnakeT is available at `http://snaket.di.unipi.it/`.

Categories and Subject Descriptors

H.3 [**Information Storage And Retrieval**]: Content Analysis and Indexing, Information Search and Retrieval, Online Information Services; I.5.3 [**Text Processing**]: Clustering

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Web Snippets Clustering, Search Engines, Information Extraction, New Search Applications and Interfaces, Personalized Web Ranking

1. INTRODUCTION

Web-snippet clustering is an innovative approach to help users in searching the web [24]. It consists of clustering the *snippets*[1] returned by a (meta-)search engine into a *hierarchy of folders* which are *labeled* with variable-length sentences. The labels should capture the "theme" of the snippets (and thus, of the corresponding web pages) contained into their associated folders. This labeled hierarchy offers a complementary view to the flat-ranked list of results returned by current search engines. Users can exploit it by *navigating* through the hierarchy of labeled folders, driven by their search needs. This technique is useful for informative [5], polysemous or poor queries.

Web-snippet clustering is a challenging variant of classical clustering because the hierarchy of labeled folders reflects in an intelligible way the different and potentially unbounded "themes" of the snippets returned by the queried search engine(s). This induces two demanding requirements: (i) The folder hierarchy must be formed on-the-fly from the snippets, whereas canonical clustering is persistent since "the folder structure is generated only once, and folder maintenance can be carried out at relatively infrequent intervals" [27]. (ii) The folder must be labeled with meaningful sentences drawn on-the-fly from the snippets. Any "fixed set of category labels" as used in [3, 7] would be not flexible enough to capture the snippets' themes; moreover, due to computational reasons, the clustering engine must process only the (short and thus poor) snippets and not their corresponding (long and thus informative) originating web pages.

Various industrial systems implement web-snippet clustering in their (meta-)search engines: Vivisimo, Mooter, Copernic, iBoogie, Kartoo, Groxis, Dogpile and Clusty. Their efficacy has been recognized with the "best meta-search engine award" assigned by SearchEngineWatch.com to Vivisimo from 2001 to 2003. In January 2005, the AOL portal adopted Vivisimo on top of the search results provided by Google. Also Google and Microsoft seem to be interested into it [28, 29] because *"clustering technology is the* PageRank *of the future"*. Very little information is available about this industrial software. The scientific literature offers several solutions to the web-snippet clustering problem, but unfortunately the attainable performance is far from the one achieved by Vivisimo (see Sect. 2).

Another approach to help users in searching the web is the *personalization* of the flat-ranked lists of query results. Personalized ranking is an intriguing extension of classical link-based ranking that focuses on the users, rather than

[1]The term "snippet" is used here to denote a fragment of a Web page returned by remote search engines and summarizing the context of searched keywords.

WWW 2005, May 10-14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

the query, by combining web-graph link information with some contextual/profiled information. Three requirements of a good personalization should be: *full adaptivity* to the variegate user behaviors/needs, *privacy protection*, and *scalability* to the number of profiles. Examples of industrial personalized services are GOOGLE [1], that collects *category-based profiles* explicitly maintained by the users over a tiny set of categories, YAHOO [2] and EUREKSTER, that require a `login` and build the profiles based on the users' activities. These approaches offer a partial solution because they either allow profiles over a tiny set of choices (GOOGLE) or need to maintain up-to-date profiles which are a critical and private resource. In the scientific literature the personalized ranking problem has been investigated proposing nice scaling techniques to classical approaches. However, these solutions ultimately need to compute, for each web page, a number of ranking values that is related to the number of user profiles (see Sect. 2).

The contribution of this paper is twofold: (i) We propose the first publicly-available software for web-snippet clustering, called SNAKET, that achieves efficiency and efficacy performance close to VIVISIMO; (ii) We exploit the labeled hierarchy of SNAKET to design an innovative form of personalized ranking that achieves full-adaptivity, privacy protection, and scalability. Overall this shows that between ranking and web-snippet clustering does exist a mutual reinforcement relationship from which both of them may benefit. In fact the better is the ranking, the more relevant are the snippets from which SNAKET distills the hierarchy of labeled folders, and thus the better is the personalization.

Our paper offers the following specific contributions:

- We describe the anatomy of SNAKET, the first complete system in the literature that offers both hierarchical clustering and folder labeling with variable-length sentences drawn on-the-fly from snippets. One specialty is that we use *gapped sentences* as labels, namely sequences of terms occurring *not-contiguously* into the snippets. The disclosed softwares do not address all these features together (Sect. 2).
- We suggest the use of web-snippet clustering as a tool for personalized ranking. We show how to plug SNAKET on top of any (un-personalized) search engine in order to obtain a form of personalization. The key idea is that a user issues a query on SNAKET, gets back a labeled folder hierarchy, and then selects a set of folder labels (themes) that best fit his/her query needs. Given this selection, SNAKET personalizes the original ranked list to the selected themes by *filtering out on-the-fly* the snippets which do not belong to the folders annotated by those selected themes. We think that this is an innovative feature offered by SNAKET's interface, in that *it allows to dynamically adapt the ranked list of (about 200 or more) results to the local choices made by any user*. This approach of course does not require an explicit `login` by the user, or a pre-compilation of a constrained user profile, or a tracking of the user's past search behavior, or a modification of the underlying search engines (Sect. 4).
- We provide a complete survey of the current literature and industrial systems (Sect. 2). We then compare SNAKET against the best available systems by executing an extensive set of experiments (Sect. 5). As a further contribution we have built, and offer to the community, a benchmark dataset for web-snippet clustering containing 77 queries, selected from the top searched ones on LYCOS and GOOGLE during 2004.
- We have implemented and engineered a public and open-source prototype that includes all the features above. It runs on top of 16 search engines about Web, Blog, News and Books domains. It has an interface similar to VIVISIMO (see Fig. 1) available at `http://snaket.di.unipi.it/`.

Figure 1: VIVISIMO (left) and SNAKET (right) on the query "*asthma*". Notice on top of SNAKET's window the personalization button.

2. RELATED WORK

The scientific literature offers various solutions to the web-snippet clustering problem. In the simplest case, the folder label is a "bag of words" and the folder clustering is flat. In the more general case, the folder label is a variable-length sentence and the folder clustering is hierarchical. We survey these approaches by classifying them into a taxonomy.

Single words and flat clustering. SCATTER/GATHER [15] was one of the first web-clustering softwares on top of an IR engine. It may be considered to belong to this class even if it was not tested upon a web search engine. WEBCAT [11] uses Transactional K-Means to produce the flat clustering. RETRIEVER [17] uses robust relational fuzzy clustering. The system of [30] expands the set of retrieved snippets with all the in-linking and out-linking pages to improve precision. However search engines do not provide a cheap access to the web graph

thus making the link retrieval efficient if limited to a local (partial) copy available at the clustering engine site. We point out that standard methods such as nearest neighbor and K-means [12], are in this category since they usually exploit single terms as features. Among these softwares only WebCat [11] is available on-line.

Sentences and flat clustering. Grouper [33] was the first publicly available software to address the web-snippet clustering problem. It used sentences of variable length to label the folders, but these sentences were drawn as *contiguous* portions of the snippets by means of a Suffix Tree data structure. Lingo [25] uses SVD on a term-document matrix to find meaningful long labels. The problem with this approach is that SVD is time consuming when applied to a large number of snippets. Recently, Microsoft [34] proposed a system that extracts (contiguous) sentences of variable length via regression on five different measures. However the clustering is flat, regression needs a training phase (hard to adapt on the whole heterogenous web), and the system is not available for testing. There are rumors about the commercialization of this product [29]. Among the softwares of this class, it is available on-line only Carrot2 [31], an open source implementation of Grouper. The original Grouper is no longer available, as the authors communicated to us.

Single words and hierarchical clustering. FIHC [10] uses an analysis based on the Frequent Itemsets Problem in order to construct the folder hierarchy. CREDO [6] uses a concept lattice on single words, and is the only system in this class available on-line.

Sentences and hierarchical clustering. This is the most interesting class including systems that try to mimic Vivisimo. Lexical Affinities Clustering [22] was the first system to propose this approach. It improves precision over recall by using a snippet representation made of *pair of words* (not necessarily contiguous) linked by a lexical affinity, i.e. a correlation of their common appearance. In [33] Etzioni proposed a simple extension of Grouper to hierarchical clustering based on the size of folders overlap. SHOC [35] uses the Suffix Array for (contiguous) sentences extraction and organizes the folders in a hierarchy via an SVD approach. Highlight [32] adopts lexical analysis and a probabilistic framework for hierarchy construction, but the authors do not provide any evaluation. CIIRarchies [21] extracts sentences from the snippets by using a pre-computed language model, and builds the hierarchy via a recursive algorithm. The authors admit that their hierarchies are often non compact, have large depth and contain some non content-bearing words which tend to repeat. [2] Recently, IBM [19] proposed a system that constructs the folder hierarchy based on the minimization of an objective function similar to the one we use in our SnakeT (cfr. Sect. 3.2). However their labels consist frequently of single words, in the other (few) cases they are contiguous sentences. The authors did not make this system available for testing. Surprisingly enough, the only systems of this class available for testing are Highlight and CIIRarchies.

Our SnakeT belongs to this last class, it is highly engineered, available on-line and widely tested, and it aims at overcoming the limitations of the systems above by using gapped sentences as labels, by adopting some special knowledge bases to rank and select the meaningful folder labels, and by building a hierarchy of possibly overlapping folders. We compared SnakeT against the softwares of the fourth class available on line: CIIRarchies, Highlight. We also tested Carrot2 for historical reasons: it offers the only available implementation of Grouper. We did not test the most recent results of [34, 19] because: they didn't provide us with an access to their software, and we could not repeat their experiments because the original datasets are missing and querying the same search engines gives now different snippets (see Sect. 5.2). As industrial engines we compared Mooter and Vivisimo because they are the most powerful web-snippet clustering engines in their categories.

Name	Word Flat	Sentences Flat	Word Hier.	Sentences Hier.	Online/ Software
WebCat	+				+
Retriever	+				
Scatter/Gather	+				
Wang *et al.*	+				
Grouper		+			
Carrot		+			+
Lingo		+			+
Microsoft		+			
FICH			+		+
Credo			+		+
IBM			+		
SHOC				+	
CIIRarchies				+	+
LA				+	
Highlight				+	+
SnakeT				+	+
Mooter			+		+
Vivisimo				+	+

Figure 2: Taxonomy of current solutions.

Personalized ranking algorithms. The scientific literature offers few solutions for the personalized ranking problem. [14] uses the pages contained in the web-directory Dmoz.com to modify the random jump of Pagerank towards a specific topic. The problem with this approach is that one needs to compute for each page, and for each topic, a different Pagerank value. A partial solution to this scaling problem was given in [16], where the dependance from the number of topics was reduced to be sub-linear but still growing with them. [8] suggests to gather user profiles by exploiting the user navigation sessions captured by a proxy, and then applies [16]. Thus all of these solutions offer a limited answer to the features of full adaptivity, privacy protection and scalability required to a good personalization service. The industrial scenario consists of a lab-preview by Google [1] and Yahoo [2], and a full-working engine offered by Eurekster. These approaches however need to maintain up-to-date and profiles, or require an explicit `login`.

3. THE ANATOMY OF SNAKET

Our software SnakeT consists of three algorithmic phases: sentence selection and ranking, hierarchical clustering and labeling, and personalized ranking. The first two phases will be detailed in this section, personalization will be discussed in the next section. See Fig. 3 for a graphical description of SnakeT's architecture.

[2] Pg 91-92 in [20] "A disadvantage present in both snippet and full text hierarchies is that they include uninformative topics. As the hierarchies become further developed, we will look for techniques to identify these uninformative topics."

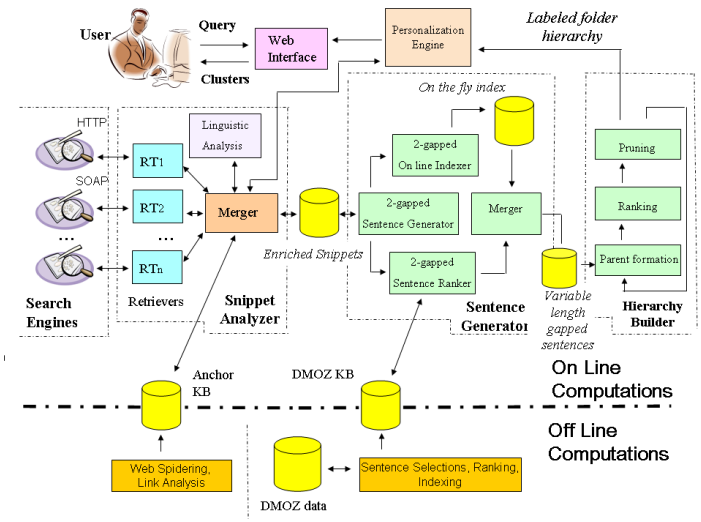

Figure 3: The architecture of SnakeT.

3.1 Sentence selection and ranking

SnakeT uses a frequent itemset-like approach to extract meaningful labels. These labels are drawn on-the-fly from the snippets as *gapped sentences* of variable length. The quality of the extracted labels is improved, and evaluated, by exploiting two knowledge bases, detailed below. This way, SnakeT overcomes the "contiguity limitation" of Grouper's labels and generalizes the notion of Lexical Affinities [22] to more than two words.

Two knowledge bases. The sentence selection process exploits two knowledge bases (KBs) which are built off-line and then used at query time. The former KB is an indexed collection of *anchor texts*[3] extracted from more than 200 millions web pages. The anchor texts of the hyperlinks pointing to a page are used at query time by SnakeT to enrich the content of the corresponding (poor) snippets. The latter KB is a ranking engine over the web-directory Dmoz.com which is freely available, controlled by humans and thus of high-quality. Dmoz classifies more than 3,500,000 sites in more than 460,000 categories, and is used for ranking and retrieval by many web search engines, like Google. We are the first, to the best of our knowledge, to use the whole Dmoz for snippet clustering.[4] Our ranking engine implements a `TFxIDF` measure over pairs of words that is centered on Dmoz-*categories* (unlike the usual document-based view). In detail, let $\#(w)$ be the total number of occurrences of the word w into Dmoz, $\#_C(w)$ be the number of Dmoz-categories in which w appears, and let $\#_C$ be the total number of Dmoz-categories. Moreover let $ns(C_i)$ be a boosting factor for the category C_i that takes into account its depth in the Dmoz-hierarchy (we are postulating an increased importance for deeper categories since they are more specific), and let $b(w, C_i)$ be a boosting factor for the word w if it is placed in a *relevant* part of C_i, such as its description or title (we are postulating an increased importance of terms in crucial parts of the category's description). We then define $\mathrm{TF}(w) = 1 + \log \#(w)$, $\mathrm{IDF}(w) = \log \frac{\#_C}{\#_C(w)}$ and compute the rank of a word w, with respect to a category C_i, as: $\texttt{rank}(w, C_i) = b(w, C_i) * \mathrm{TF}(w) * \mathrm{IDF}(w) * ns(C_i)$. The rank for a pair of words (w_h, w_k) is then defined as: $\texttt{rank}(w_h, w_k) = \max_{C_i}\{\prod_{r=h,k} b(w_r, C_i) * \mathrm{TF}(w_r) * \mathrm{IDF}(w_r) * ns(C_i)\}$.

Query time processing. SnakeT draws about 200 snippets from 16 search engines and enriches them by querying the anchor-text KB. The (enriched) snippets are subsequently filtered against a stop-list, stemmed, segmented into phrases and finally analyzed for extracting Part-of-Speeches and Named Entities. The overall process is called *snippet analyzer* in Fig. 3.

SnakeT then generates the gapped sentences used for the labeling of the folders via an incremental approach (see module *sentence generator* in Fig. 3). Initially, SnakeT extracts from the (enriched) snippets all the pairs of words which occur within some fixed proximity window. These pairs are then ranked using the above Dmoz-based engine and some boosting factors that depend on PoS and NE. Low ranked word-pairs are discarded. The remaining pairs are *incrementally* merged to form longer gapped sentences. We use neither suffix trees nor suffix arrays for word-pairs merging, since in our (gapped) sentences words may occur not contiguously in the snippets. Our approach is therefore based on a combination of inverted lists and bitmaps, built fast and on-the-fly over the snippets. These data structures allow to efficiently and incrementally merge a gapped sentence g with a word pair (w_h, w_k) if they appear in the same snippet and within a proximity window. The resulting longer gapped sentence preserves the original order in which g, w_h, w_k appear in the source snippet. The rank of this sentence is a function of the ranks of its constituting pairs of adjacent words, and is computed by the Dmoz ranking engine. Low ranked sentences are discarded and the process is repeated until no merge is possible or sentences are formed by 8 words (this number is customizable). All the sentences that have been not discarded in whole process provide the *candidate* labels for the annotation of the leaves of the folder hierarchy. We notice that the cost of the merging process is negligible with respect to the other operations, since it is linear in the size of the inverted lists.

3.2 Hierarchical clustering

SnakeT uses an innovative bottom-up hierarchical clustering algorithm whose aim is to construct a folder hierarchy which is *compact* in terms of total number of folders, *balanced* in terms of descending folders, and *overlapping* because a snippet might cover multiple themes. SnakeT also aims at assigning folder labels that are *accurate* with respect to the snippets' themes, *distinct* to avoid an overwhelming repetition in the words, and *intelligible* by means of variable-length sentences. The overall process is called *hierarchy builder* in Fig. 3.

Initially, snippets are grouped into folders according to the (candidate) gapped sentences they share. These folders provide the leaves of our hierarchy, and their labels provide their annotations (called *primary labels*). We are postulating that *snippets sharing the same gapped sentence deal with the same theme*, and thus must be clustered into the same folder.

In order to agglomerate (leaf) folders for the hierarchy

[3]An anchor text is the segment of a web page surrounding an hyperlink.

[4]Unlike [7], we are using Dmoz only for the ranking of the sentences which are extracted on-the-fly from the snippets. Therefore Dmoz is not used for building the folder hierarchy.

construction, SnakeT enriches each folder C with a *set of secondary labels*, defined as gapped sentences that occur in the $c\%$ of C's snippets (currently $c = 80$). The primary label provides a finer description of C, its secondary labels provide a coarser description of the folder's snippets. To manage primary and secondary labels efficiently, we concatenate them into a unique string and use a special character as a separator. This string is called the *signature* of the folder C, denoted by $\texttt{sig}(C)$.

The inductive step of the bottom-up hierarchy construction process consists of three main phases: parent formation, ranking and pruning. A *parent* folder P is created for each group $C_1, C_2, \ldots, C_j$ of folders that share a gapped sentence among their signatures (hence, both primary and secondary labels are deployed). The shared sentence provides the primary label of P, and thus its annotating label $\ell(P)$. The set of secondary labels of P is formed by the secondary labels of the C_i's that occur in at least the $c\%$ of P's snippets. $\texttt{sig}(P)$ is obtained by concatenating $\ell(P)$ with all P's secondary labels. We note that the efficient computation of the shared sentence $\ell(P)$ is done via a Suffix Array built over all folder signatures. Since $\ell(P)$ is computed among the gapped sentences, it is a gapped sentence and is not necessarily a substring of $\ell(C_1), \ldots, \ell(C_j)$.

After the formation of all parent folders and their labels, SnakeT ranks them by exploiting the rank of the labels of their children folders. Indeed, the rank of P is computed from the rank of the labels of the C_is (see Sect. 3.1). Then, SnakeT builds a *weighted bipartite* graph G in which the sets vertices are given by the parent folders (currently under construction) and their children folders, the edges denote the parent-child relationship, and the weight of a vertex is the rank of the corresponding folder.

This graph is exploited in the next pruning phase, whose goal is to clean up the current level of the folder hierarchy in order to match the goals stated at the beginning of the section. SnakeT adopts two different pruning rules to discard some of the currently formed parent folders. The first rule aims at discarding parent folders which are redundant wrt a *graph-covering relation*: if two parent folders cover (almost) the same children folders, then SnakeT keeps the parent folder having the largest rank. The second rule aims at discarding parent folders which are redundant wrt a *syntactic-similarity relation* among labels: if two parent folders are annotated with (almost) the same labeling words, then SnakeT keeps the parent folder having the largest rank. This latter rule takes into account label conciseness, accuracy and distinctiveness. The rules are applied on the weighted bipartite graph via a greedy approach (details in the full paper). The number of the discarded folders is not negligible and their deletion contributes to make the overall hierarchy more intelligible and compact. We note that the graph-covering rule is similar to the one proposed in [19], but here we additionally use gapped sentences as labels and apply the syntactic-similarity relation to obtain better folder labels (see e.g. Fig. 6).

After the pruning, the remaining parent folders provide the next level upon which the bottom-up process is repeated again. The process is currently stopped after that three levels have been built (a deeper hierarchy would be not user friendly). We remark that a snippet may occur in many folders, and this is consistent with the observation that a web page can cover *multiple themes*. Moreover, we observe that the use of gapped sentences allows SnakeT to cluster together two snippets even if some terms are missing, or even, occur in different order within them. For example snippets containing the sentences "John Fitzegerald Kennedy", "Kennedy John" and "John F. Kennedy" would be clustered together by SnakeT.

Fig. 4 reports the time complexity of SnakeT and of the other engines. Experiments showed that on a small number of snippets and extracted labels SnakeT achieves excellent results.

Flat Clustering	Time Complexity	Hierarchical Clustering	Time Complexity
Retriever	$O(nk)$	LA	$O(n^2 \log n^2)$
Wang *et al.*	$O(kn \log n)$	IBM	$O(kn)$
Grouper	$O(n)$	SHOC	$O(n)$
Carrot	$O(n)$		
Microsoft	$O(n)$	SnakeT	$O(n\ logn + m \log mp)$

Figure 4: Notation: n is the number of processed snippets, k is the number of desired folders, m is the number of extracted sentences/words, p is the number of labels extracted by SnakeT.

4. PERSONALIZING SEARCH RESULTS

Link-based ranking approaches tend to produce results which are biased towards the most popular meaning of an ambiguous query: "Jaguar" on Google does not get answers related to the mayan civilization in the first ten results. Conversely, SnakeT is able to distill from the web snippets *few key concepts* (possibly some of low rank, see Fig. 5) that may be subsequently deployed by the user to *personalize* the results produced by the underlying search engines (see Fig. 8). SnakeT exploits the labeled folder hierarchy also for query refinement, disambiguation and knowledge extraction as detailed below (see the module *personalization engine* in Fig. 3).

Figure 5: Knowledge extraction for "jaguar"

Hierarchy browsing for knowledge extraction. Users can navigate through the hierarchy by expanding or collapsing on-the-fly its folders. The expansion is cheap since occurs at the client side. The navigation can be seen as a

form of *knowledge extraction* process that allows the user to acquire several points of view on the 200 or more query results, without the effort of scanning all of them. This is useful because users frequently look at just the first top-ten results of the flat-ranked list. See Fig. 5 where a user learns from the folder labels created for the query "jaguar" that this term refers to: an animal, a car, the mayan civilization, a British rock-band, and Mac OS X.

Hierarchy browsing for results selection. Users can narrow the ranked list of snippet results to those ones which generate a label l, by just clicking on l. This is pretty much similar to what VIVISIMO does, with the speciality that SNAKET does everything at the client side.

Query Refinement. Once the user looks at the folder hierarchy, (s)he can decide to refine the query Q in two different ways. Either (s)he can deploy the folder labels to choose *new keywords* for composing a *new refined query* to be submitted to SNAKET. Or (s)he can choose as *additional keywords* the words of a label l, by clicking on it. In this latter case SNAKET submits automatically the refined query $Q' = Q \wedge l$ to its pool of search engines, and then builds a new folder hierarchy for them. See Fig. 6 where the query "allergy" may be refined as "latex allergy" by clicking onto the label "Relief/Latex Allergy". This is a form of *query expansion/suggestion* used by many commercial search engines, here re-interpreted in the web-snippet hierarchical clustering framework.

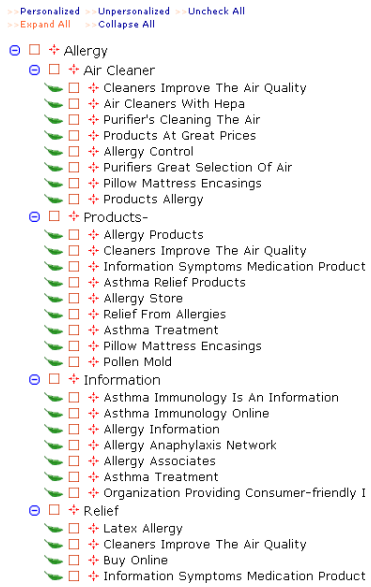

Figure 6: Hierarchy browsing for "Allergy"

Personalized Ranking. Users can select a set of labels $L = \{l_1, \ldots, l_f\}$ and ask SNAKET to *filter out* from the ranked list, returned by the queried search engines, the snippets which do not belong to the folders labeled by L's labels. We think that this is the most innovative feature offered by SNAKET's interface, in that it allows to dynamically adapt the ranked list of (about 200 or more) results to the local choices made by any user. As far we know, we are the first to suggest the use of web-snippet clustering as a tool for personalizing the ranked list of results returned by a (meta-)search engine. This feature turns out to be particularly effective when the users submit informative [5], polysemous, or poor queries. See Fig. 8 for an example in which a user aiming at introductory material about the programming language "java", first formulates the query "java", and then selects the labels "Tutorials" and "Training" for getting *personalized results*.

Figure 7: SNAKET on the query "java"

Figure 8: Personalized SNAKET: the user selects the two labels "Tutorial" and "Training" and gets its *personalized* ranked list.

We refer the reader to Sect. 2 for a discussion of the literature on personalization. We note here that SNAKET's personalization is fully adaptive, scalable, and non intrusive for the user. It is fully adaptive and scalable because it is not profile-based and users can adapt the choice of their selected labels according to their subjective and time-varying interests. SNAKET also protects the user privacy because it does not require an explicit `login`, a pre-compilation of a user profile, and tracking the user's past searches.

Notice that the user can change multiple times the selected labels and thus modify on-the-fly his/her set of personalized results. The filtering is done on all the 200 (or more) snippets returned by the queried search engines. Everything occurs at the client side, thus being computationally cheap. In summary, SNAKET is a plug-in that turns

any un-personalized (meta-)search engine into a personalized one.

Personalized Web Interface. We remark that SnakeT offers a lightweight web-client interface that does not require to maintain any state on the server. For each query, the server performs the hierarchical clustering of the snippets returned by the queried engines and then sends to the web client all the information needed to perform the above tasks via a one-time communication in XML. Folder expansion, browsing and personalization is scalable since they occur at the client side. Conversely, Vivisimo requires a communication between the client and the server for each folder refinement.

5. EXPERIMENTAL RESULTS

SnakeT runs currently on a commodity PC with Linux, P4 CPU and RAM 1.5Gb. This is the system where we executed our tests. For space reasons we report just the most important results, whereas a more extensive testing is available on line, see [9]. See Fig. 9 for time figures.

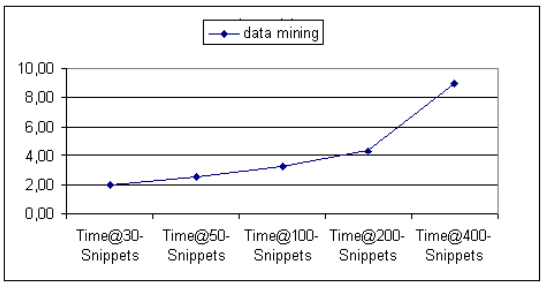

Figure 9: Time (secs) taken by SnakeT to retrieve and cluster a growing number of snippets on the query "data mining".

The literature of web-snippet clustering offers three different methodologies for comparing the systems in Fig. 2: anecdotal evidence for the quality of the results, user surveys conducted on a set of users for a set of queries, and some mathematical functions. There is not a general consensus about the measure to use for evaluating a web-snippet clustering engine. Moreover, although there are many proposals for evaluating flat clustering [13, 23], it is still open the definition of a mathematical evaluation which takes into account the *expressiveness* of the labels within a folder hierarchy. In the following, we evaluate SnakeT by executing some user surveys, by drawing anecdotal evidence of the overall efficacy of SnakeT and of its modules, and by extending the methodology of [34] with the goal of addressing the "label expressiveness" issue. These last two evaluations deploy a unique (in the literature) dataset of snippets, enriched with clustering results, that we have collected from the 16 search engines using 77 queries, selected from the top searched ones on Lycos and Google during 2004. This dataset is available on line [9] and can be used freely by the research community either to reproduce our experiments or to test any new web-snippet clustering engine.

5.1 Users surveys

First Study: Is web clustering beneficial? This study was aimed at understanding whether a Web-snippet clustering engine is a useful complement to the flat, ranked list of results offered by classical search engines (like Google). We asked to 45 people, of intermediate web ability, to use Vivisimo during their day-by-day search activities. After a test period of 20 days, 85% of them reported that using the tool "[..] get a good sense of range alternatives with their meaningful labels", and 72% said that the most useful feature is "[..] the ability to produce on-the-fly clusters in response to a query, with labels extracted from the text". This study confirms the evaluation reported by SearchEngineWatch.com.

Second Study: SnakeT vs other available systems. We selected 18 queries from our dataset belonging to many different topics (*iraq, bush, data mining, bill gates, last minute, car rental, mp3, divx, sony, final fantasy, ipo, equity, google ipo, warterlo, second war, aids, allergy, nasa*), and asked to three users to compare our results against those provided by Mooter, CIIArchies, Highlight, Carrot2 (see Sect. 2). For a large part of the queries the users did not like Mooter, since it provides folders labeled with single words. Carrot2 often tends to create a number of folders which exceeds the number of snippets, thus impacting negatively onto the usability of such software. Carrot2 also fails to cluster together similar labels such as "knowledge, knowledge discovery", "mining and knowledge", and furthermore it labels the hierarchy paths with sentences which are one the substring of the other thus introducing few additional knowledge during the browsing. Highlight obtains its top-level's labels by using classification, so that they are few and of little use. Moreover, its clustering frequently produces identical subtrees under different top level categories and a number of folders which exceeds the number of snippets themselves (e.g 160 folders for the "iraq" query). CIIArchies provides good hierarchies but, as the authors admit, they are often not compact, have large depth and contain some non content-bearing words which tend to repeat. From the point of view of the performance we remark that the two best available tools, CIIArchies and Highlight, are significantly slower than SnakeT. We also remind to the reader that no other system is available on line for comparison, as discussed in Sect. 2, and that our three-users survey is fair because of the objectivity of their negative comments.

Third Study: SnakeT vs Vivisimo. We tried to draw a preliminary evaluation of our software against Vivisimo. We selected 20 students of the University of Pisa and asked them to execute the above 18 queries on these two engines. 75% of them were satisfied of the quality of our folder hierarchy and of its labels. Hence we can state that SnakeT achieves performance close to Vivisimo. See Fig. 10 for details.

5.2 SnakeT's dataset and anecdotal evidence

We have built a dataset consisting of the snippets collected by 16 search engines in response to 77 queries. It is available on-line [9]. Queries are selected among the top searched ones on Lycos and Google during 2004. For each query we retrieved from 180 to 220 snippets. The dataset has been manually annotated by humans who judged the correctness of the results wrt the query. Moreover, the dataset has been enriched with the labels of the folders produced by SnakeT. This way the dataset can be used to infer anecdotal evidence about the quality of the folder hierarchy and to tune

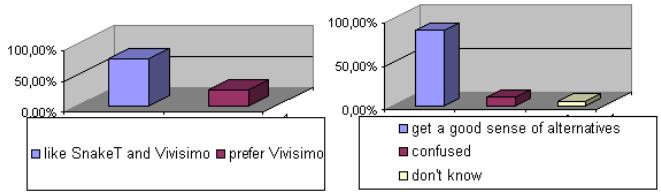

Figure 10: Left: judgement of SnakeT's results. Right: user preferences.

the software modules building up SnakeT (Figs. 5,6,7,8 are extracted from the dataset). As far we know, this dataset is the largest available on line and the only one built over queries retrieved by many search engines. It can be used freely by the research community either to reproduce our experimental results or to test any new web-snippet clustering engine. This dataset is crucial because the web-snippet clustering is a form of ephemeral clustering [22], and thus its results may change over the time as a consequence of a change in the list of snippets returned by the search engines.

5.3 Evaluation of SnakeT

We ran an extensive testing to tune up the different modules composing SnakeT. Because of the 10-page limit we are forced to report here few results, and thus we refer the reader to [26] for a full report. We evaluated SnakeT by using our dataset and a mathematical measure that extends the one adopted in [34] by taking into account the labeled folder hierarchy. We actually evaluated the precision at the first N labels associated to the top-level folders generated by SnakeT for each of the 77 queries. Precision at top N is defined as: $P@N = \frac{M@N}{N}$, where $M@N$ is the number of labels which have been *manually tagged* relevant among the N top-level labels computed by SnakeT. If a label l has been tagged as "ambiguous", we judge l relevant if the majority of its children labels are relevant. We believe that P@N, specialized on the top-level folder labels, reflects the natural user behavior of considering these labels as the most important for hierarchy navigation. We use P@3, P@5, P@7 and P@10 since the lazy users does not like to browse a wider folder hierarchy.

Benefits of using Dmoz. The Dmoz index acts as a ranking engine for driving the selection of the best gapped sentences as folder labels. This engine produces both a significative boost of P@N and an increase in the number of relevant top-level labels. In our experiments we noticed this phenomenon on a very large set of queries (see Fig. 11).

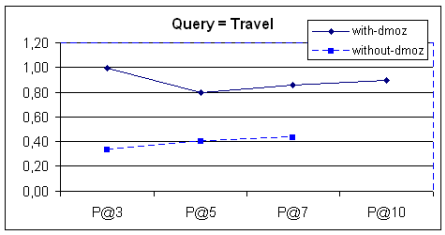

Figure 11: P@N using the Dmoz index

Benefits of using anchor-text index. This index was introduced to enrich the pages that do not have a good textual description, or belong to the web frontier reached by the spiders. We have in this set also the top authorities which are described well by many other sites [18]. In Fig. 12 we report different values of P@N for the query "Yahoo", using or not using the anchor-text index. It may be observed a nice phenomenon common to many queries: The anchor-texts increase P@N for the lower values of N.

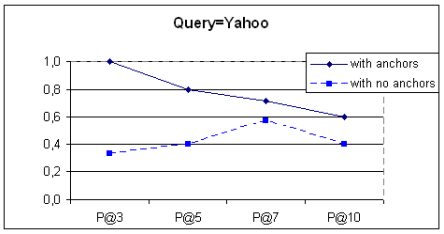

Figure 12: P@N using the Anchor index

Benefits of using multiple engines. Multiple engines offer a better coverage of the web because of the low overlap of current search engines [4]. Usually this is seen as a limitation, rather than a resource, because of the difficulty in combining their multiple ranked lists. Fig. 13 reports a different view on this issue: the use of query results coming from many search engines produces a more detailed and meaningful labeled folder hierarchy. This may better help the user in exploiting the various forms of personalization offered by SnakeT.

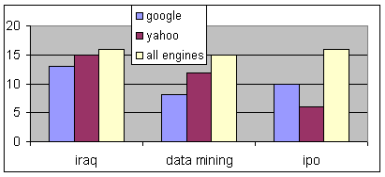

Figure 13: Number of top levels folders

Benefits of using gapped sentences as folder labels. Since some software uses contiguous sentences as folder labels (see Carrot2 and [34]), we tried to evaluate the impact of SnakeT's gapped sentences on the meaningfulness of the selected labels. We studied the distribution of the gaps for the most relevant pair of words within the snippets of our dataset. Fig. 14 reports the distribution, in logarithmic scale, of four relevant word-pairs for the query "car rental" within different text gaps. Due to these experimental observations, SnakeT adopts a maximum gap of four for generating the gapped sentences.

We are left with the study on the overall performance of SnakeT. Here, we comment on the most salient aspects and refer to Table 19 for details.

Label precision over our dataset. For all 77 queries composing our dataset (Sect. 5.2), we achieved an average precision over the top-level folder labels of P@3=91%, P@5=83%, P@7=78% and P@10=79%, see Fig. 15.

Number of top-level labels. SnakeT was tuned to produce only meaningful top-level labels and to discard those

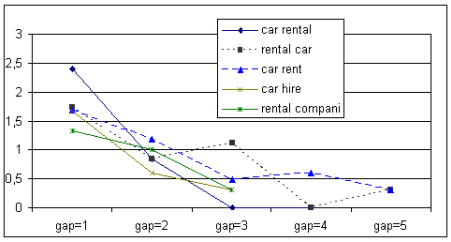

Figure 14: Log-distribution of relevant word pairs

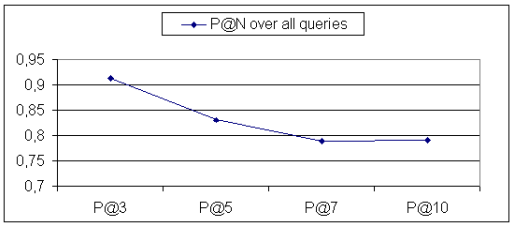

Figure 15: P@N on our dataset

below a fixed rank. Therefore not all the queries produce ten top-level labels (and hence ten top-level folders). In Fig. 16 we report the exact number of generated top-level labels (folders) for all the 77 queries of our dataset. Notice that all queries produce at least three, and many of them produce up to ten, top-level labels.

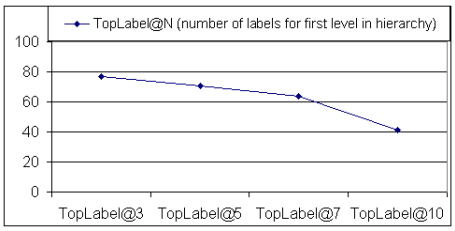

Figure 16: Number of queries generating N top-level labels in our dataset (TopLabels@N)

Precision over the personalized results. In sect. 3.1 we described the ranking adopted by SnakeT to aggregate the results of the 16 search engines, and in sect. 4 we described how SnakeT personalizes the search results. We studied how the precision changes when SnakeT's personalization is applied. We used *over the snippets* the following precision measure: $P@N_{snippets} = \frac{MS@N}{N}$, where $MS@N$ is the number of snippets which have been *manually tagged* relevant among the N top-snippets returned by SnakeT's personalization. We measured an increase in $P@N_{snippet}$ of about 22% averaged over N and over our dataset of queries. In Fig. 17 we compare $P@N_{snippets}$ for personalized vs. unpersonalized results on the query "divx".

Number of web snippets contained into folders. The *weight of a folder* is the number of snippets it contains. A hierarchy is defined *weight balanced* if nodes at the same level have comparable weights. In Fig. 18 we report the distribution of the weights for the top-level folders generated for the

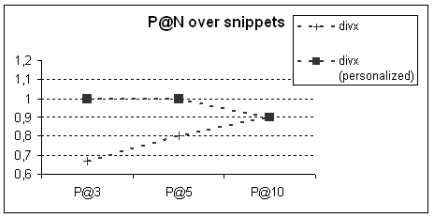

Figure 17: P@N over the snippets for "divx"

query "data mining". For space constrains, we expanded only the top-folder "Software". Notice that SnakeT's hierarchy is balanced, and this phenomenon occurs on many queries of our dataset. We remark that a good balance is crucial for personalization since it enforces the folder hierarchy to equally detail all the concepts behind the query.

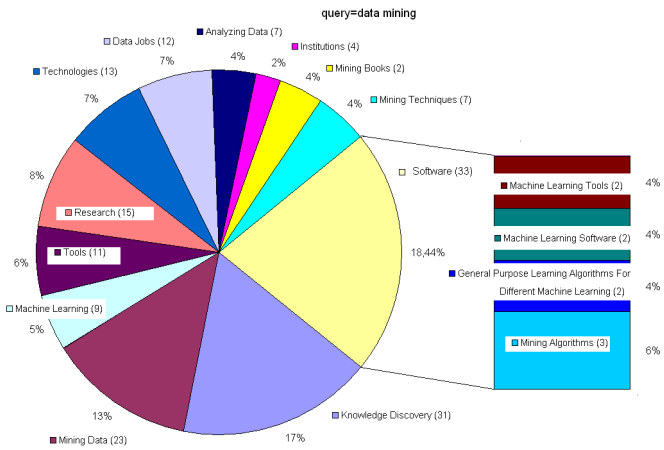

Figure 18: Covering Distribution for "data mining"

6. CONCLUSION

SnakeT is a unifying hierarchical web-snippet clustering system with a web interface for web search, books, news and blog domains. Space constraints prevented us to describe the extension of SnakeT to the last three domains. Readers can nonetheless check on-line these features. Given the extensive experiments executed on SnakeT and the engineering of its modules, we can state that its time performance and quality of the labeled folder hierarchy are comparable to Vivisimo but SnakeT is open-source. In addition, SnakeT provides an innovative form of personalized ranking that is fully adaptive to user needs, privacy preserving and scalable to the number of users.

7. REFERENCES

[1] `http://labs.google.com/personalized`.
[2] `http://mysearch.yahoo.com/`.
[3] G. Attardi, A. Gulli, and F. Sebastiani. Theseus: categorization by context. In *WWW8*, 1999.
[4] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW7*, 1998.

[5] A. Broder. A taxonomy of web search. In *SIGIR Forum 36*, 2002.

[6] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004.

[7] H. Chen and S. T. Dumais. Bringing order to the web: automatically categorizing search results. In *SIGCHI00*.

[8] P. A. Chirita, D. Olmedilla, and W. Nejdl. PROS: A personalized ranking platform for web search. In *Int. Conf. on Adaptive Hypermedia and Web-based Syst.*, 2004.

[9] SnakeT Dataset. `http://roquefort.di.unipi.it/~gulli/listAllowed/testSnakeT/`.

[10] B. Fung, K. Wang, and M. Ester. Large hierarchical document clustering using frequent itemsets. In *SDM03*.

[11] F. Giannotti, M. Nanni, and D. Pedreschi. Webcat: Automatic categorization of web search results. In *SEBD03*.

[12] J. Grabmeier and A. Rudolph. Techniques of cluster algorithms in data mining. In *Data Mining and Knowledge Discovery*, volume 6(4), pages 303–360, 2002.

[13] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. In *JIIS*, 2001.

[14] T. Haveliwala. Topic-sensitive pagerank. In *WWW12*, 2002.

[15] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *SIGIR-96*.

[16] G. Jeh and J. Widom. Scaling personalized Web search. In *WWW13*, 2003.

[17] Z. Jiang, A. Joshi, R. Krishnapuram, and L. Yi. Retriever: Improving web search engine results using clustering. In *Managing Business with Electronic Commerce 02*.

[18] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *JASM*, 1999.

[19] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW13*, 2004.

[20] D. J. Lawrie. *Language Models for Hierarchical Summarization*. PhD thesis, Amherst, 2003.

[21] D. J. Lawrie and W. B. Croft. Generating hiearchical summaries for web searches. In *SIGIR03*.

[22] Y. S. Maarek, R. Fagin, I. Z. Ben-Shaul, and D. Pelleg. Ephemeral document clustering for web applications. Technical Report RJ 10186, IBM Research, 2000.

[23] M. Meila. Comparing clusterings. Technical Report 418, University of Washington, 2002.

[24] Javed Mostafa. Seeking better web searches. *Scientific American*, February 2005.

[25] S. Osinski and D. Weiss. Conceptual clustering using lingo algorithm: Evaluation on open directory project data. In *IIPWM04*, 2004.

[26] SnakeT Test Results. `http://roquefort.di.unipi.it/~gulli/listAllowed/testing/`.

[27] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.

[28] `http://www.searchenginelowdown.com/2004/10/web-20-exclusive-demonstration-of.html`.

[29] `http://www.betanews.com/article/Microsoft_Tests_Search_Clustering/1106319504`.

[30] Y. Wang and M. Kitsuregawa. On combining link and contents information for web page clustering. In *DEXA02*.

[31] D. Weiss and J. Stefanowski. Web search results clustering in polish: Experimental evaluation of carrot. In *IIS03*.

[32] Y. Wu and X. Chen. Extracting features from web search returned hits for hierarchical classification. In *IKE03*.

[33] O. Zamir and O. Etzioni. Grouper: a dynamic clustering interface to Web search results. In *WWW8*, 1999.

[34] H. Zeng, Q. He, Z. Chen, and W. Ma. Learning to cluster web search results. In *SIGIR04*.

[35] D. Zhang and Y. Dong. Semantic, hierarchical, online clustering of web search results. In *WIDM01*.

QUERIES	M@3	M@5	M@7	M@10
adsl	3	4	6	
aids	3	5		
airline flight tracking	3			
allergy	3	5	7	10
asthma	3	5	7	10
athens	2	2	3	
avril lavigne	3	5	6	8
bats	3	4	6	6
bible	3	5	6	8
britney spears	3	5	7	10
bush	3	4		
christina aguillera	3			
data mining	3	5	7	10
david beckham	2	4	6	8
divx	3	5		
dragon ball	3	3	5	8
dvd	3	5	7	10
dylan dog	3	3	3	5
eminem	3	5	7	
fbi	3	5	7	
final fantasy	3	5	7	10
final ipo	1	1	3	
firewall	3	5	7	
google	3	5		
grande fratello	3	5	7	8
guppy	3	3	4	
halloween	3	4	5	6
harry potter	3	5	7	10
hurricane	3	5	7	
ikea	2	2	2	
iraq	3	4	6	9
jaguar	3	4	6	
janet jackson	3	4	4	7
java	3	5	7	10
jennifer lopez	3	5	6	
kazaa	3	4	4	
las vegas	3	5	6	6
madonna	3	5	7	9
marjuiana	2	4	6	9
matrix	2	2	4	7
michelle vieth	3	4	4	
morpheus	3	5	7	9
mousetrap	3	3	5	6
movie	3	5	6	9
mp3	3			
music	3	5	7	10
napster	3	4		
nasa	2			
news	2	3	4	4
new york	3	5	5	6
nokia suonerie	2			
nostradamus	3	4	4	
pagerank	3			
perl	3	4	6	9
pink	2	3	3	
pisa	2	3	3	
pokemon	2	3	5	
retrovirus	2	3	3	5
sailor moon	3	5	5	5
samba	3	5	6	7
search	2	4	6	
seti	2	3	5	
shakira	3	4	6	
silvio berlusconi	2	4	4	
simpson	2	4	4	7
skateboard	3	5	6	9
sony	3	5	7	
spiderman	2	3	4	7
tattos	3	5	7	9
terrorism	3	5	7	10
time	3	5	6	8
travel	3	5	7	
vasco rossi	3	4	4	6
waterloo	3	5	6	7
winmx	2	2		
wtc	3	3		
wwf	3	4	6	7
	P@3	P@5	P@7	P@10
	91%	83%	78%	79%

Figure 19: M@N and P@N for SnakeT 's dataset