Graph Visualization Techniques for Web Clustering Engines

39 Paper Citations **785** Full Text Views

4 Author(s)

Emilio Di Giacomo ; Walter Didimo ; Luca Grilli ; Giuseppe Liotta

View All Authors

Abstract:
One of the most challenging issues in mining information from the World Wide Web is the design of systems that present the data to the end user by clustering them into meaningful semantic categories. We show that the analysis of the results of a clustering engine can significantly take advantage of enhanced graph drawing and visualization techniques. We propose a graph-based user interface for Web clustering engines that makes it possible for the user to explore and visualize the different semantic categories and their relationships at the desired level of detail

Published in: IEEE Transactions on Visualization and Computer Graphics (Volume: 13, Issue: 2, March-April 2007)

Date of Publication: 22 January 2007

INSPEC Accession Number: 9288987

Print ISSN: 1077-2626

DOI: 10.1109/TVCG.2007.40

PubMed ID: 17218746

Publisher: IEEE

Sponsored by: IEEE Computer Society

SECTION 1
Introduction

There is growing consensus that, as the amount of information available on the Web dramatically increases, traditional Web search engines start to show considerable limitations (see, e.g., [3], [4], [5]). In particular, the need of tools capable of organizing Web pages in a more effective way than a long list of URLs has motivated the flourishing of a new generation of search engines, known as *Web clustering engines* (see, e.g., Vivísimo, [1]Boogie, [2] and SnakeT [3] [6]). These systems typically receive a query from the user, forward this query to one or more traditional search engines, and organize the retrieved results into a set of clusters, also called *categories*. Each category contains Web pages that are semantically related to each other and it is labeled with a string that describes its content. The categories are usually organized in a hierarchy, which is displayed to the user. By analyzing the returned hierarchy, the user has a global view of the different semantic areas invoked by her query, and can explore those areas in which she is mainly interested. Fig. 1 illustrates the schema of a Web clustering engine.

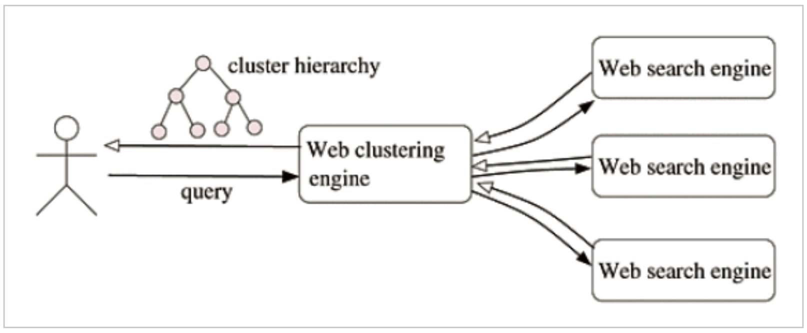


Fig. 1. Schema of a Web clustering engine.

This paper is devoted to the design of effective graphical user interfaces for Web clustering engines. We show that these systems can significantly benefit of advanced graph drawing techniques, which make it possible to overcome some of the limitations of the existing GUIs. We remark that the graphical user interface plays a fundamental role for Web clustering engines since a visually appealing presentation of the categories and of their semantic relationships is essential for efficiently retrieving the wanted information.

In order to better clarify our contribution, we shall first briefly digress on some limitations of the typical visualization paradigm of most Web clustering engines and then overview the results in this paper.

1.1 GUI of Web Clustering Engines

The vast majority of Web clustering engines have a GUI in which the set of clusters is represented as a tree of directories and subdirectories. This type of visual interaction presents several limitations:

- In a tree of categories, nodes that are not in an ancestor/descendant relation might also have strong semantic connections which can be underestimated (if not completely lost) by following the tree structure. This problem is particularly relevant when the labels of some categories are not informative enough for the user. Suppose, for example, that the user’s query is “Armstrong” and that a portion of the tree of clusters is as the one depicted in Fig. 2a. Is the category “Biography” related to “Louis” or to “Lance” or to both (or to neither of them, but to the astronaut Neil Armstrong?). If there were one edge as in Fig. 2b or if there were two edges with different weights as in Fig. 2c, the user could decide whether or not the category “Biography” is of her interest.

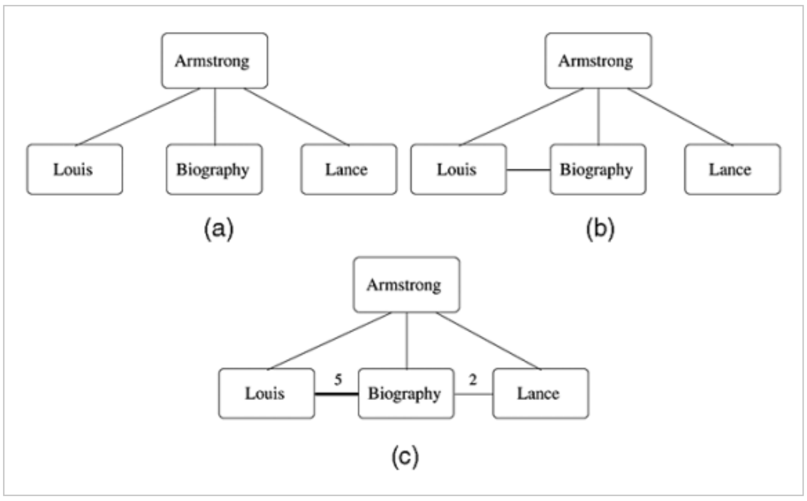


Fig. 2. (a) A portion of a tree of categories for the query “Armstrong.” (b) and (c) The same tree, plus edges that highlight cluster relationships. In (c), each edge is labeled with a number that represents the strength of the edge.

- Showing the categories as the vertices of a graph can also help the user to visually detect communities of categories which identify a common and more general topic and/or to highlight isolated categories which may be pruned if their labels are not meaningful for the user’s query. This type of analysis is clearly infeasible if all categories are displayed in a tree structure.

- Finally, the visualization paradigm adopted by systems like Vivísimo often uses basic drawing techniques: The tree is displayed with poor aspect ratio on the left-hand side of the interaction window. This makes the visual exploration particularly uncomfortable when the degree of the tree is high; expanding a few nodes at the same level can make the rest of the tree no longer visible on the screen.

1.2 Our Results

Our target is to provide the user with a diagram that is structurally more complex and informative than a tree. The user should be allowed to interact with a "clustered graph;" each cluster represents a semantic category that groups a coherent set of documents (vertices of the graph) and may have subclusters; each edge of the graph represents a semantic connection between two clusters. The user should be allowed to expand or to contract each cluster, so deciding the preferred level of detail in the visualization of the information. The edges of the graph should help the user to relate less expressive categories to more expressive ones. The interface should also support the analysis of the user with several visual functionalities, including automatic ranking of categories, pruning or refinement of categories and their connections, and preservation of the user mental map during the browsing.

In order to devise a graph-based graphical user interface having the above characteristics, we present in this paper a new methodology for the design of Web clustering engines, which we call the *topology-driven approach*. As a proof of concept of our methodology, we describe a system, named WhatsOnWeb, that implements the topology-driven approach. We use WhatsOnWeb to compare the effectiveness of a graph-based interface versus a tree-based one and experimentally prove that the graph-based interface makes it easier to find clusters that are related to a given topic even if their labels are not expressive enough.

To the best of our knowledge, the only other Web clustering engine that supports the user with a graph-based interface is Kartoo. [4] However, the graph visualization offered by Kartoo is, in our opinion, not expressive enough. The nodes in the graph of Kartoo are the individual Web pages and the categories are represented as regions of a virtual map. If the user moves the mouse over a page in the map, the system highlights all categories in which the page is contained; by selecting a category, the system highlights all pages that form the specified category. The main drawbacks of Kartoo are: 1) The categories are not organized in a hierarchy. 2) Every map contains only a small number of pages (at most 30) and the user can easily lose the global context of the query. 3) Relationships between categories are not explicitly shown.

1.3 Structure of the Paper

The remainder of this paper is structured as follows: In Section 2, we formally introduce the topology-driven approach. In Section 3, we describe how to construct a graph representing the semantic relationships between documents and, in Section 4, we describe how to compute a clustering of such a graph. Section 5 illustrates a visualization paradigm for the clustered graph and explains how it can be implemented using advanced graph drawing algorithms. The system WhatsOnWeb and its visual analysis tools are described in Section 6. Experiments are provided in Section 7. Future research is discussed in Section 8.

SECTION 2
The Topology-Driven Approach

Our target is to design a graph-based interface for a Web clustering engine, where the user can interact with a clustered graph; each cluster represents a semantic category of coherent set of documents and each edge of the graph represents a relationship between two clusters. This target gives rise to two main issues: 1) How can one compute the clustered graph? 2) Which visualization paradigm can be adopted to represent it in an effective fashion?

We remark that all Web clustering engines we are aware of use clustering strategies that are not suitable for computing a graph of clusters. They adopt agglomerative approaches that construct a tree (and not a graph) of categories from bottom to top (see, e.g., [6], [5], [7]); the semantic relationships between pages in different clusters are not computed.

A natural solution is therefore based on constructing a *base graph* that summarizes the semantic relationships among the pages and on using a divisive clustering algorithm that computes a hierarchy of clusters (for example, based on the topological connectivity of the base graph). The definition of the base graph is however not immediately clear. For example, one could think about it as a subgraph of the Web graph, where edges represent hyperlinks among Web pages. Unfortunately, this approach has two main drawbacks: 1) Recognizing hyperlinks requires downloading and parsing the entire Web pages, which is computationally not reasonable for online interactions. 2) The hyperlinks among the documents returned by a Web search engine in response to a user's query usually give rise to a very sparse graph that is not representative enough of the actual semantic relations among these documents.

We define in this paper a new methodology, called *topology-driven approach*, for the design of a Web clustering engine. It works in three distinct phases:

1. In the first phase, a base graph is computed that summarizes the semantic relationships of the snippets [5] returned by the classical search engines; we call it the *snippet graph*.

2. A clustering algorithm is applied on the snippet graph in order to compute a hierarchy of semantic categories, each category grouping a certain subset of snippets. Relationships among categories that are not in an ancestor/descendant relationship are automatically induced by the edges of the snippet graph; the resulting clustered graph is called the *graph of categories*.

3. The graph of categories is visualized to the user along with a set of functionalities to browse and analyze it.

Each of the three steps can be tackled with different strategies and specific optimization goals. In the next sections, we describe one possible solution for each step of the topology-driven approach.

SECTION 3
The Snippet Graph

Let U be a set of URLs returned by a Web search engine in response to a user's query. The *snippet graph* of U is a weighted graph whose vertices represent the elements of U and whose edges represent the relationships among the snippets of the elements of U. The weight of each edge measures the strength of the relationship between the end-vertices of the edge. Intuitively, the snippet graph summarizes a self-contained set of semantic connections between the elements of U. In the following, if $u \in U$, we call *text of* u the concatenation of the title and the snippet associated with u. In order to construct the snippet graph of U, we perform a sequence of preliminary steps:

- **Cleaning Step**. Stop-words and HTML symbols are removed in the text of each element of U.

- **Stemming Step**. A stemming algorithm is applied on the cleaned texts and with each computed stem s, we associate a word z_s chosen among those whose stem is equal to s; words z_s will be used later to label the clusters. After this step, the text of each URL u is viewed as an ordered sequence of stems, denoted by S_u.

- **Scoring Step**. Each stem s is assigned with a *score* w_s, which measures the "relevance" of s in all URL texts containing s. Several criteria can be used to determine w_s. We adopt a standard function called *tf-idf* (*term frequency-inverse document frequency*) [8]. Namely, let D be the set of URL texts and let $D_s \subseteq D$ be the subset of texts that contain s. The *tf-idf* of s with respect to a text $d \in D$ is defined as: $t(s, d) = f(s, d) \log(|D|/|D_s|)$, where $f(s, d)$ is the frequency of s in d. The score w_s is the average of the *tf-idf* of s over all texts of D_s, i.e.,

$$w_s = \frac{\sum_{d \in D_s} t(s, d)}{|D_s|}.$$

View Source

The snippet graph G of U is defined as follows:

- G has a vertex v_u associated with each element $u \in U$. The label of v_u can be either the title of u or its description as URL.

- G has an edge $e = (v_{u_1}, v_{u_2})$ (where $u_1, u_2 \in U$) if the texts of u_1 and u_2 share some stems, i.e., if $S_{u_1} \cap S_{u_2} \neq \emptyset$. To determine the weight and the label of e, the following procedure is applied. Compute the set $\mathcal{S}_e$ of all maximal subsequences of consecutive stems (each subsequence with single multiplicity) shared by S_{u_1} and S_{u_2}; the elements of $\mathcal{S}_e$ are called *sentences*. The *score of a sentence* $\sigma \in \mathcal{S}_e$ is defined as $w_\sigma = \sum_{s \in \sigma} w_s$, and the weight of e is $w_e = \sum_{\sigma \in \mathcal{S}_e} w_\sigma$. The label of e is set equal to $\mathcal{S}_e$.

According to the above definition, the weight of an edge (v_{u_1}, v_{u_2}) of the snippet graph depends both on the number of sentences shared by texts of u_1, u_2 and on the relevance (score) of these sentences. We remark that, as also pointed out by other authors (see, e.g., [6], [9]), considering sentences shared by two texts is, in general, more informative than considering unordered sets of terms and it allows better estimation of the strength of the connections between the two texts. Sentences can be also used to compute more effective labels for clusters.

The time complexity of computing the snippet graph for a set of n snippets is $\Theta(n^2)$ because each snippet has a size bounded by a constant number. Indeed, a snippet graph may have up to $n(n-1)/2$ edges, and there are cases in which the snippet graph has exactly $n(n-1)/2$ edges (for example when every pair of snippets share a word).

SECTION 4
Computing the Graph of Categories

The *graph of categories* is the clustered graph obtained by the snippet graph together with a hierarchy of clusters of its vertices.

In order to determine clusters in the snippet graph, we search *communities of vertices*, i.e., sets of vertices that are strongly connected from a topological point of view. Among the wide range of graph clustering approaches proposed in the literature, we direct our attention to those adopting a recursive divisive strategy based on graph connectivity (see, e.g., [10], [11], [12]). According to this strategy, the cluster hierarchy is determined by recursively cutting out some edges that disconnect the graph. We exploit a recent algorithm proposed by Brinkmeier [10], based on a new definition of communities of vertices. The definition is as follows: Let $G=(V,E)$ be a weighted graph and let $V'\subseteq V$ be a subset of vertices of G. The *community of* V' *in* G is the largest subgraph of G of maximum edge-connectivity among all subgraphs containing V'. We recall here that the edge-connectivity of a weighted graph G is the weight $W(E')$ of a subset of edges $E'\subset E$ such that the removal of E' disconnects the graph and $W(E')$ is minimum over all subsets E'. According to Brinkmeier's definition, the community of every subset of vertices of G is uniquely defined (this is not true for other definitions based on graph connectivity). The hierarchy of communities is represented as a tree T, where each leaf is a vertex of G and each internal node represents a community in G. We call T the *community tree* of G. The pair (G,T) describes a clustered graph, according to the definition given by Feng et al. [13].

Fig. 3 shows an example of a clustered graph (G,T). Each node μ of T defines a different community (cluster) in G; the vertices of G in the cluster defined by μ are the leaves of the subtree of T rooted at μ. The community of a subset V' of vertices of G is the lowest common ancestor of the leaves of T corresponding to the vertices in V'. For example, the community of vertices 1, 4 is node C.

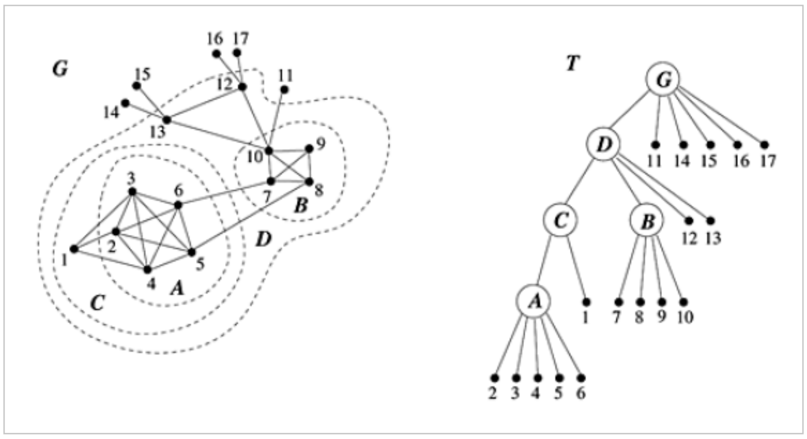


Fig. 3. A clustered graph (G,T), where T is the community tree of G.

Let G be the snippet graph of a set U of URLs. We use the decomposition strategy in [10] as a tool to construct from G the graph of categories. We denote by H_G the hierarchy of categories; the internal nodes of H_G represent the different semantic categories and the root represents the whole set of URLs. The nodes of level $i>0$ (levels of H_G are counted top-to-bottom with the root at level 0) are also referred to as *level-i categories*. In particular, the level-1 categories partition the set of all URLs and represent the macro categories initially shown to the user. Each of these categories can be itself partitioned into disjoint subcategories, that are level-2 categories, etc. The leaves of H_G represent URLs. Notice that H_G does not allow repetition of URLs and, hence, each URL is represented by exactly one leaf of H_G. This implies that, for any pair of categories A and B, either A and B are disjoint or one of them includes the other. In fact, in order to simplify the information returned to the user, we aim at assigning each URL to its most representative category and possible relationships between disjoint categories will be highlighted using a new visualization paradigm (see Section 5).

We now describe in detail how to compute H_G. The set of level-i categories of H_G is denoted by L_i.

- **Computing and labeling level-1 categories**. The level-1 categories are computed by repeatedly applying the following three steps: 1) Compute the community tree T of G. 2) Insert in L_1 the nodes of T having only leaves as children (for example nodes A and B in Fig. 3). 3) Remove from G the vertices corresponding to the children of the nodes inserted in L_1 during the previous step (for example, vertices 2, 3, 4, 5, 6, 7, 8, 9, and 10 in Fig. 3). We experimentally observed that seven to eight iterations are enough to compute the set of the most meaningful categories. The URLs that are not inserted in any category are stored in a new category that we call a *dummy category*.

To label a level-1 category μ of H_G, we apply the following procedure. Let $G_\mu = (V_\mu, E_\mu)$ be the subgraph of G induced by the vertices of G contained in μ, and let $\mathcal{S}_\mu$ be the set of all labels of the edges of G_μ, that is $\mathcal{S}_\mu = \cup_{e \in E_\mu} \mathcal{S}_e$. If a sentence $\sigma \in \mathcal{S}_\mu$ occurs in the label of k edges $(k > 0)$ of G_μ, then kw_σ is called the *total score of* σ *in* G_μ. Denoted by $\sigma = s_1, s_2, \ldots, s_h (h > 0)$ any sentence of S_μ with maximum total score, the label of μ is defined as the concatenation of the (spaced) words $z_{s_1}, z_{s_2}, \ldots, z_{s_h}$. The dummy category is labeled with a dummy label, for example, the label Other Topics. Notice that the labeling procedure may give rise to pairs of categories $\{\mu_1, \mu_2\}$ such that all words of the label of μ_2 belong to the label of μ_1. In this case, we merge μ_1 and μ_2 into a new category μ and assign to μ the label of μ_1.

- **Computing and labeling level-$i$categories**. The level-i categories $(i > 1)$ are computed similarly to the level-1 categories, using a recursive approach that progressively simplifies the snippet graph. Suppose that all level-i categories have been computed, for some $i \geq 1$. We compute the level-$(i+1)$ categories as follows:

  - For each level-i category μ of H_G, let G_μ be the subgraph of G induced by the vertices in μ. We compute a graph $G_\mu{}^-$ from G_μ by simplifying the labels and the weights of the edges. Namely, let e be an edge of G_μ, let $\sigma_1, \ldots, \sigma_k (k \geq 1)$ be the sentences in the label of e, and let σ be the sequence of the stems of the words in the label of μ. If all the words of $\sigma_i (i = 1, \ldots, k)$ are contained in σ then σ_i is removed from the label of e, and the weight of e is decreased accordingly.

  - Compute L_{i+1} by applying on every $G_\mu{}^-$ the same procedure as the one applied on G for computing L_1. The nodes in L_{i+1} obtained from $G_\mu{}^-$ are children of μ in H_G.

  Given a level-i category μ of H_G, the recursive algorithm stops to further decompose μ if one of the two following cases holds: 1) the community tree of G_μ only consists of one node, representing G_μ itself and 2) i has reached a desired value.

  The labeling procedure of level-i categories is the same as the one used for the level-1 categories.

- **Computing the leaves**. Let $\mathcal{L}$ be the set of categories that have no child in H_G (i.e., the deepest categories). For each $\mu \in \mathcal{L}$ and for every vertex of G in μ, we create a new corresponding child-node of μ, which will be a leaf of H_G. Each leaf of H_G has the label of its associated vertex of G.

The theoretical time complexity of constructing H_G mainly depends on the time required for computing a community tree. Indeed, both the number of levels of H_G and the number of iterations applied to determine the nodes of each level can be set to constant numbers. To compute the community tree, Brinkmeier [10] describes an $O(n^2 m + n^3 \log n)$ technique based on recursively applying a minimum cut algorithm (n and m are the number of vertices and the number of edges of the input graph, respectively). Although this time complexity is rather high from a theoretical perspective, in practice, the computation of a minimum cut can be speeded-up in many cases, using some simple considerations on the degree of the vertices (see also [10]).

SECTION 5
The Visualization Paradigm

There are many possible ways of visualizing and browsing the graph of categories. They depend both on the amount of information (vertices and links) shown to the user at each exploration step and on the graph drawing convention adopted to represent such a relational information.

As pointed out by many authors (see, for example, [14], [15], [16]), the following goals should be taken into account in the design of a graph visualization paradigm:

- The information shown to the user should not be overly complex. This means that not all vertices and edges of the graph of categories are necessarily displayed at the same time; instead, the user should be allowed to visualize only those clusters in which she is interested, by means of a dynamic exploration. Also, since the snippet graph usually has a high number of edges, which may generate many crossings in the drawing, some mechanisms should be applied to automatically simplify the relationships among clusters, so emphasizing only those considered as the most relevant.

- The drawing algorithm adopted to visualize the graph of categories should compute aesthetically pleasing drawings and should guarantee the preservation of the user's mental map during the browsing.

We propose here a visualization paradigm that takes into account the two goals described above. Let G be the snippet graph of a set of URLs and let H_G be a tree of categories. For any internal node μ of H_G, we denote by G_μ the subgraph of G induced by the leaves of the subtree rooted at μ. Our drawing strategy works in three steps:

- **Removal of redundant edges**. We apply on G a preprocessing step in order to remove those edges that do not provide additional useful information to the structure of the clusters. Namely, for each internal node μ of H_G, we remove from G_μ every edge whose label only consists of sentences that, in their unstemmed version, are completely contained in the label of μ. Denote by G' the graph obtained from G applying this cleaning operation over all subgraphs G_μ. We maintain on G' the same cluster hierarchy as G, i.e., $H_{G'} = H_G$.

- **Merge of cluster relationships**. We transform graph G' into a new *super clustered graph* G'' such that each cluster μ is explicitly represented as a kind of *super node*, which we denote by C_μ. Super nodes can be connected by *super edges* to emphasize the relationships among clusters. Super edges can be seen as extra arcs connecting nodes of $H_{G'}$. More formally (see also Figs. 4a and 4b), let μ_1 and μ_2 be any two internal nodes of $H_{G'}$ that are children of the same node μ, and let $E(\mu_1, \mu_2)$ be the subset of edges of G' that connect vertices of G'_{μ_1} to vertices of G'_{μ_2}. If $E(\mu_1, \mu_2)$ is not empty, we replace this set of edges by a new super edge e connecting the super nodes C_{μ_1} and C_{μ_2}. The label of e is the union of all labels of the edges in $E(\mu_1, \mu_2)$, and the weight of e is the sum of all weights of the edges in $E(\mu_1, \mu_2)$.

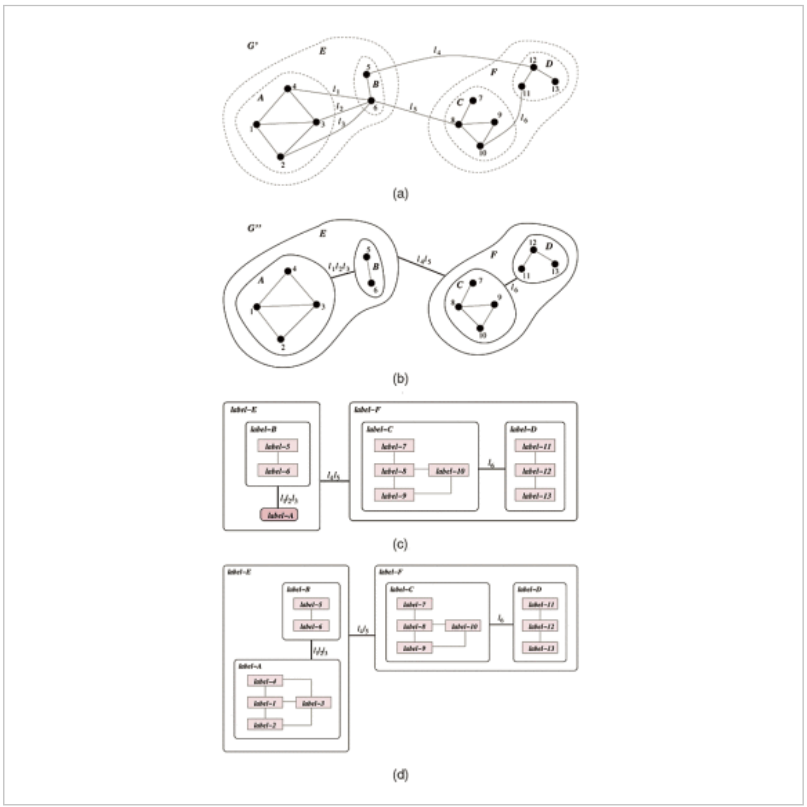


Fig. 4. (a) A clustered graph G' with no redundant edges; the edges connecting vertices in distinct clusters are labeled. (b) The graph G'' obtained from G' by representing clusters with super nodes. (c) A drawing of G''; the string *label*-x denotes the label of object (cluster or vertex) x; cluster A is contracted. (d) A drawing of G'', where all clusters are expanded.

- **Visualization of maps**. Let μ be any cluster of G''. The graph consisting of the children of μ and their relationships is represented as an *orthogonal drawing* with box-vertices; each vertex is drawn as a box with the size required to host its label or its subclusters, depending on the fact that it is contracted or expanded. Each edge is drawn as a chain of horizontal and vertical segments between its extremal vertices. Figs. 4c and 4d show two different maps of the clustered graph G'' in Fig. 4b. During the browsing of the user, the map displayed on the screen changes dynamically, depending on the expansion or contraction operations performed by the user. The preservation of the user's mental map is achieved by computing an "orthogonal shape" for each subgraph of the super clustered graph G'' and by always using the same shape each time a drawing of that subgraph is displayed. An orthogonal shape describes the angles at vertices formed by two incident edges and the left and right turns on each edge. Our drawing algorithm works within a popular graph drawing approach, called *topology-shape-metrics* [17], which experimentally guarantees highly readable drawings in terms of number of edge crossings, number of edge bends and drawing area [18]. In order to deal with box-vertices of different sizes and any degree, we adopt the variant described in [19], which computes highly compacted drawings. During the browsing of the user, each cluster μ maintains a state that tells whether it is expanded or contracted. If μ is contracted, it is represented as a “small” rectangle containing its label; all its subclusters are hidden. If μ is expanded, its super node C_μ is drawn as a rectangular region r_μ that contains the drawings of its subclusters and the label of μ. The dimensions of r_μ depend on the state of the subclusters of μ, which can be expanded or not. To determine the dimensions of r_μ, we apply a procedure that recursively constructs a drawing Γ of the subclusters of μ and then sets the height and the width of r_μ as the height and the width of the bounding box of Γ, plus a small area needed to place the label of μ.

SECTION 6
The System WhatsOnWeb

To prove the feasibility of the topology-driven approach and its effectiveness we developed a system prototype called WhatsOnWeb, which implements the clustering algorithm described in Section 4 and the visualization paradigm described in Section 5. The system, which is available online at the URL http://whatsonweb.diei.unipg.it, uses a Java applet from the client side and a Java servlet from the server side. The servlet manages the retrieval of data from classical search engines and interacts with a graph drawing engine based on the GDToolkit library. [6] In the current implementation, WhatsOnWeb only retrieves data from the Google search engine.

The graphical environment for the user consists of two frames (see, e.g., Fig. 5). In the left frame, the cluster hierarchy is presented to the user as a classical tree of directories. In the right frame, the user interacts with a map that gives a graphical view of the graph of categories at the desired level of detail. Initially, the map shows only the level-1 categories and their relationships, according to the principles given in the previous section. The user can expand or contract any category, by simply clicking on the corresponding box. Fig. 5 shows a snapshot of the interface, where the results for the query “Armstrong” are presented; in the figure, the category “Louis Armstrong” has been expanded by the user. During the browsing, the system automatically keeps consistent the map and the tree, and preserves the user’s mental map by preserving the shape of the drawing. For example, Fig. 6 shows the map obtained by expanding the categories “Jazz,” “School,” and “Louis Armstrong Stamp” in the map of Fig. 5.

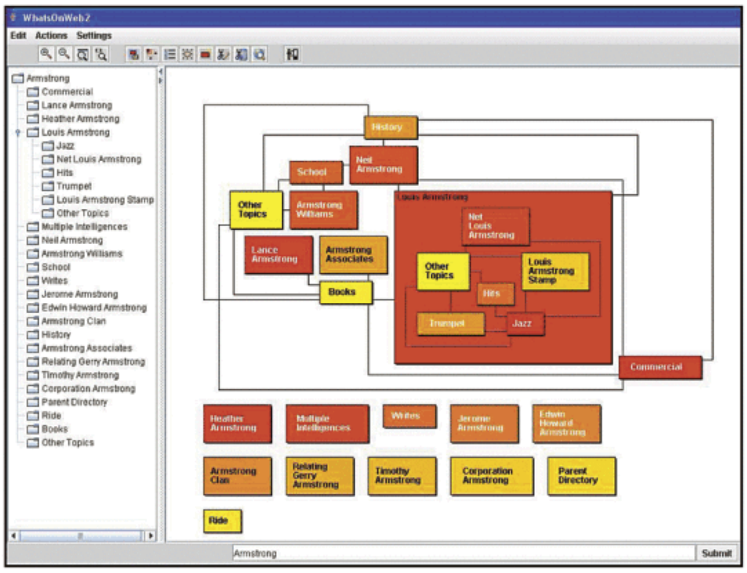


Fig. 5. Snapshots of the user interface: A map for the query “Armstrong;” in the map, the user performed the expansion of the category “Louis Armstrong.”

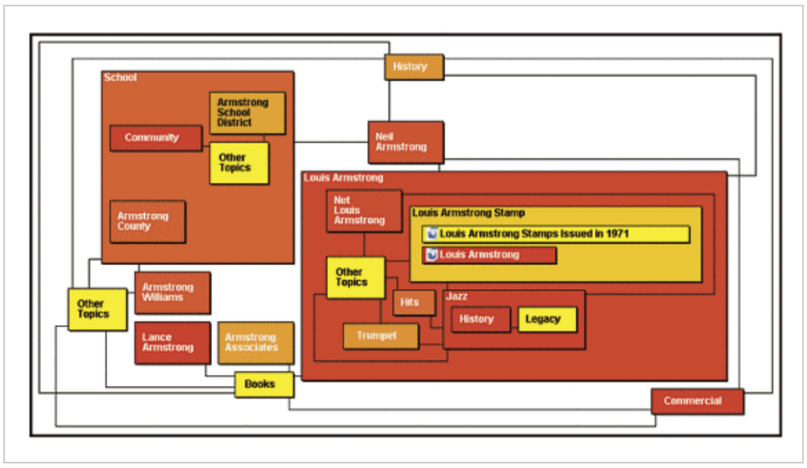


Fig. 6. Snapshots of the user interface: A map obtained from that of Fig. 5 by expanding the categories “Jazz,” “School,” and “Louis Armstrong Stamp” ; this last category contains two URLs, represented by using their title.

Besides the browsing functionalities described above, the interface is equipped with several visual analysis tools that increase the interaction between the human and the system. They are listed and described below:

- **Ranking of Categories**. WhatsOnWeb ranks the semantic categories at each level, by assigning to each category a *relevance score*. This score is proportional to these three main parameters: 1) the size of the category, 2) the maximum rank (returned by Google) of a document in the category, and 3) the deviation of the label score of the category from the average score of all sentences in the same category. We use this third parameter as an estimate of the reliability of the sentence used to label the category. In the tree representation, the categories are ordered from top to bottom according to their decreasing relevance score. In the map, the relevance score is conveyed to the user adopting different colors in the red scale (the scale of colors can be customized); highly red categories are the most relevant ones, while yellow categories are the least relevant. For example, in Fig. 5, the expanded category “Louis Armstrong” is supposed to be one of the most relevant categories at the first level. Conversely, the category “Ride” has a low relevance score for the system, and indeed its label does not appear meaningful; this category is even completely unrelated to any other category.

- **Automatic Filtering**. The user can customize the visibility threshold both for the categories and for their relationships. The system automatically hides those edges whose weight is less than the given threshold and those categories that are isolated and whose score is less than the specified threshold.

- **Pruning of Categories**. The user can select a desired subset of categories in which she is no longer interested. After this selection, the system can be forced to only recompute the drawing, without changing the content of the remaining clusters or it can be forced to also recompute the whole cluster hierarchy using the remaining documents. This pruning operation can be used to progressively reduce the amount of information that the user handles, and to refine the remaining information if necessary.

- **Edge Exploration**. The user can explore the information associated with the edges of the map. Namely, if the mouse is positioned on an edge (u, v), a tool-tip is displayed that shows both the weight and the complete list of strings that form the labels of the relationship between u and v. Clicking on the edge, a drawing is shown to the user to display the URLs in u, the URLs in v, and the relationships between these two sets of URLs, as they appear in the snippet graph. This feature makes it possible to isolate all those documents that give rise to a specific relationship and can be also used as a tool to evaluate and tune the mechanism adopted by the system for creating cluster relationships. Since the graph to be shown when the user explore an edge is bipartite, the system represents it as a 2-layered drawing, i.e., a drawing where the vertices of the two sets of the partition lie on two parallel straight lines (vertical in our case). For example, Fig. 7 depicts the 2-layered drawing that shows the relationships between the URLs of “Louis Armstrong Stamp” and those of “Jazz.”

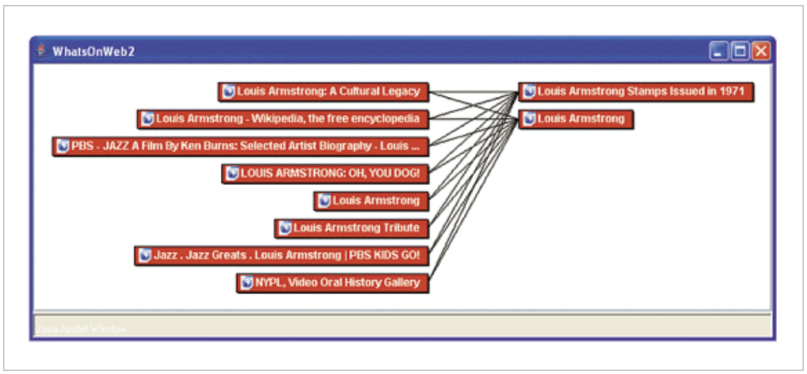

WhatsOnWeb2
Louis Armstrong: A Cultural Legacy
Louis Armstrong - Wikipedia, the free encyclopedia
PBS - JAZZ A Film By Ken Burns: Selected Artist Biography - Louis ...
LOUIS ARMSTRONG: OH, YOU DOG!
Louis Armstrong
Louis Armstrong Tribute
Jazz . Jazz Greats . Louis Armstrong | PBS KIDS GO!
NYPL, Video Oral History Gallery
Louis Armstrong Stamps Issued in 1971
Louis Armstrong
Java Applet Window

Fig. 7. A *2*-layered drawing that shows the relationships between the URLs of “Louis Armstrong Stamp” and those of “Jazz.”

- **Query Refinement**. The user can select one or more categories and can ask the system to repeat the query by adding to the original query string the labels of the selected categories. In this way, it is possible to obtain semantic categories that are more and more specific for the domain in which the user is interested.

SECTION 7
Experimental Analysis

Our main goal is measuring how much an end-user can benefit of an exploration of a hierarchy of semantic categories using the graph-based visualization paradigm of WhatsOnWeb (Section 7.2). As a preliminary step, we identified a set of queries for the experiment and studied the properties of the corresponding clustering of WhatsOnWeb (Section 7.1). We also measured the running time taken by WhatsOnWeb for computing and visualizing the clustered graphs relative to the chosen queries (Section 7.3).

7.1 Queries and Properties of the Clusters

The experiment used a set of nine queries, classified as follows: 1) *Ambiguous queries*: Armstrong, Jaguar, Matrix; 2) *Generic queries*: Music, Flower, Travel; and 3) *Specific queries*: Olympic Games, Iraq, World War 2. We recall that an analogous classification of queries has been already described in the literature [20]. Although such a classification highly relies on intuition and it is not easy to formalize from an operational point of view (in [20], for example, it is used without any other explanation), we used the following rules to classify each query q in our set.

Query q was classified as *ambiguous* if there were at least two completely unrelated topics associated with q according to the common sense, and there were at least 10 snippets belonging to each of these topics among those returned by the system Vivísimo, which is considered one of the most effective Web clustering engines [7] (see also [6]). For example, concerning the query Jaguar it was possible to find three unrelated topics: The first one about Jaguar cars containing 20 snippets, the second one about software (operating system, software, etc.) containing 12 snippets, and the third one about the animal containing 14 snippets. Query q was classified as *specific* if it was the name of an entity (e.g., person, place, event) and it was not ambiguous. Finally, q was classified as *generic* if it was a generic word of the English dictionary that did not describe a person, a place, or an event and that cannot be classified as ambiguous.

To analyze the quality of the clustering computed by WhatsOnWeb on the above defined queries, we measured the similarity between its 1-level clusters and those computed by either a human or the system Vivísimo.

The sets of clusters computed by WhatsOnWeb, by the human, and by Vivísimo are denoted as C_W, C_H, and C_V, respectively. Both C_W, C_H, and C_V were computed on the same set S of snippets, those returned by Vivísimo. The size of S ranged from 180 to 220 depending on the specific query q. In order to consider the clustering C_H as a “ground truth,” we gave to the human the possibility of accessing the entire Web pages associated with the snippets in S and gave her no guidance about the number and the size of the clusters to be computed. To measure the similarity between C_W and C_H and between C_W and C_V, we adopted a standard function known as *F-measure*. See [21] for details for the definition of this function.

Fig. 8 shows the similarity between the clusters of WhatsOnWeb and those of Vivísimo and the human. The data show that the similarity between the clustering of WhatsOnWeb and that of the human tends to increase from specific queries to ambiguous ones. For ambiguous queries, the “quality” of the clusters is rather good; as the next experiment shows, the lower quality of the clusters in the case of specific queries is compensated by the existence of links between categories that help the user to retrieve relevant information. In our opinion, the differences in the quality of the clusters for the different types of queries can be, in part, justified observing that for ambiguous queries there exist well distinct semantic areas and the content of the snippets is often informative enough to easily discriminate whether a page belongs to a semantic area or to another; hence the behavior of WhatsOnWeb is closer to the human perception for this type of queries. At the other extreme, the context of specific queries is much more confined and deciding how to further classify the documents becomes a more artificial task; as a consequence, the set of clusters may vary significantly, depending on the system/human that compute them.

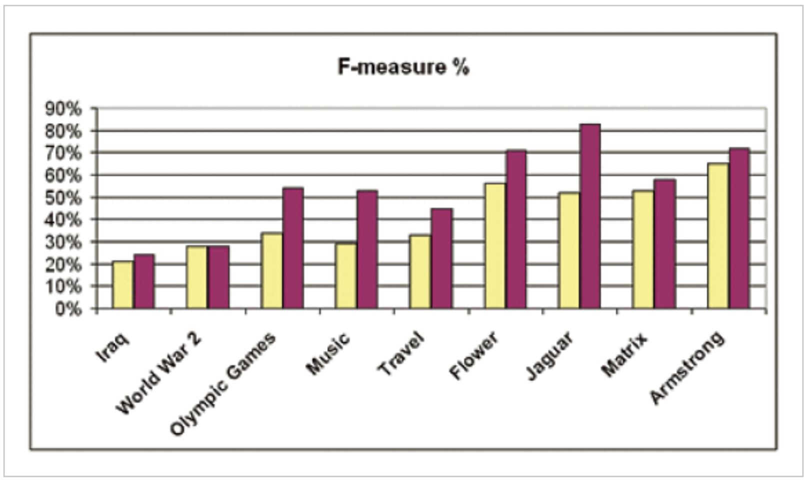


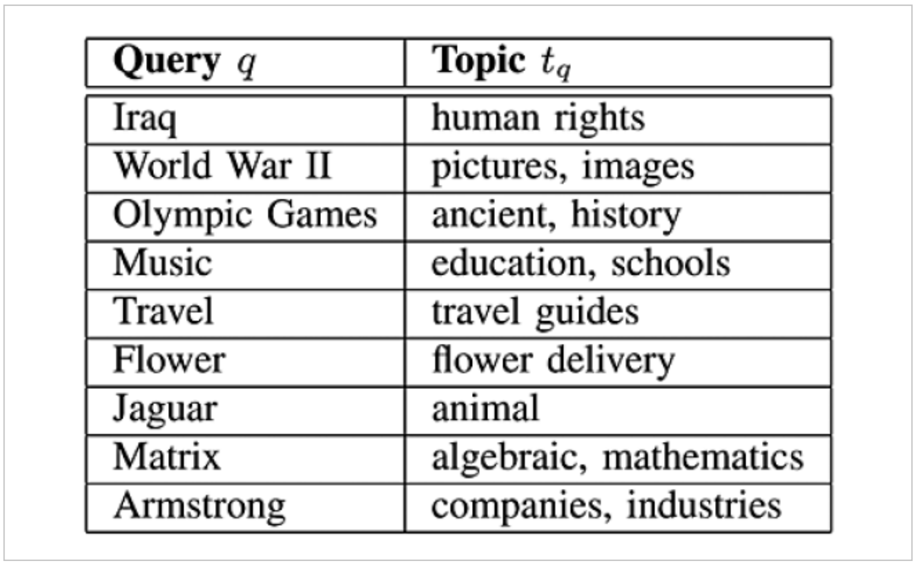
Fig. 8. Similarity between clusters computed by WhatsOnWeb and clusters computed by: 1) a human (light bars) and 2) Vivísimo (dark bars).

A second observation on the data is that the similarity with the clusters computed by Vivísimo is higher (sometimes significantly) than the similarity with the clusters computed by the human.

7.2 Benefits of the Graph-Based Visualization Paradigm

Starting from the observation that Web clustering engines are explicitly designed for those users who wish to find groups of semantically coherent Web pages, we performed an experiment where users were given a query q and a topic t_q related to q, and were asked to find all pages relevant for t_q. For example, for the query Jaguar, the topic was Animal, i.e., the user was asked to select all pages about the animal, discriminating them from those about the car or the operating systems. For each query, the corresponding topic is shown in Table 1.

TABLE 1 Queries and Topics for the Experimental Analysis



Query q	**Topic** t_q
Iraq	human rights
World War II	pictures, images
Olympic Games	ancient, history
Music	education, schools
Travel	travel guides
Flower	flower delivery
Jaguar	animal
Matrix	algebraic, mathematics
Armstrong	companies, industries

We wanted to experimentally confirm our intuition that the tree-based interface can effectively support the user mainly when the clusters have meaningful labels for t_q; however, after these clusters have been explored the user basically follows an exhaustive sequential visit of the remaining clusters with less expressive labels, in search of pages that could be relevant for t_q. Conversely, the graph-based interface makes it easier for the user to identify those clusters that are related to the topic t_q even if their labels are not expressive enough, since these clusters are typically connected by edges to other clusters with more informative labels.

In order to run the experiment, we hired 10 volunteers (either students or staff members in our same engineering school) who are between 20 and 40 and are well acquainted with using classical Web search engines. The setup for our experiments consisted of two distinct interfaces, both resulting from a customization of WhatsOnWeb: one interface showed only a full-screen view of the tree of clusters (the left side of the interface in Fig. 5) and the other showed only a full-screen view of the graph of clusters (the right side of the interface in Fig. 5). Each interface made it possible to store a page in an “experimental basket” by clicking on an icon near to the label of the page. In order to select a page the user could: 1) Read the label of the page, 2) read the associated snippet in a tool-tip, and 3) open the entire page in the Web browser.

For each pair (q, t_q) the set of pages, clusters, and labels was the same in the two interfaces and also the initial set of clusters showed to the user was identical in both interfaces. The task of the user was to find and select in a given amount of time (7 minutes) those pages that in her opinion were pertinent to t_q. We crossed users and queries such that: 1) Each user never experimented the same pair (q, t_q) on both the tree and the graph interfaces and 2) each pair (q, t_q) was experimented on each interface by half of all users.

We traced in a log file all cluster expansions performed by the user and collected all pages inserted in the experimental basket. We measured two parameters: The total number of correct pages found in the given time, and the ratio between the number of correct pages found and the number of clusters expanded in each interface; we call this second parameter *hit ratio*: The higher the hit ratio the smaller is the number of clusters unsuccessfully browsed in search of a meaningful page. Measuring the hit ratio is particularly relevant in our opinion, because a high number of unfruitful cluster expansions witness some sort of sequential search, comparable with just scrolling the list of URLs returned by a traditional search engine.

Fig. 9 shows the total number of correct pages found for each query with the two interfaces, within the given time. We observe that, for most queries, 7 minutes was a sufficiently long time to successfully attack the search problem by sequentially exploring all clusters in the tree-based interface. Indeed, the number of correct pages found with the two interfaces was comparable in most cases, although there were two queries (Iraq and Travel) for which the improvement of the graph-based interface with respect to the tree-based one was up to 50 percent. The results about the hit ratio, however, showed that the effort in finding the correct pages in terms of number of clusters explored is dramatically lower for the graph-based interface (see Fig. 10). Indeed, the hit ratio of the graph-based interface outperforms the one of the tree-based interface in all cases and it is greater than or equal to 1 in many cases. The average improvement of the graph-based interface with respect to the hit ratio in the tree-based interface is around 104 percent for specific queries, 75 percent for generic queries, and 56.14 percent for ambiguous queries. This trend can be in our opinion justified in part by observing the results of the previous experiment (see Fig. 8), which shows that the similarity between the clustering of Whats OnWeb and the human clustering tends to increase when going from specific to ambiguous queries. Since the way of clustering results for ambiguous queries is closer to the human one, the labels of the clusters are typically more informative for the users and suffice to retrieve the relevant information. On the other hand, the label of a cluster for a specific query often does not allow the user to clearly guess its content and the additional links between clusters in the graph-based interface represent a valuable support to cope with this problem.

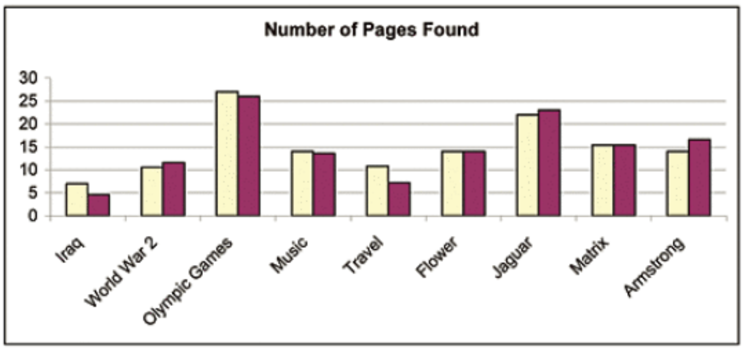


Fig. 9. Average number of correct pages found (y-axis) in the time allotted for each query (7 minutes), with the graph-based interface (light bars) and with the tree-based interface (dark bars).

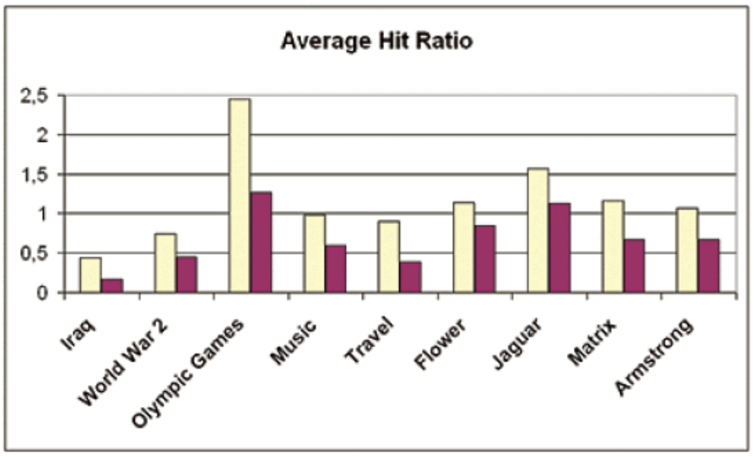
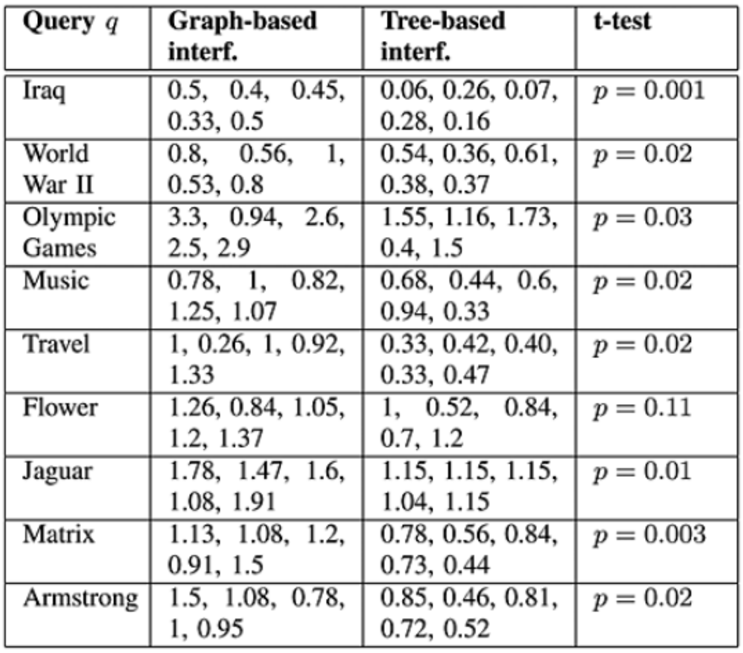


Fig. 10. Average hit ratio (y -axis) for each query with the graph-based interface (light bars) and with the tree-based interface (dark bars).

The reliability of the data from a statistical point of view was proven by performing a two-tailed t-test [8] on the raw data of the hit ratio for each user and for each query. These data together with the p value are reported in Table 2.

TABLE 2 Hit Ratio for Each User and Each Query



Query q	Graph-based interf.	Tree-based interf.	t-test
Iraq	0.5, 0.4, 0.45, 0.33, 0.5	0.06, 0.26, 0.07, 0.28, 0.16	$p = 0.001$
World War II	0.8, 0.56, 1, 0.53, 0.8	0.54, 0.36, 0.61, 0.38, 0.37	$p = 0.02$
Olympic Games	3.3, 0.94, 2.6, 2.5, 2.9	1.55, 1.16, 1.73, 0.4, 1.5	$p = 0.03$
Music	0.78, 1, 0.82, 1.25, 1.07	0.68, 0.44, 0.6, 0.94, 0.33	$p = 0.02$
Travel	1, 0.26, 1, 0.92, 1.33	0.33, 0.42, 0.40, 0.33, 0.47	$p = 0.02$
Flower	1.26, 0.84, 1.05, 1.2, 1.37	1, 0.52, 0.84, 0.7, 1.2	$p = 0.11$
Jaguar	1.78, 1.47, 1.6, 1.08, 1.91	1.15, 1.15, 1.15, 1.04, 1.15	$p = 0.01$
Matrix	1.13, 1.08, 1.2, 0.91, 1.5	0.78, 0.56, 0.84, 0.73, 0.44	$p = 0.003$
Armstrong	1.5, 1.08, 0.78, 1, 0.95	0.85, 0.46, 0.81, 0.72, 0.52	$p = 0.02$

7.3 Running Time

We measured the running time taken by WhatsOnWeb to perform each step of the topology-driven approach on the same set of queries used in the previous experiments. For each query, the system analyzed the first 200 snippets returned by Google. We remark that, with the development of the system we were mainly interested in proving both the feasibility of the topology-driven approach and the advantages that a graph-based user interface offers with respect to a classical tree-based one, while we did not focus on building up a particularly fast implementation. Indeed, from the efficiency point of view, our system leaves room for further improvements.

For most of the queries, the total running time was less than 30 seconds. The most expensive step is the computation of the cluster hierarchy starting from the snippet graph, while the creation of the snippet graph and the visualization step are rather fast in general. We also observed that the running time is higher for specific queries, because these queries give rise to snippet graphs that are quite dense, i.e., that have a high number of edges; in fact, snippets relative to specific queries usually share a considerable number of terms, because their semantic context is quite well confined. For ambiguous queries, the running time was typically less than 15 seconds.

SECTION 8
Future Research

The results in this paper give rise to several new research directions. We just presented one possible way of implementing each step of the topology-driven approach, but different algorithms can be investigated and compared. In particular, we aim at devising more efficient clustering strategies, still based on a divisive technique. Also, for the representation of the graph of categories, we want to study new visualization paradigms that allow us to convey the rank of the categories by placing them on different geometric layers (for example, parallel lines or concentric circles), and experimentally compare them.

ACKNOWLEDGMENTS

This paper describes the results of a two year project. Preliminary versions of part of its contents have been presented in conferences. An extended abstract introducing the topology-driven-approach was presented at SOFSEM 2005 [1]; an extended abstract focused on the graph-based user interface was presented at GD 2005 [2]; the experimental analysis was never submitted to any conference. The research was partially supported by the MIUR Project ALGO-NEXT: Algorithms for the Next Generation Internet and Web: Methodologies, Design and Applications.

Keywords

IEEE Keywords
Graphical user interfaces, Search engines, Data visualization, Web pages, Tree data structures, Web sites, User interfaces, Software systems, Web search, Organizing

INSPEC: Controlled Indexing
search engines, graph theory, graphical user interfaces, information retrieval, Internet

INSPEC: Non-Controlled Indexing
graphical user interfaces, graph visualization, Web clustering engines, information mining, World Wide Web, semantic categories, graph drawing, graph-based user interface, Web search, information visualization

Author Keywords
graphical user interfaces., Web search, clustering, information visualization, visualization systems and software, visualization techniques and methodologies

MeSH Terms
Algorithms, Cluster Analysis, Computer Graphics, Database Management Systems, Databases, Factual, Information Dissemination, Information Storage and Retrieval, Internet, Pattern Recognition, Automated, Semantics, User-Computer Interface

Authors

Emilio Di Giacomo
Dipt. di Ingegneria Elettronica e dell'Informazione, Univ. degli Studi di Perugia

Emilio Di Giacomo received the MS degree in electronic engineering from the University of Rome “La Sapienza” in 2000 and the PhD degree in computer science from the University of Perugia in 2003. He is currently an assistant professor of computer science at the University of Perugia. His research interests include graph drawing, information visualization, algorithm engineering, and computational geometry.

Walter Didimo
Dipt. di Ingegneria Elettronica e dell'Informazione, Univ. degli Studi di Perugia

Walter Didimo received the PhD in computer science from the University of Rome “La Sapienza” in 2000. He is currently an associate professor of computer science at the University of Perugia. His research interests include graph drawing, information visualization, algorithm engineering, and computational geometry. He collected more than 40 international publications in the above areas. He served in organizing and program committees of international conferences, including the International Symposium on Graph Drawing and the ACM Symposium on Computational Geometry. He is involved in several national and international research projects, including projects supported by the Italian Ministry of Education and by the European Community.

Luca Grilli
Dipt. di Ingegneria Elettronica e dell'Informazione, Univ. degli Studi di Perugia

Luca Grilli received the MS degree in electronic engineering from the University of Perugia, Italy, in 2003. He is a PhD candidate in information engineering at the University of Perugia, Italy. His research interests include Web clustering, information visualization, and Web data mining. Currently, he works on Web clustering engines and applications of graph drawing techniques for Web searching.

Giuseppe Liotta
Dipt. di Ingegneria Elettronica e dell'Informazione, Univ. degli Studi di Perugia

Giuseppe Liotta received the PhD degree in computer engineering from the University of Rome “La Sapienza” in 1995. He was a postdoctorial candidate at Brown University from May 1995 to September 1996. From October 1996 to October 1998, he was an assistant professor (ricercatore) in the Department of Computer Engineering of the University of Rome “La Sapienza.” He then joined the Department of Electrical Engineering and Computer Science of the University of Perugia, where he served as associate professor from 1998 to 2002. Since 2002, he has been a professor of computer science and currently serves as the chair of the studies in computer science and electrical engineering of the University of Perugia. His research interests are mainly directed at the design of algorithms for solving problems motivated from graph drawing, computational geometry, and information visualization. On these topics, he has edited special issues, written book chapters and surveys, organized conferences and workshops, and published more than 80 research papers. His research has been funded by MIUR, CNR, NATO, EU, and by several industrial sponsors.

Related Articles

Guest editorial - architectures and protocols for wireless mobile internet
W.W. Lu

A site for sore eyes?
P. Clapham