Automatic Construction of Multifaceted Browsing Interfaces

Wisam Dakka
Columbia University
wisam@cs.columbia.edu

Panagiotis G. Ipeirotis
New York University
panos@stern.nyu.edu

Kenneth R. Wood
Microsoft Research
Cambridge
krw@microsoft.com

ABSTRACT

Databases of text and text-annotated data constitute a significant fraction of the information available in electronic form. Searching and browsing are the typical ways that users locate items of interest in such databases. Interfaces that use multifaceted hierarchies represent a new powerful browsing paradigm which has been proven to be a successful complement to keyword searching. Thus far, multifaceted hierarchies have been created manually or semi-automatically, making it difficult to deploy multifaceted interfaces over a large number of databases. We present automatic and scalable methods for creation of multifaceted interfaces. Our methods are integrated with traditional relational databases and can scale well for large databases. Furthermore, we present methods for selecting the best portions of the generated hierarchies when the screen space is not sufficient for displaying all the hierarchy at once. We apply our technique to a range of large data sets, including annotated images, television programming schedules, and web pages. The results are promising and suggest directions for future research.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Multifaceted Hierarchies, Browsing, Hiearchy Construction, Faceted Classification, Faceted Navigation

1. INTRODUCTION

Databases of text and text-annotated data constitute a significant fraction of the information available in electronic form. Users typically locate items of interest in such databases either by using keyword-based search or by browsing through the contents of the collection. Concept hierarchies are commonly used to support browsing activity. Most libraries use the Dewey Decimal system to organize their holdings, allowing their users to navigate easily through the contents of the library. Yahoo! also uses a topic-based hierarchy to organize web sites according to their topic and allows the users to identify quickly web pages of interest.

Most of the existing systems use a single hierarchy to present the contents of a collection. Early work by Pollitt [28] and more recently by Yee et al. [33] showed that multifaceted hierarchies, which allow users to browse across multiple dimensions, are superior than single, monolithic hierarchies. For example, consider the case of a user looking for images with dogs playing with children in a farm. Having multiple hierarchies, the user can browse first through the hierarchy "Animals" and select the category "Mammals → Carnivores → Dogs." Then, having the "Animal" dimension fixed, the user can browse through the hierarchy "Places" to locate images with farms, and then browse through the hierarchy "People" to locate images with children. Such multifaceted interfaces expose the contents of the underlying collection and can help users more quickly locate items of interest. The major shortcoming of the systems that use multifaceted hierarchies is the need to:

1. Identify *manually* the dimensions/facets that can be used to browse a collection, and
2. Create *manually* the hierarchies for each dimension.

In this paper, we present techniques for *automatically constructing multifaceted hierarchies* from a large collection of text or text-annotated objects. Specifically, the contributions of this paper are:

1. A technique for discovering facets that can be used to browse a collection and identifying the appropriate keywords for each discovered facet (Section 3).
2. An efficient hierarchy construction algorithm that leverages the capabilities of relational database systems and stores the hierarchy in an RDBMS for scalable deployment (Section 4).
3. A set of methods for selecting the best portions of the generated hierarchies when the screen space is not sufficient for displaying all the hierarchy at once (Section 5).
4. An extensive experimental evaluation, including three real-life data sets, demonstrating that the proposed techniques are scalable and generate concept hierarchies with good structural properties.

The rest of the paper is organized as follows: Section 2 gives the necessary background. Section 3 describes our facet

CIKM'05, October 31–November 5, 2005, Bremen, Germany.

extraction technique and Section 4 outlines our subsumption-based hierarchy construction algorithm. Section 5 describes our algorithms for selecting the best categories from a hierarchy. Section 6 describe the experimental setting and results. Finally, Section 7 discusses related work and Section 8 concludes the paper.

2. BACKGROUND

Although we are aware of no work on automatic construction of *multifaceted* browsing schemes, automatic creation of subject hierarchies has been attracting interest for a long time, mainly in the form of *clustering* [34, 23]. Scatter/Gather [6] was a pioneering system that used clustering for browsing large document collections. Scatter/Gather demonstrated that clustering reveals quickly to the user the contents of the underlying collection. However, automatic clustering techniques generate clusters that are typically labeled using a set of keywords, resulting in category titles such as "*battery california technology mile state recharge impact official hour cost government*" [12]. While it is possible to understand the content of the documents in the cluster from the keywords, this presentation is hardly ideal.

Sanderson and Croft [30] presented an alternative technique for generating concept hierarchies from text, based on the concept of *subsumption*. For terms x and y from a document collection, term x subsumes y ($x \rightarrow y$) if:

$$P(x|y) > 0.8 \text{ and } P(y|x) < 1$$

where $P(x|y)$ is the probability that term x occurs in a document, given that term y does. Using the generated subsumption relations, the algorithm creates a hierarchy of terms, which has as top-level categories the general terms, and as lower-level categories terms that are more specific in nature.

While subsumption is a simple and effective technique for hierarchy construction, it has two shortcomings: (1) it requires n^2 computations of conditional probabilities, where n is the number of terms in the collection, and (2) it requires the terms to have a unique meaning (i.e., requires sense disambiguation when a term has multiple potential meaning). Sanderson and Croft sidestepped these problems by focusing only on query results and using only terms that appear more frequently in the query results than in the whole collection.[1] These heuristics guaranteed that the terms were not ambiguous and that only a small number of terms is used.

Unfortunately, these heuristics do not apply if we want to create hierarchies for arbitrary collections. Next, in Section 3, we show how our facet extraction technique reduces the ambiguity problem, and then, in Section 4, we show how we can improve the complexity of the hierarchy construction algorithm.

3. AUTOMATIC FACET EXTRACTION

One of the problems that may appear during the construction of a concept hierarchy is the fact that the same collection can be browsed in many different, orthogonal ways. Consider for example how a user can browse the schedule of TV programs. It is possible to browse by time, by TV channel, or by title. It is also possible to browse by actor, for example, or by many other dimensions. Mixing terms that belong to different dimensions might result in an awkward hierarchy: an actor might be classified under the term "Monday" because of a sitcom that is shown every Monday night. While it is a perfectly valid subsumption relation, it is not a structure useful for browsing.

In this section we present our technique for extracting important navigational facets from a collection. Since there are already collections that have metadata organized across different facets (e.g., the Corbis image collection – see Section 6.1), we can use these data to train a machine learning algorithm to classify keywords in the appropriate facets. For example, the words "cat" and "dog" are under the "Animals" facet, while the words "mountain" and "fields" are under "Topographic Features." Therefore, it is possible to use the facet as a target class and the keywords as features, in order to assign keywords to the appropriate facet. Unfortunately, such a straightforward approach does not generalize. A classifier trained in this way will classify correctly only words that have been assigned to facets before. A classifier might classify correctly the words "cat" and "dog" in the "Animals" facet, but a new word, such as "sheep," will not be assigned to any facet.

To allow our technique to generalize, we rely on the observation that keywords under different facets tend to have different "hypernyms."[2] Based on this observation, we expand each keyword using its hypernyms from a lexical corpus, such as WordNet [8]. After the expansion, each keyword is represented as a set of words. For example, the word "cat" is represented as "cat, feline, carnivore, mammal, animal, living being, object, entity". The new representation allows the classifier to generalize more easily and assign unseen words to the correct facets.

However, using hypernyms does not resolve the problem of sense disambiguation. Each word can have different meanings according to its context. Consider the word "kid," which can mean either a young person or a young goat. Before assigning this word to a facet, we have first to decide the true meaning of the word. To identify the correct sense of the word, we exploit the fact that keywords are associated with objects and each object is characterized by a set of other keywords, which provide valuable clues for the true meaning of the word. (The use of context is the basis of many techniques [21] for sense disambiguation.[3]) For example, when the word "kid" appears together with the words "goat," and "grazing" then "kid" is much more likely to refer to a young goat than to a child.

Based on the above observations, we treat facet classification as a text classification problem. In text classification [20, 7], we characterize each document using a set of words; based on the presence of these words across categories, we train a classifier to assign documents to the appropriate categories. In our case, we treat each keyword as *three sets* of words. The first set of words contain the keyword itself, the second set contains the hypernyms of all the senses of the keyword, and the third set contains the other keywords associated with the object.

[1] On average they extracted 2350 terms for hierarchy construction.

[2] Hypernym is a linguistic term for a word whose meaning includes the meanings of other words, as the meaning of vehicle includes the meaning of car, truck, motorcycle, and so on.

[3] We should emphasize that disambiguation for facet extraction is easier than the general problem of sense disambiguation. First, the context keywords are of high quality, something that is not always the case in natural language sentences. Second, and most important, while a word might have multiple senses, the senses are often closely related (see for example, the WordNet senses for "gear" and "battle"). While sense disambiguation is hard for such words, closely-related senses typically correspond to the same facet ("Generic Thing" for "gear" and "Action, Process, or Activity" for "battle"), eliminating the need for disambiguation for facet extraction.

Specifically, our algorithm, for assigning the keywords into facets, performs the following steps:

1. Get a collection D of text-annotated objects. Each object d_i has a set of associated keywords $k_{i1}, \ldots, k_{in}$ and each keyword $k_{ij}$is assigned to a facet F_{ij}.
2. For each keyword/facet pair k_{ij}/F_{ij}:
   (a) Define the facet F_{ij} as the target class.
   (b) Add the keyword k_{ij} in the first set of words.
   (c) Add the hypernyms of k_{ij} as the second set of words.
   (d) Add the other keywords associated with d_i (and their hypernyms) in the third set of words.
3. Train a document classifier over the prepared training data.

After creating the classifier, we can use it over a new set of annotated objects, to identify the facets that appear in the collection. After running the classifier over the keywords of the new objects, we can examine which facets appear frequently in the new data and use these facets for browsing. Empirically, we observed that facets that appear in 5% of the data can be proved useful for locating content of interest. One "disadvantage" of supervised learning techniques is that they cannot "discover" new types of facets. Identifying new, previously unknown dimensions for browsing is an interesting direction for future research.

In this paper, we gathered our training data from a set of annotated images from the Corbis collection (see Section 6.1), which contained a comprehensive set of facets.[4] We report the experimental setting and the results in Section 6.

4. FAST HIERARCHY CONSTRUCTION

The objective of this work is to construct automatically concept hierarchies for browsing large collections of text or text-annotated objects. Current algorithms for concept hierarchy construction do not scale well and are inadequate for creating concept hierarchies for collections with tens of thousands of documents.

4.1 Subsumption Hierarchies

Our hierarchy construction algorithm extends the subsumption-based algorithm from [30], to include the notion of *directionality* and the notion of *equivalent terms*. Two terms x and y might co-occur in all but a few documents: these terms, from a co-occurrence point of view, are equivalent. A naive use of subsumption, though, would consider one of them as "parent" term and the other as "child." The directionality property ensures that x subsumes y only when y appears in a small fraction of the documents that contain x (i.e., x is considerably "more general" than y). If two terms co-occur frequently, and none of them subsumes the other, we consider them "equivalent" for the purpose of hierarchy construction.

DEFINITION 4.1. **[Subsumption and Equivalency]** *A term x subsumes y, $(x \rightarrow y)$ if:*

- $P(x|y) > \tau_s$ and $P(x|y) > \tau_d \cdot P(y|x)$.

The terms x and y are equivalent $(x \leftrightarrow y)$ if:

- $P(x|y) \geq \tau_s$ and $P(y|x) \geq \tau_s$, and
- $\tau_d \cdot P(y|x) \geq P(x|y) \geq \frac{1}{\tau_d} \cdot P(y|x)$.

We denote with τ_d $(\tau_d > 1)$ the directionality threshold *and with τ_s $(0 < \tau_s < 1)$ the* subsumption threshold. *We define $P(x|y) = \frac{f(xy)}{f(y)}$, where $f(xy)$ is the number of objects that contain both terms x and y and $f(x)$ is the number of objects that contain term x.* □

One of the shortcomings of subsumption is its reliance on pairwise computation of conditional probabilities. Therefore the cost of a naive method is at least $O(n^2)$ where n is the number of terms. Our facet extraction algorithm from Section 3 improves this complexity by separating the keywords into different facets. When we create separate subsumption hierarchies for each facet, then the complexity is $O(m \cdot (\frac{n}{m})^2) = O(\frac{n^2}{m})$, where m is the number of facets. However, even this improvement is not sufficient when we have to work with collections that contain tens of thousands of objects and terms. Next, we present a set of techniques that can improve substantially the efficiency of the hierarchy construction algorithms.

4.2 Efficient Hierarchy Construction

In this section we present a novel algorithm that reduces substantially the number of pairwise computations. Before describing the algorithm, we describe one property of subsumption:

LEMMA 4.1. *A term x can subsume term y, only if $f(x) > \tau_d \cdot f(y)$ and $f(x) > \tau_s \cdot f(y)$, where $f(x)$ and $f(y)$ are the document frequencies of x and y, respectively.*

By exploiting this lemma, we can build hierarchies by computing a substantially smaller number of conditional probabilities, using the following algorithm:

1. Sort the terms by increasing document frequency.
2. For each term x, compute the conditional probability $P(x|y)$ for all terms y, with $f(y) \cdot \min\{1/\tau_d, \tau_s\} < f(x)$.
3. Examine which pairs x, y satisfy the subsumption requirement (Definition 4.1)

Since most keyword frequencies in most collections follow a Zipfian distribution, this algorithm needs only 25%-30% of the time needed by the basic subsumption algorithm, depending on the value of the thresholds. (We omit the proof due to space constraints.) In Section 6.3 we quantify in detail the benefits our the algorithm.

Furthermore, by construction, the graph generated by this algorithm is a directed acyclic graph. To construct the concept hierarchy, it is necessary to eliminate the forward-edges, i.e., edges that connect a term x with its "grandchildren." To perform this task we have to perform a topological sort on the graph, which increases the computational cost. To avoid creating edges that will be later eliminated, we can exploit the following lemma.

LEMMA 4.2. *If x subsumes y $(x \rightarrow y)$ and y subsumes z $(y \rightarrow z)$, then x subsumes z but the edge $x \rightarrow z$ will be a forward-edge and will be eliminated from the final hierarchy.*

Based on this lemma, we can modify the hierarchy construction algorithm to eliminate from consideration all the term pairs that form "redundant" subsumption relations (i.e., subsumptions that will be eliminated later). Finally, we can speed up considerably the identification of pairs of equivalent terms using the following lemma:

[4]The whole collection contains more than 3 million images, and approximately 40 facets.

CreateHierarchy
Extract all terms $T = \{x_1, \ldots, x_n\}$
from the objects in the database.
Sort the terms T by increasing document frequency.
foreach term $x \in T$ LocateEquivalent(x).
foreach term $x \in T$ CreateEdges(x).

LocateEquivalent(Term x)
a1:**foreach** term $y \in T$ with $f(y) > f(x)$ and
$f(y) < \min\{\tau_d, 1/\tau_s\} \cdot f(x)$
a2: **if** (x is equivalent to y)
a3: Merge x and y ($x \leftrightarrow y$)
a4: Remove y from T

CreateEdges(Term t)
// Check only terms with sufficiently smaller frequency
b1:**foreach** term $y \in T$ with $f(y) < \min\{1/\tau_d, \tau_s\} \cdot f(x)$:
b2: **if** ((x subsumes y)
AND NOT (y covered by successors of x))
b3: Create the edge $x \rightarrow y$
b4: Mark y as covered by x
b5: CreateEdges(y)

Figure 1: Creating the hierarchy

LEMMA 4.3. *Two terms x and y, with $f(x) \geq f(y)$, can be equivalent only if $f(x) \leq \tau_d \cdot f(y)$ and $f(x) \leq f(y)/\tau_s$, where $f(x)$ and $f(y)$ are the document frequencies of x and y, respectively.*

The resulting algorithm that exploits all three lemmas is shown in Figure 1. The algorithm guarantees that the resulting graph is a directed acyclic graph, with no forward edges; therefore it can be used directly to browse the collection. Furthermore, our strategy eliminates from consideration term pairs that a-priori cannot form subsumption or equivalency relations (steps a1 and b1). Additionally, the predicates in a1 and b1 are easy to implement in relational database systems, which have been specially optimized to execute such band joins. The only requirement is to keep a table with the term frequencies and build an index on the frequency field. We implemented our prototype on top of a relational database system (Microsoft SQL Server 2000) using mainly SQL statements for hierarchy construction. The presented algorithm works best when low-frequency terms (which are often highly-specific) are subsumed "early" by other terms. This is fortunately the case for most of the cases where it is possible to extract a meaningful hierarchy from the collection.

4.3 Extracting Terms

So far, we have described the hierarchy construction algorithm assuming that each object comes associated with a set of keywords. For some object types (e.g., annotated images) the associated keywords are highly descriptive and can be used directly with no further steps. The keyword extraction process, though, is more complicated when the object annotation is in free text form. One approach would be to use as terms all the words that appear in the collection. Still, even after removing stopwords and filtering out infrequent words, uninformative words like "slept," or "approximately" appeared in the generated hierarchies.

To eliminate such problems, we decided to keep as terms only the nouns and noun phrases that appear in the free text annotations. This choice is supported by earlier studies [11], which indicated that nouns and noun phrases are good features for constructing clusters of text documents. While parsing to identify nouns and noun phrases adds an additional overhead in the hierarchy construction process, it also reduces substantially the overall number of terms. This reduction results in large gains of efficiency, as we have discussed in Section 4.2. Additionally, nouns and noun phrases are typically better suited for titles of categories, and this results in a better browsing experience.

5. RANKING CATEGORIES

Consider a user browsing a collection using a small-screen device, such as a smartphone. Typically, the screen cannot display more than 5-10 categories. Thus, if a category has many children then the user cannot see all the children at once. The most prevalent solution to this problem is to paginate the results, showing the "best" categories first. Ranking categories is a non-trivial problem; the lack of explicit user goals (characteristic of browsing) makes the problem even harder.

In this section, we propose a set of techniques for ranking the children of a category. Section 5.1 presents a frequency-based ranking, which serves as our baseline. Then, Section 5.2 presents our set-cover-based approach, which tries to maximize coverage of the collection. Finally, Section 5.3 presents our merit-based approach, which ranks higher categories that expose the largest fraction of the collection with the lowest cost for the user.

5.1 Frequency-based Ranking

This technique simply ranks categories by the number of objects that are classified under the category. Categories that contain the larger number of objects are ranked higher. This heuristic allows the user to see first categories with the greatest wealth of information. Moreover, it guarantees that low-ranked categories (that might not appear on screen) represent only a small fraction of the collection.

This ranking scheme is very easy to implement. Furthermore, if the categories do not overlap (i.e., no object is classified under multiple categories) then this ranking scheme is optimal in terms of collection coverage. Unfortunately, this is not always the case. When categories overlap, then the frequency-based ranking may become sub-optimal by presenting first categories with highly overlapping content. The ranking scheme that we present next resolves this problem.

5.2 Set-cover Ranking

The objective of *set-cover ranking* is to maximize the number of *distinct* objects that are accessible from the top-k ranked categories. In other words, this ranking tries to maximize the cardinality of the set $o(C_1) \cup \ldots \cup o(C_k)$, where $C_1 \ldots C_k$ are the top-k categories and $o(C_i)$ is the set of objects classified under the category C_i.

This problem is an instance of the *set-cover* problem, which is a well-known NP-complete problem [5]. Consequently, the optimal ranking, which exposes the maximum fraction of the collection, would take time exponential to the number of categories. For our purposes, though, the optimal solution is unnecessary: not only it is unjustifiably expensive, but it also has the unfortunate property of generating "non-monotonic" rankings. That is, the top-k categories are not necessarily a subset of the set with the top-$k+1$ categories. This property can lead to non-intuitive interface behavior: after enlarging the browser to show larger parts of the hierarchy, categories that used to be highly-ranked might disappear.

To avoid both the complexity and the non-monotonicity issues, we decided to use a greedy algorithm for category ranking. The greedy algorithm is the best polynomial algorithm for approximating the set-cover problem, and runs in time linear to the number of categories. Specifically, the algorithm works as follows:

1. Mark all the objects as uncovered.
2. Select category C with the largest number of *uncovered* objects.
3. Mark all the objects classified under C as *covered*.
4. If all objects covered or ranked k categories, stop; else go to Step 2.

Using this greedy algorithm we can quickly compute the top-k categories and ensure that the displayed categories cover a large fraction of the underlying collection. If we want to see more categories, the algorithm can easily resume and rank the remaining categories.

Both the set-cover and the frequency-based algorithm try to maximize the number of objects that are covered by the displayed, top-k categories, regardless of how easy it is to access these objects. Next, we present an alternative approach that considers the structure of the underlying hierarchy and the respective effort that the user has to put to locate items of interest.

5.3 Merit-based Ranking

The ranking methods presented so far focused only on covering as many objects as possible. The *merit-based* ranking method that we present now, takes into consideration the structural properties of the sub-hierarchies under the categories selected. Specifically, the *merit-based* method ranks higher categories that enable users to access their contents with the smallest cost, on average.

Before describing our merit-based approach, we first define the *cost* of locating an object n, classified in a hierarchy H (H is a directed acyclic graph.) We assume for simplicity that n is reachable through only one path from the root of the hierarchy and denote the path as $C_1 \rightarrow \ldots \rightarrow C_h$. Category C_1 is the root node of H and C_h is the leaf node that contains the object n. We treat the browsing activity of a user as a random walk on the hierarchy H. We denote with $T(C_i)$ the time required to reach n from the node C_i. In general, the time spent to reach n consists of three components:

- *Reading the category headings*: While browsing the contents of a category C_i, the user spends time linear to $b(C_i)$ to read the descriptions of the subcategories of C_i, where $b(C_i)$ is the number of children of C_i. Hence, to browse through the children of C_i the user spends time $\kappa b(C_i)$, where κ is a constant, equal to the time needed to read the title of a category.
- *Correcting mistakes*: The user does not always make the correct decision about the child of C_i that leads to n. We assume that with an error probability $P_e(C_i)$ the user selects a wrong subcategory of C_i. In that case, we assume that the user realizes that the chosen subcategory is the wrong one, returns to C_i, and tries again.[5] Thus,, the user spends time $P_e(C_i) \cdot T(C_i)$ recovering from a mistake.
- *Browsing the correct subtree*: The user selects with probability $1 - P_e(C_i)$ the correct sub-category C_{i+1}. Therefore, the user spends time $(1 - P_e(C_i))T(C_{i+1})$ browsing through the correct subtree of C_i, which leads to n.

Hence, the time $T(C_i)$ that a user spends on C_i is:

$$\begin{aligned} T(C_i) &= \kappa b(C_i) + P_e(C_i) \cdot T(C_i) + (1 - P_e(C_i))\, T(C_{i+i}) \\ &= \frac{\kappa b(C_i)}{1 - P_e(C_i)} + T(C_{i+i}) \end{aligned} \quad (1)$$

Therefore, the time spent browsing through the whole hierarchy H to reach n is:

$$T(H) = T(C_1) = \sum_{i=1}^{h} \frac{\kappa b(C_i)}{(1 - P_e(C_i))} \quad (2)$$

Using this methodology, we can compute the average cost for accessing any object classified under the hierarchy H_i rooted at category C_i. It is clear from Equation 2 that hierarchies H with large number of children and long paths have large $T(H)$. However, these are the categories that contain the largest number of objects. To avoid unfairly penalizing such categories, we introduce the *merit(C)* metric for a category C:

$$merit(C) = \frac{2 \cdot \frac{1}{T(C)} \cdot o(C)}{\frac{1}{T(C)} + o(C)} = \frac{2 \cdot o(C)}{1 + T(C) \cdot o(C)} \quad (3)$$

where $o(C)$ is the number of distinct objects classified under C and $T(C)$ is defined in Equation 2. This metric is similar to the F_1-measure (which favors high precision and recall) and favors categories with low cost $T(C)$ and large number of objects $o(C)$. It should be noted that the *merit* can be computed very efficiently in a bottom-up fashion.

Using the *merit* of each category, we can rank categories appropriately, putting first categories that have good hierarchy structures under them and provide access to a large number of objects.

6. EXPERIMENTAL RESULTS

In this section present the experimental evaluation of our techniques. We first describe the data sets that we used (Section 6.1). Then, in Section 6.2, we evaluate our facet extraction technique of Section 3; in Section 6.3, we examine the efficiency of our hierarchy construction algorithm of Section 4; and in Section 6.4 we evaluate the structural properties of the generated hierarchies. Finally, in Section 6.5 we provide further discussion, outlining possible future directions.

6.1 Data Sets

- **Corbis**: This is a set of 36,820 annotated images, taken from the Corbis royalty-free collection[6]. Each image has a title, a free-text description, and a set of keywords associated with it. Each keyword is manually assigned by the Corbis annotators to one of the 38 facets that are used from Corbis. The *Corbis* data set has a total of 65,521 keywords, mainly assigned to 14 out of the 38 facets. The remaining 24 facets had less than 100 keywords assigned to them and we ignored them during our evaluation.

  We also used the Corbis collection to test our facet extraction algorithm of Section 3. Since our algorithm relies on the existence of pre-annotated data, we picked

[5] We can add an extra cost for browsing the contents of the wrong category and then returning. For simplicity we do not explore this direction further in this paper.

[6] `http://www.corbis.com`

11,000 keywords and their associated facets from the Corbis collection and to train and test our algorithm. To avoid any bias, we randomly picked the 11,000 keywords from 11,000 randomly selected images (one keyword per image).

- **XMLTV**: This set contains all the television programs broadcasted over 261 channels on the RCN New York City digital cable system[7] over a period of eleven days (Jan/9/2005 – Jan/19/2005). We used the XMLTV utilities[8] to retrieve the schedules and we augmented the annotation with data from the Internet Movie Database[9]. This resulted 29,975 programs, organized in separate XML files that follow the XMLTV DTD format. The each program had a total of 12 facets (actor, category, channel name, commentator, description, director, guest, presenter, showtime, title, writer). After processing the different fields, we extracted a total of 58,702 keywords.

- **DMOZ**: This set contains real web pages from Open Directory[10], manually classified by human editors in a manually created hierarchy. To run extensive experiments keeping the running times manageable, we restricted our data set to a random sample of 19,392 pages from the "Health" and "Science" top-level categories. Each page contains a human-edited title and description, as well as the actual text in the web page[11]. After processing the different fields, we extracted a total of 80,810 keywords.

6.2 Effectiveness of Facet Extraction

We evaluated our facet extraction algorithm of Section 3 using the keywords from the *Corbis* data set. For classifier we used Support Vector Machines (SVM) with linear kernels.

Initially, we tested the accuracy of the classifier without using the WordNet hypernyms and without using the keywords associated with the same image. The classifier, as expected, could not generalize. The accuracy (as measured by the F_1-measure) was 10%, only slightly above the accuracy of a random classifier. By adding the hypernyms, the performance improved considerably, reaching an average F_1-measure of 71% (detailed results omitted due to space constraints.) This improvement confirmed our hypothesis that hypernyms are useful features for allocating keywords to facets. Nonetheless, the sense ambiguity is still a problem in this case: after adding as extra features the remaining keywords from each document, the classification performance improved considerably, reaching an average F_1-measure of 81% (see Table 1).

We also wanted to compare our method against variations of other techniques. One hypothesis was that we can create facets by picking some high-level hypernyms from WordNet, which can serve as root nodes for the corresponding facets. For example, the term "animal/fauna" in WordNet could serve as the root node for the "Animal" facet. Subsequently, all terms that have "animal/fauna" as a hypernym could be assigned to the "Animal" facet. (This approach would be close in spirit with the hierarchy construction algorithm in [31].) To test the accuracy of this approach, we trained RIPPER [4], a *rule-based* classifier, using the keywords and their hypernyms as features. The average F_1-measure in that case was close to 55%, significantly worse than the corresponding results for SVMs. The results also highlighted that some classes work well with simple, rule-based assignments of terms to facets, but there are other classes that need more elaborate classifiers. For example, for the facet GAN (**G**eneric **AN**imals) the rule-based classifier resulted in an F_1-measure of 93.3%, showing that simple rules work well for this facet. However, for the APA facet (**A**ction, **P**rocess, or **A**ctivity) the F_1-measure was only 35.9%, showing that simple rules do not work well for such a complicated facet.

Class	Precision	Recall	F_1-measure
GTH	87.70%	83.00%	85.29%
APA	75.80%	75.80%	75.80%
ATT	78.20%	83.50%	80.76%
ABC	85.20%	87.60%	86.38%
GCF	74.70%	76.76%	75.72%
NCF	82.40%	87.57%	84.91%
GTF	86.70%	75.00%	80.43%
GPL	81.70%	90.10%	85.69%
ATY	80.00%	81.30%	80.64%
GEV	79.40%	56.30%	65.88%
GAN	92.90%	92.90%	92.90%
RPS	85.60%	76.30%	80.68%
NTF	82.40%	80.30%	81.34%
NORG	75.40%	76.58%	75.99%
Average	**82.01%**	**80.22%**	**80.89%**

Table 1: The average performance of the facet extraction technique for each of the 14 facets in the *Corbis* data set. (Results obtained using 10-fold cross-validation.)

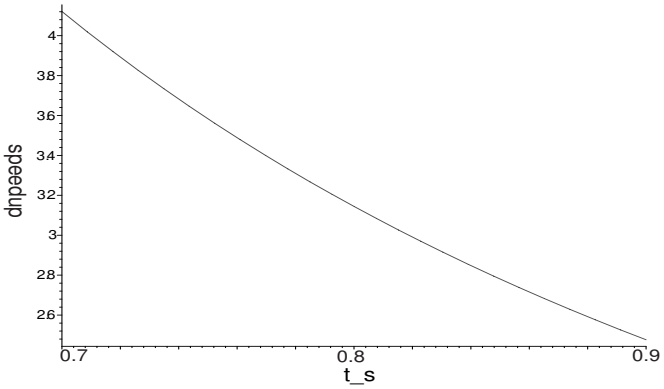

Figure 2: The speedup of the our hierarchy construction algorithm over the basic subsumption algorithm, for different values of the subsumption threshold τ_s, for the DMOZ dataset and for $\tau_d = 1.2$.

6.3 Efficiency of Hierarchy Construction

We ran an extensive set of experiments examining the effect of the different parameters of our technique on the running time of the algorithm.

Initially, we examined the speedup obtained after eliminating from consideration terms with low frequency in the collection (less than a threshold τ_f). We observed that the time required for constructing the hierarchies decreased *exponentially* with the threshold τ_f. Based on these results about the quality of the hierarchies, we decided to set $\tau_f = 16$, which resulted in efficient executions and good quality of the hierarchies (see Section 6.4).

Then, we examined the effect of the thresholds τ_s and τ_d (see Definition 4.1 in Section 4) on the efficiency of our al-

[7] `http://www.rcn.com`
[8] `http://membled.com/work/apps/xmltv`
[9] `http://www.imdb.com`
[10] `http://www.dmoz.org`
[11] We kept only the text from each page by stripping the HTML tags using the "*lynx –dump*" command.

	τ_d			
DataSet	**1.0**	**1.2**	**1.5**	**2.0**
Corbis	24.3 (43%)	18.1 (52%)	19.5 (54%)	21.9 (54%)
XMLTV	8.5 (30%)	6.4 (37%)	7.8 (39%)	8.2 (39%)
DMOZ	10.4 (0.4%)	10.2 (1.4%)	11.4 (1.9%)	11.8 (1.9%)

Table 2: The time required to build a concept hierarchy (in minutes) as a function of the directionality threshold τ_d ($\tau_s = 0.8$). In parentheses the percentage of the terms that were matched as "equivalent" during the "LocateEquivalent" step.

DataSet	**Frequency**	**Set-cover**	**Merit-based**
Corbis	0.835	0.924	0.898
XMLTV	0.812	0.931	0.900
DMOZ	0.627	0.749	0.713

Table 3: The average coverage of the collection, for the different data sets and different ranking methods. ($\tau_s = 0.8, \tau_d = 1.2$)

gorithm. We observed that execution time decreases as the subsumption threshold τ_s becomes smaller. This was an expected outcome according to Lemma 4.1: small values of τ_s disallow the formation of subsumption relations between terms with similar frequencies.

The effect of the directionality threshold τ_d is more mixed (see Table 2). Large values of τ_d force the algorithm of Figure 1 to perform more comparisons to locate equivalent terms, slowing the execution of the "LocateEquivalent" procedure. However, large values of τ_d identify more "equivalent" terms, which in turn reduces considerably the number of terms used by the "CreateEdges" procedure. In our data sets, we identified a value of $\tau_d \approx 1.2$ to be a good choice, a choice reinforced by the experiments of Section 6.4.

Finally, Figure 2 shows the speed improvement of our algorithm over the basic subsumption algorithm, for different values of τ_s and $\tau_d = 1.2$, for the DMOZ data set. For the value $\tau_s = 0.8$, the algorithm runs 3 times faster than the basic subsumption algorithm. The speedups are higher for the XMLTV and Corbis datasets that benefit more from the reduction of nodes from the "LocateEquivalent" step of our algorithm of Figure 1. (See Table 2 for the statistics.)

6.4 Structural Evaluation of Hierarchies

Beyond efficiency, another important aspect of hierarchy construction is the quality of the generated hierarchies. For this purpose, we examined the structural properties of the generated hierarchies and examined how these properties can affect the browsing experience. In the future we plan to conduct a study of the user interface using human subjects, but the results of such a study are out of the scope of this paper. (See Section 6.5 for some anecdotal results.) We examined the following properties of the generated hierarchies:

Coverage: We define coverage as the fraction of the collection objects that are reachable using the hierarchy. Unreachable objects are those annotated only with terms that do not appear in the hierarchy. The perfect hierarchy has coverage is equal to 1, allowing the user to reach all objects of the hierarchy. We used coverage as one of the metrics to evaluate the ranking schemes of Section 5. We show the results in Table 3 for $\tau_s = 0.8$ and $\tau_d = 1.2$ and after selecting 10 categories per node. (The results were similar for other values of the thresholds.) While all methods achieve a relatively good coverage, the *set-cover* method (as expected) consistently covers larger fraction of the collection compared to the other two methods.

DataSet	**Frequency**	**Set-cover**	**Merit-based**
Corbis	42.99	30.62	22.05
XMLTV	33.57	29.69	26.98
DMOZ	40.61	39.78	35.30

Table 4: The average cost for reaching an object using hierarchies created by different ranking methods, computed using Equation 2 ($\kappa = 1, P_e(\cdot) = 0.2$).

The *merit-based* method performs slightly worse than the *set-cover*, an expected result since *set-cover* is explicitly designed to optimize the coverage of the hierarchy.

Cost: In addition to coverage, we also measured additional structural characteristics of the hierarchy, such as the *average path length* to find an object (shorter paths are preferable), and the *average branching factor* (small branching factors are preferred, since users can decide faster which category is best). Since configurations that optimize average path length tend to have larger branching factors, we decided to use the *cost* metric from Section 5.3 to combine meaningfully these metrics. Table 4 summarizes the results. *Merit-based* hierarchies consistently perform better than the other two approaches, decreasing by 10-50% the time needed to locate items of interest, compared to the other approaches.

Conclusions: In general, *merit-based* performs very well and offers fast access to the contents of the collection. Additionally, *merit-based* rankings are efficient to implement on top of relational database systems, while the *set-cover* rankings typically take longer to compute.

6.5 Further Discussion

To experience first-hand the quality of the hierarchies, we built a hierarchy browser and we used it to explore the contents of multiple collections. We also experimented with different methods for presenting the hierarchies (e.g., using RSVP techniques for text [9]) but analyzing the user interface is out of the scope of this paper.

Our informal results suggest that keyword extraction is vital for generating good quality hierarchies. For example, for the Corbis and XMLTV data sets, the objects contained high-quality keywords that in turn helped generate high-quality hierarchies that were informative and easy to use. For the DMOZ dataset, the quality of the hierarchies depended on the quality of the textual content in each web page. Web pages with little or no quality text were difficult to characterize properly and, correspondingly, difficult to organize in a hierarchy. We believe that more advanced techniques for keyword indexing (e.g.,[32, 13] for text or [1, 14] for images) and standard techniques for web indexing (e.g., use of anchor text) can substantially improve the hierarchies.

7. RELATED WORK

Several methods have been proposed for browsing collections of text or multimedia documents. Scatter/Gather [6] demonstrated that clustering can be used for browsing large collection of text documents. Multiple methods (e.g., [34, 2, 23]) have been proposed for clustering text and web data. The difficulty of meaningfully labeling the clusters led to the introduction of concept hierarchies: Sanderson and Croft [30] introduced the subsumption hierarchies and Lawrie and Croft [18] showed experimentally that subsumption hierarchies outperform lexical hierarchies [25, 26, 27]. Lawrie et al. [17, 19] suggested a technique for identifying terms that are useful for hierarchy construction. The algorithm in [17] is similar to the

algorithm used by the *set-cover* ranking. The main difference is the optimization goal: *set-cover* tries to optimize the coverage of the collection from the hierarchy, while the algorithm in [17] tries to maximize the (weighted) coverage of the collection vocabulary.

Kominek and Kazman [15] use the hierarchical structure of WordNet [8] to offer a hierarchy view over the topics covered in videoconference discussions. Stoica and Hearst [31] also use WordNet together with a tree-minimization algorithm to create an appropriate concept hierarchy for a collection. Barnard et al. [1] used WordNet to disambiguate the keywords associated with each image, and to generate clusters of higher quality. As an alternative to creating a separate hierarchy for each collection, Chaffee and Gauch [3] presented a system that uses a personalized ontology to offer a common browsing experience across collections of web pages (i.e., web sites) that organize their contents in different ways. Other, less common browsing structures were proposed (e.g., wavelet-based text visualization [24], dynamic document linking [10]) but clustering and hierarchy-based approaches continue to be the most popular interfaces for browsing.

Faceted interfaces, which use multiple, orthogonal classification schemes to present the contents of a database, become increasingly popular. A large number of e-commerce web sites use faceted interfaces [16], based on engines provided by companies such as Endeca[12] and Mercado[13], which expose the facets that are already defined for the products (e.g., "by price," "by genre" and so on). Academically-developed systems, such as Flamenco [33], HiBrowse [28], and OVDL [22], demonstrated the superiority of faceted interfaces over single hierarchies. Our work on automatic construction of multifaceted interfaces contributes to this area and facilitates the deployment of faceted databases. In an orthogonal direction, Ross and Janevski [29] presented work on *searching* faceted databases and described an associated entity algebra and an query engine.

8. CONCLUSION

We presented methods for automatically constructing multifaceted hierarchies, and methods for selecting the best parts of the generated hierarchies when it is not possible to fit all the categories on screen. Our experiments with real-life data sets indicate that automatic construction of multifaceted interfaces is feasible, and generates high-quality hierarchies. We are interested in exploring different ways of presenting the hierarchies to expose the contents of the collection in efficient ways. Furthermore, we are interested in integrating better browsing and searching in multifaceted databases. Creating the appropriate indexing structures to support concurrent searching and browsing is a promising direction for future research.

Acknowledgements

The authors would like to thank Gavin Smyth of Microsoft Research Cambridge for his work on the implementation of the hierarchy browser.

9. REFERENCES

[1] K. Barnard, P. Duygulu, and D. A. Forsyth. Clustering art. In *CVPR*, pages 434–441, 2001.

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW6*, pages 1157–1166, 1997.

[12] `http://www.endeca.com`

[13] `http://www.mercado.com`

[3] J. Chaffee and S. Gauch. Personal ontologies for web navigation. In *CIKM*, pages 227–234, 2000.

[4] W. W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[6] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *SIGIR*, pages 318–329, 1992.

[7] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM*, pages 148–155, 1998.

[8] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, May 1998.

[9] M. Goldstein, G. Öqvist, M. Bayat-M, P. Ljungstrand, and S. Björk. Enhancing the reading experience: Using adaptive and sonified RSVP for reading on small displays. In *Mobile HCI*, 2001.

[10] G. Golovchinsky. Queries? Links? Is there a difference? In *CHI*, pages 407–414, 1997.

[11] V. Hatzivassiloglou, L. Gravano, and A. Maganti. An investigation of linguistic features and clustering algorithms for topical document clustering. In *SIGIR*, pages 224–231, 2001.

[12] M. A. Hearst and J. O. Pedersen. Rexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *SIGIR*, pages 76–84, 1996.

[13] A. Hulth. Reducing false positives by expert combination in automatic keyword indexing. In *RANLP*, pages 367–376, 2003.

[14] J. Jeon, V. Lavrenko, and R. Manmatha. Automatic image annotation and retrieval using cross-media relevance models. In *SIGIR*, pages 119–126, 2003.

[15] J. Kominek and R. Kazman. Accessing multimedia through concept clustering. In *CHI*, pages 19–26, 1997.

[16] K. La Barre. Adventures in faceted classification: A brave new world or a world of confusion? In *ISKO*, 2004.

[17] D. Lawrie, W. B. Croft, and A. Rosenberg. Finding topic words for hierarchical summarization. In *SIGIR*, pages 349–357, 2000.

[18] D. J. Lawrie and W. B. Croft. Discovering and comparing hierarchies. In *RIAO*, pages 314–330, 2000.

[19] D. J. Lawrie and W. B. Croft. Generating hierarchical summaries for web searches. In *SIGIR*, pages 457–458, 2003.

[20] D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. In *SIGIR*, pages 298–306, 1996.

[21] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[22] G. Marchionini and G. Geisler. The Open Video Digital Library. *D-Lib Magazine*, 8(12), Dec. 2002.

[23] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods. *Machine Learning*, 42(1/2):9–29, 2001.

[24] N. E. Miller, P. C. Wong, M. Brewster, and H. Foote. Topic islands: A wavelet-based text visualization system. In *VIS*, pages 189–196, 1998.

[25] C. G. Nevill-Manning, I. H. Witten, and G. W. Paynter. Lexically-generated subject hierarchies for browsing large collections. *International Journal on Digital Libraries*, 2(2-3):111–123, 1999.

[26] G. W. Paynter and I. H. Witten. A combined phrase and thesaurus browser for large document collections. In *ECDL*, pages 25–36, 2001.

[27] G. W. Paynter, I. H. Witten, S. J. Cunningham, and G. Buchanan. Scalable browsing for large collections: A case study. In *ACM DL*, pages 215–223, 2000.

[28] A. S. Pollitt. The key role of classification and indexing in view-based searching. In *IFLA*, 1997.

[29] K. A. Ross and A. Janevski. Querying faceted databases. In *Proceedings of the Second Workshop on Semantic Web and Databases*, 2004.

[30] M. Sanderson and W. B. Croft. Deriving concept hierarchies from text. In *SIGIR*, pages 206–213, 1999.

[31] E. Stoica and M. A. Hearst. Nearly-automated metadata hierarchy creation. In *HLT-NAACL: Short Papers*, pages 117–120, 2004.

[32] P. D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

[33] K.-P. Yee, K. Swearingen, K. Li, and M. A. Hearst. Faceted metadata for image search and browsing. In *CHI*, pages 401–408, 2003.

[34] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *SIGIR*, pages 210–217, 2004.