An obvious and natural approach to organizing a large corpus of data is a hierarchical index--akin to a book's table of contents. The type of corpus dealt with here is a bibliographical repository, with entries form a limited domain. Given such an index, it is desirable that search results point to relevant locations in the hierarchy, rather than just providing a flat list of entries. This is useful not only to support user searching, but also as an aid suggesting possible places to link new entries that are inserted into the repository. BoW is an online bibliographical repository based on a hierarchical concept index to which entries are linked. Searching in the repository should therefore return matching topics from the hierarchy, rather than just a list of entries. Likewise, when new entries are inserted, a search for relevant topics to which they should be linked is required. The study develops a vector-based algorithm that creates keyword vectors for the set of competing topics at each node in the hierarchy, and show how its performance improves when domain-specific features are added (such as special handling of topic titles and author names). The results of a 7-fold cross validation on a corpus of some 3,500 entries with a 5-level index are hit ratios in the range of 89-95%, and most of the misclassifications are indeed ambiguous to begin with. (Contains 34 references.) (Author/AEF)
Hierarchical Indexing and Document Matching in BoW

Maayan Geffet and Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University, 91904 Jerusalem, Israel
{mary,feit}@cs.huji.ac.il

1. INTRODUCTION

An obvious and natural approach to organize a large corpus of data is a hierarchical index — akin to a book’s table of contents. The type of corpus we deal with is a bibliographical repository, with entries from a limited domain (our prototype is on “parallel processing”). Given such an index, it is desirable that search results point to relevant locations in the hierarchy, rather than just providing a flat list of entries. This is useful not only to support user searching, but also as an aid suggesting possible places to link new entries that are inserted into the repository.

1.1 BoW – Bibliography on the Web

The goal of the BoW project [9] is to create a convenient environment for using and maintaining an on-line bibliographic repository. The key idea is that this be a communal effort shared by all the users. Thus every user can benefit from the input and experience of other users, and can also make contributions. In fact, the system tabulates user activity, so merely searching through the repository and exporting selected items already contributes to the ranking of items in terms of user interest. A prototype implementation is available at http://www.bow.cs.huji.ac.il.

The heart of the BoW repository is a deep (multi-level) hierarchical index spanning the whole domain. The nodes in the hierarchy are called concept pages. Pages near the top of the hierarchy represent broad concepts, while those near the bottom represent more narrow concepts. The depth of the hierarchy should be sufficient so that the bottommost pages only contain a handful of tightly related entries (as opposed to Web search engines and scientific literature databases like CORA [5] which contain a relatively shallow directory). A subtree containing all the concept pages reachable from a certain (high level) concept page is referred to as a topic. Entries can be linked to multiple concept pages, if they pertain to multiple concepts. Likewise, they can be linked at different levels of the hierarchy, depending on their breadth and generality.

The index is navigated using a conventional browser. Normally three frames are available (Fig. 1). The first shows the hierarchical index, and the currently selected concept page. The second lists entries linked to this concept page, and allows for the selection of a specific entry. The third displays the surrogate of the chosen entry, including all the bibliographical data (authors, title, where and when published), user annotations, and additional links (e.g. to the full text is available). Available operations on the current entry include marking it for export, adding an annotation, and adding links. This includes links from additional concept pages to the entry, links between this entry and related entries (e.g. from a preliminary version of a paper to the final version), and links to external resources such as the full text.

The index structure is created by the site editor. The vocabulary used in the index and annotations is uncontrolled by the system, and users also query the system using natural language [2]. Indexing is simplified by the fact that we use concise surrogates, rather than full text documents [13]. We make up for the reduction in data by enlisting users to verify indexing suggestions. Thus, when a user introduces a new entry, the system uses the text of the entry as a query, and finds concept pages that contain similar entries. But the actual decision to link the new entry to these concept pages is left to the discretion of the user.

The indexing described in this paper is based on lexical analysis of concept pages and entries linked to them. For each topic, we create a list of keywords that differentiate it from other topics that have the same parent. The indexing then proceeds from the root, choosing the most suitable sub-topic(s) at each point. As only contending topics are considered, the complexity of the search is reduced [14, 20].

1.2 Related Work

There are three basic approaches for textual documents processing [15]: lexical, syntactic, and semantic analysis. A number of systems using syntactic and semantic analysis have been developed and are being used for research, such as DR-LINK [18], CLARIT [8] and TREC [7, 31]. However, they are typically not significantly
better than the best lexical analyzers. We will discuss various lexical analyzers throughout the paper, in relation to our work.

Very little has been done so far on hierarchical indexing. In general, it has been shown that hierarchical indexing methods outperform traditional flat algorithms [20, 14]. However, these studies were based on a very wide domain and a relatively shallow hierarchy (e.g., two levels). Our work, in contrast, requires a very fine classification, as the bottom levels of the hierarchy only contain a small number of entries each.

Search and browsing based on a hierarchy was suggested in [24]. However, in this case the hierarchy is very strict and depends on nested key phrases (e.g., “forest fires” is under “forest”), which allows it to be automated. We take the opposite approach: the hierarchy is created by humans so as to capture pertinent concepts, and the automation comes in trying to find what characterizes this structure.

2. OFF-LINE PREPARATION OF KEYWORD VECTORS

The hierarchical indexing mechanism consists of two parts. The first is an off-line traversal of the whole repository, repeated at reg-
Therefore create a unified vocabulary that includes all the words that
vocabulary vectors of these topics must span the same space. We
summed as described below. The counters indicating how many times each word appears are
includes the complete vocabulary of the topic:

\[ \text{VOCtopic} = \text{VOCpage}_1 \cup \text{VOCpage}_2 \cup \ldots \cup \text{VOCpage}_n. \]

The counters indicating how many times each word appears are
in order to compute keyword vectors for all the topics. The second is a matching scheme that compares
new entries or queries with these pre-computed keyword vectors.
The off-line part is executed recursively for every level of the
index, top-down. The main idea is that each topic encompasses
all the concept pages in a sub-tree of the index, therefore all of
them should be taken into account while constructing its keywords
vector. The group of sibling topics, located at the same level and
having the same parent in the index are called a competitive topics
set, since they compete for keywords with each other. The algo-

Algorithm generates keywords vectors in five steps: parse all the pages
in the topic's sub-tree, merge them into one vector, unify the result-
ving vectors to include the same words, normalize the weights of
the words in all the vectors, and choose the most relatively frequent
ones to represent the corresponding topics.

2.1 Parsing

The first stage is parsing the text of concept pages, with the goal
of creating a vector of all the words in the given concept page [30,
33], denoted by \( \text{VOCpage} \). This of course requires us to define
"word".

The natural definition is a completely separated meaningful string.
This has the well-known disadvantages of treating related words as
being different, and the well-known solutions such as stemming
(e.g. [23, 19]). An alternative is to use \( n \)-grams (substrings of
length \( n \) of words: for example, "algorithm" will be turned into
"algor", "igor", "gorit", "orith", and "rithm") [1]. We prefer the latter,
and specifically use 5-grams, based on a separate study\(^1\) in
which documents were clustered automatically based on similarity
and this was compared with manual clustering (Table 1). But in
order to avoid 5-grams that are largely based on common suffixes
and therefore meaningless, we also use stemming first.

Note that longer words are represented by more 5-grams in the
vocabulary vector than shorter ones, which gives them more weight
in the comparisons. Thus it would be interesting to check if similar
results would be obtained by using whole words, and weighting
them according to length.

In any case, from now on the word "word" will mean a 5-gram.

2.2 Merging

After parsing all the concept pages in a topic's sub-tree, the re-
resulting vocabulary vectors are merged. The resulting vector in-
cludes the complete vocabulary of the topic:

\[ \text{VOCtopic} = \text{VOCpage}_1 \cup \text{VOCpage}_2 \cup \ldots \cup \text{VOCpage}_n. \]

The counters indicating how many times each word appears are
summed as described below.

2.3 Unification

In order to compare a query with a set of competitive topics, the
vocabulary vectors of these topics must span the same space. We
therefore create a unified vocabulary that includes all the words that

---

\(^1\) In cooperation with E. Boncheck.
Keywords Selection Heuristic

A keyword is a word that characterizes a concept and differentiates one topic from others [15]. Thus, in order to decide whether a word is a keyword of some topic, one should consider its frequency (weight) in this topic, and also compare with its weights in all the competitive topics. The basic idea is that if a word is extremely frequent in one particular topic and relatively rare in others, then we may use it as a keyword for this topic. If a word has similar weight in all the topics, then it does not represent any of them, even if its weight is high [29].

One way to assess the discriminatory power of a word is based on the difference between its maximal and minimal counter values in different topics in the competitive set. More formally, the algorithm is as follows (where NormVoc_t(w) denotes the counter value for word w in the normalized vector of topic t):

1. For each topic in the competitive set, find those words that achieve their maximal counter value in this topic:
   \[ \text{Max}_t = \{ w | \forall i, i \neq t : \text{NormVoc}_i(w) > \text{NormVoc}_t(w) \}. \]

2. For these words, find the range of counter values:
   \[ \text{Diff}(w) = \max_{i \neq t} (\text{NormVoc}_i(w) - \text{NormVoc}_t(w)). \]

3. Sort the words in \( \text{Max}_t \) according to \( \text{Diff}(w) \) in a descending order.

4. Choose the top 10% of the words (those with the biggest difference values) and place them in the keywords vector \( \text{Keywords} \).

A possible problem with this definition is that the difference can be large because the minimal value is very small. An alternative is therefore to use the difference between the two top counter values in step 2. The definition then becomes

\[ \text{Diff}(w) = \max_{i \neq t} (\text{NormVoc}_i(w) - \text{NormVoc}_t(w)). \]

This version selects the words with significantly greater weight in one particular topic than in all the others, but may miss cases in which a word has a high count in 2 or 3 topics (which may happen as shown in Fig. 4). Specifically, in the BoW corpus the gap between the two highest values is the largest in 65-79% of the cases, but the gap between the 2nd and 3rd is the largest in another 15-22%.

Another disadvantage of this heuristic is the percentage of words to be chosen as the most significant: we decided to choose an empirically-determined 10% threshold, but maybe for other repositories it will be reasonable to use another threshold. An alternative is to choose the most significant words according to their statistics. Specifically, we propose to select those words whose counter value is larger than the average plus one standard deviation:

1. For each word calculate the average counter value:
   \[ \text{average}(w) = \frac{1}{n} \sum_{1 \leq i \leq n} \text{NormVoc}_i(w) \]
   (where \( n \) is the number of topics in the competitive set).

2. Calculate the standard deviation:
   \[ \text{std.dev}(w) = \sqrt{\frac{1}{n} \sum_{1 \leq i \leq n} (\text{NormVoc}_i(w) - \text{average}(w))^2}. \]

3. If \( \text{max.weight}(w) > \text{average}(w) + \text{std.dev}(w) \) then the word \( w \) is a keyword of the maximal weight topic, otherwise it does not represent any topic since it is almost equally frequent in all of them.

To check if the word should be a keyword for other topics as well, the highest value is removed and the procedure repeated for the remaining topics.

To compare the above heuristics we used them to classify 200 entries from the BoW prototype repository. The results are shown Table 2, and indicate that the last heuristic (using the average and standard deviation) is the best.

### Table 2: Correct classification rate when using alternative heuristics for keyword selection.

<table>
<thead>
<tr>
<th>Heuristic used</th>
<th>Hit ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>extreme values differences</td>
<td>48%</td>
</tr>
<tr>
<td>two highest values differences</td>
<td>62%</td>
</tr>
<tr>
<td>if (max_weight &gt; avg+std_dev)</td>
<td>87%</td>
</tr>
</tbody>
</table>

2.6 Optimizations

2.6.1 Stop-lists

A well-known optimization in classifications based on lexical analysis is the definition of a stop-list — a list of common words that should be ignored. In order to generate the list automatically, a threshold distinguishing the most common words should be found. Numerous studies of documents show that 30% of general English text encompassing millions of words is made up of only 18 distinct words [15]. Usually, stop-lists contain about 250-300 terms [32, 25, 10]. However, our repository is limited to a focused scientific domain, so its language is rather limited, and may vary among topics. Thus, the stop-list should contain only those words which are
The tcms repository has shown that most of the stop-words at all the levels go to the stop-list. However, our observation of the parallel systems repository has shown that most of the stop-words at all the levels were the same, while for every lower level several additional common stop-words were added. The total number of stop-words is around 200 with slight differences for various competitive sets.

2.6.2 Special Treatment for Selected Fields

Another means for optimization is using domain-specific knowledge. In our case the domain is a bibliographical repository, which is classified into topics. Thus special fields like author names and topic titles may carry special significance.

For example, the topics and sub-topics title fields may be expected to reflect the contents of the topic, and this is based on a semantic understanding by a human editor. It is therefore desirable to use these words as keywords, even if the counter-based algorithm described above does not recognize them as such.

The special treatment of author names is founded on the assumption that usually scientists tend to concentrate their work in a rather narrow area of research. Therefore if several of the given author’s publications appear in one specific topic of the competitive set, but not in the others, then it is sensible to suggest that the new article will also belong to this topic. As most of the author names appear too rarely and thus do not survive the keyword filtering process, special treatment is required. Just as in the case of topic titles, we simply treat author names explicitly as keywords. For this purpose,
the first and last names are concatenated and treated as a single term.

2.6.3 Thesauri

The final major problem to be considered here is the use of similar or related terms (synonyms). Thus the use of thesauri in order to recognize variants or to control the vocabulary has been suggested [3]. A specific feature of our index is that it contains a lot of names of projects, systems, and tools, which are often referred to by acronyms. Text observations show that typically such terms occur in one of the following formats at least once [16]:

1. The full term words with capital letters and then the acronym consisting of the same first capital letters in parenthesis.
2. The acronym is followed by the parenthesized full term words interpretation.

Based on this we developed a thesaurus-builder which is responsible for lexical text analysis and extracting the full expressions and their acronyms, and used it to construct a dictionary of acronyms. This was used during parsing to check if the acronym or its interpretation occur in any particular concept vocabulary, and if so it was explicitly entered into the keywords vector. User queries are also checked against the thesaurus, and expanded in a similar manner.

3. ON-LINE SEARCHING

Given the keyword vectors for all the repository's topics, those matching queries can be found. This is done in two cases: when a user issues a search by specifying authors and/or keywords, and when a user inserts a new entry into the repository. In this latter case, the goal is to recommend topics to which the new entry may be linked.

An important goal is that a retrieved set will be of "reasonable" size — large enough to give the user a choice but not too large. BoW therefore doesn't retrieve a set of individual documents in response to a query. Instead, it returns whole concept pages. Moreover, if many of these concept pages belong to the same higher-level topic, that topic is returned rather than listing the lower level ones.

3.1 Matching and Ranking

Matching and ranking go together — we want to find the topics that match the query to the highest degree. Several methods for such ranking exist [21]. The most popular are based on the TFIDF algorithm described in section 2.4 [26, 30, 28, 27] and will be rejected here for the same reasons. An alternative approach which is usually used in clustering (e.g. in Isodata Clustering) is to compute the distances between the keyword vectors. This can be applied in our case, by comparing the distances between the query vector and the competitive set vectors. However, the query is typically so short that it is not reasonable to weight its terms [11] so the terms relative frequencies distance between the query and the index vectors is not useful in our model. Thus we have to use a boolean ranking method [17], rather than a vector space algorithm.

Our matching process works as follows:

1. Check the query data against the acronyms thesaurus, and insert both acronyms and their full interpretation into the initially empty query vocabulary vector \(QV_{oc}\).
2. Parse the query (or new entry) and insert the resulting 5-grams into the vocabulary vector \(QV_{oc}\) (with no terms weights considerations).
3. Starting from the highest level topics, measure the similarity of the query to all topics in the competitive set by counting the number of common words in the vectors:

\[
\text{score}_{topic} = |QV_{oc} \cap T \text{Key}_{topic}|
\]

4. Select the topics with the highest score, and continue recursively to lower levels. The selection criterion is that the score be higher than the average plus a standard deviation, as was done in section 2.5. This gives good results because in 84%-91% of the queries the biggest gap is between the highest and the next topic, or between the second highest and the third one (Fig. 5).

Note that we don't examine all the tree branches, but only those which survive the filtering criteria, thus reducing the computational cost. This technique, called tree pruning, was also employed by others [14, 20], except that they choose only the single most suitable sub-topic at each level. The main disadvantage of such aggressive "single-path" pruning is that a failure at one of the higher levels will cause all the classification process to fail, whereas pruning that keeps two or three branches for further examination attains almost the same accuracy as full tree evaluation. Therefore, our ranking scheme does not suffer from the irreversible errors occurrence problem. Choosing more than one also meets our expectation that an article may refer to several categories in the bibliography.

3.2 Output Representation

Observe that the total number of selected topics may grow exponentially while descending the tree, if most subtopics are selected at each stage. To avoid showing the user such a long list of hits, we replace them all by their shared father. As the result, the more general (higher level) topic will be returned to the user. The condition for such output compression is that at least 50% of the particular topic's children and more than two of them are in the resulting list. The compressing routine is performed recursively from the bottom to the root of the index. The results of output compression are demonstrated in Fig. 6. The output was compressed for about 25% of the queries, where the majority of the compressed output sets were those including 14 links and more, only 10% of them remained untouched. On the other hand, only 10% of smaller sets (up to 13 links) were compressed. The compression ratio is quite big, and the size of compressed output sets was decreased by half in average.

Given the topics selected by the ranking process, and remaining after output compression, the question is how to display them on the screen. The dilemma is how to reconcile two contradicting considerations: keep both the concept pages' topological locations in the hierarchy (as in the Berkeley Cha-Cha Search Engine [4]), and their respective ranking with regard to this query (as is typically done in search engines, e.g. Northernlight [22]). Our solution is to display the original index tree, with the selected links opened and marked with different colors and font sizes according to their relevance to the query.

4. EVALUATION

In order to check the final algorithm performance we have conducted a sequence of 5 experiments employing 7-fold cross validation over a corpus of about 3,500 bibliographic entries. The corpus is focused on the domain of parallel systems, with an index that has an average depth of 5 and an average branching factor of 6. Every experiment was based on about 500 randomly chosen entries, which were extracted from the repository. The automatic off-line
Figure 5: Distribution of topics scores (in a competitive set of 6 topics) for 30 sample queries. The topics are sorted in a descending order for each one.

Figure 6: Results of the compression procedure for 500 queries from one of the 7-fold cross validation experiments (described below), sorted in descending order by the uncompressed output sets sizes.

indexing was performed on the remaining 3,000 entries, and the resulting keyword vectors used to re-insert the 500 entries that were extracted. The hit ratio for each case was computed by comparing the algorithm’s classification of these entries with their original manual classifications (Fig. 7). Manually checking those that were misclassified revealed that in many cases they were indeed ambiguous, and had very short annotations that only included very general terms.

Our experimental results have corroborated those of McCallum et al. that larger vocabulary sizes generally perform better. For larger branches of the index our algorithm selects more keywords, and the classification reached its highest accuracy (near 100%). For example, the “Operating Systems and Run-Time Support” topic, which is one of the biggest topics in the repository with over 7,700 distinct five-grains vocabulary, got 100% hit ratio, whereas “Algorithms and Applications” which is a smaller topic, containing about 3,700 keywords, attained only 92% hit ratio. Another evidence is the decrease in hits percentage for lower levels, due to the smaller number of entries and therefore the smaller number of keywords, as shown in Fig. 8.

Generally, the results indicate that the more information is available about each concept and each query, the better the matching
Authors as keywords and acronyms thesaurus
Use stoplists:
Titles treated as keywords
Normalized counters
Unnormalized counters

5. CONCLUSIONS

We have developed and presented the details of a data classification algorithm for effective concept-based storage and retrieval of scientific papers in multi-level hierarchical repositories. The three main features of the algorithm are its homogeneity, scale independence, and self-updateability. The algorithm is homogeneous in that it produces good results at all levels of the hierarchical index, and does not depend on the index depth. It is scale independent due to the normalization of the keyword vectors, resulting in fair judgments for various-sized concept pages. It updates the keyword vectors regularly, thus keeping them current and adjusting to changes in the repository contents. This is done at selected intervals, rather than on-line for each new entry, because every local change in an individual concept page causes changes in the entire topic's vocabulary, and so in the selection of keywords across the entire competitive set; moreover, this effect can propagate up the hierarchy.

Results of experimentation with the BoW prototype repository on parallel systems are very promising. At the top level, nearly 95% of the entries were classified correctly, and this dropped to just under 90% for the lowest levels. Remarkably, this was achieved with only the entry details (mainly title and authors), and very short annotations typically between one and three sentences long. There was no access to or use of full text. The entries that were misclassified were found to be ambiguous and had short or missing annotations.

In the future we hope to test our algorithm on additional repositories. Possible extensions include automatic construction of a full thesaurus for all the words and phrases in the given corpus. A bigger challenge is automatic index creation from scratch. Our suggestion is to use one of the hierarchical clustering methods [12] combined with the described automatic indexing algorithm.

Acknowledgements

This work was supported by the Ministry of Science, Culture & Sport under the program to develop scientific and technological infrastructure.

6. REFERENCES


This document is covered by a signed "Reproduction Release (Blanket) form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").