A Ranking Algorithm for the Detection of Composite Concepts Based on Multiple Taxonomies

Daniel Kimmig
Institute for Applied Computer Science
Karlsruhe Institute of Technology
Germany
daniel.kimmig@kit.edu

Steffen Scholz
Institute for Applied Computer Science
Karlsruhe Institute of Technology
Germany
steffen.scholz@kit.edu

Andreas Schmidt
Department of Computer Science and Business Information Systems
Karlsruhe University of Applied Sciences
Germany
andreas.schmidt@hs-karlsruhe.de

***Abstract*—A full-text search is typically not appropriate for concept mining. For that reason, we use taxonomies to describe the concepts we are looking for. A typical input for our search consists of two or more taxonomies, describing the concept we are looking for. In this paper, we present a similarity measure between the input taxonomies and the searched documents. The algorithm is based on the idea of word n-tuples, where each word in a result tuple comes from another taxonomy. Because of the vast number of available documents, our similarity function must be fast to allow a quick ranking of the retrieved documents. We also provide an optimized implementation for our algorithm, which allows a fast ranking of the searched documents.**

Keywords–ranking algorithm; taxonomy based search; similarity function; performance measure

I. INTRODUCTION

In previous work [1], [2] we developed a searching strategy for (composite) concepts in document sets. The strategy was based on the idea of formulating concepts as taxonomies. So for example to look for documents containing information about "energy", we can use the taxonomy in Figure 1. The search process is then performed by looking for every word or phrase (and also defined synonyms - not shown in the Figure) in the taxonomy tree in the documents. The result for such a search is then a quantified taxonomy tree for each document, containing the number of occurrences of the words, phrases and synonyms. Additionally, the counts are cumulated toward the root of the tree. Figure 2 shows such a quantified result tree. Below each word in the taxonomy you find the number of occurrences inside the document. For all non-leaf nodes (*renewable fuel*, *coal*, *oil*, *fossil fuel*, and *energy*) you find additionally the accumulated number of occurrences for this sub tree, i.e., for the sub tree *renewable fuel*: $3\,(solar\ energy) + 4\,(wind\ power) + 2(geothermal) + 5(renewable\,fuel) = 14$. A more intuitive representation form could represent the weight of a node by different font sizes or colors or by different line widths of the edges in the taxonomy. The ranking of different documents can than be performed by simply taking the weight at the root of a taxonomy tree and an additional normalization step (i.e., division by the number of words in a text).

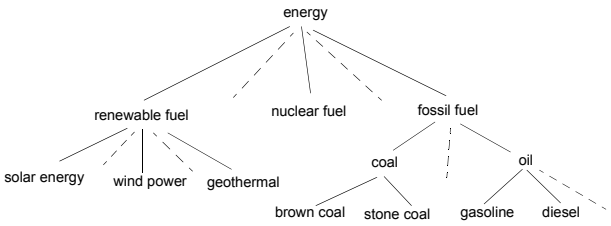

Figure 1. Energy taxonomy

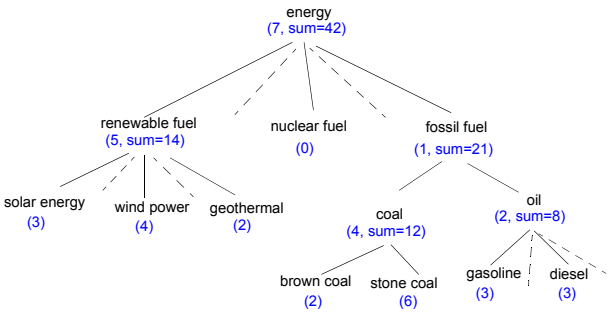

Figure 2. Quantified result taxonomy

Typically, we do not only search for one concept (like *energy*), but for a combination of concepts forming a more sophisticated concept like, i.e., *"used materials in the automotive industry"*. So instead of only looking for the concept of *material*, one is required to consider the application of different materials in *automotive manufacturing*. Relevant terms and phrases in the context of *vehicle manufacturing* (right side) and *material* (left side) are shown in Figure 3, representing a *isa* and a *is-part-of* taxonomy.

Looking for a single concept in a document is technically speaking a query which looks for the different words from the given taxonomy with an adjacent construction of the quantified result taxonomy, based on the words found in the document. In contrary, when we look for multiple concepts in a document, we have to search the documents for tuples, from which one word is from taxonomy A and the other word is from taxonomy B (in the case of two taxonomies).

 ISBN: 978-1-61208-311-7

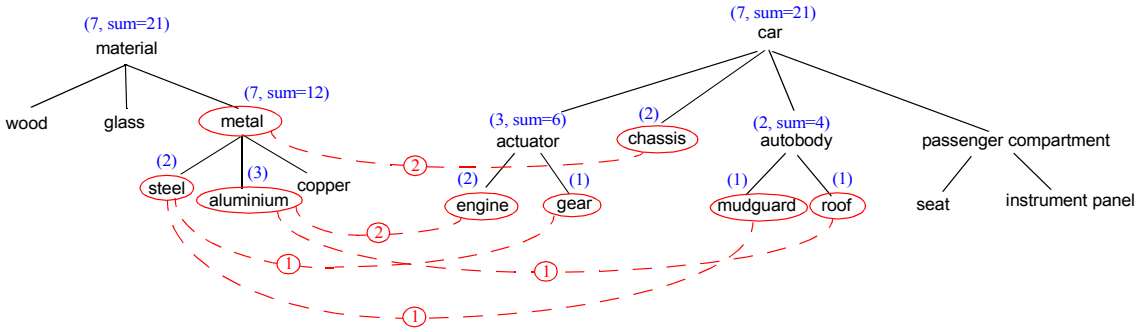

Figure 3. Relationship between taxonomies

In the case of three, four or more concepts to search, we have to find the corresponding n-tuples. This is illustrated in Figure 3. Here, the two taxonomies *car* and *material* are shown and the tuples which can be found in a concrete document are shown by connected ellipses (in red). The value along a connection link indicates how often a tuple combination was found in a document (i.e., the tuple *metal*, *chassis* appears two times in the document). As in the case of a single taxonomy these values are propagated toward the roots of the two trees (in blue). Mind, that in contrast to the case with one taxonomy, not the number of occurrences of the words/short phrases is counted, but only those words of the taxonomies which occur in a tuple. But this is only a simplified case, because it does not consider the distance of the words from the different taxonomies inside a found tuple. Consider the situation in Figure 4. In both situations the same number of tuples were found. In the first case (a), the average distance between words in the result tuple is much higher than in case (b). If the words from the different taxonomies appear near to each other, the probability is high, that the desired concept is described (i.e., an aluminum chassis). As a consequence, we have not only to consider the number of tuples found, but also the distance of the words in the found tuples for the ranking function.

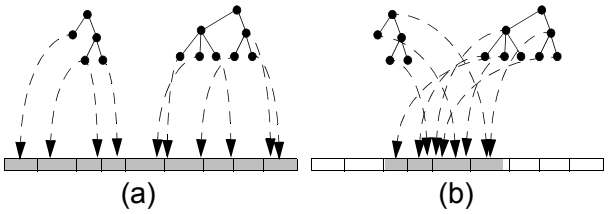

Figure 4. Interrelationships between concepts

While the implementation of a ranking function for the "one taxonomy" case is straightforward using an `OR` search in a conventional full text search engine like Lucene [3] or Sphinx [4] as basis and implementing the tree aggregation part on top, this is not possible for the multi-taxonomy case. The remainder of the paper is structured as follows: In Section II we review related work, which has already been done in this field. Afterwards, in Section III the concept of our fast ranking-algorithm is explained. Section IV shows the runtime behaviour of our algorithm, compared to the naive approach. We finish our paper in Section V with a scientific outlook for furher research.

II. Related Work

In the work of Cummins and O'Riordan [5], different proximity measures between pairwise terms were defined. Some of these measures are based on the distance between the occurences of the terms. Other measures take into account the term frequencies (tf) [6] of the related terms. In our work, we also use a proximity measure based on the distance of the involved terms, but in contrast to the previously mentioned work, we have to consider multiple taxonomies instead of multiple terms. Tao and Zhai [7] also mention the distance of the search terms in a document as an important factor for proximity measure. The authors add different metrics as complementary scoring components to different existing retrieval models to slightly adjust the final ranking. The results show that adding metrics, based on the distance of the terms, improve the overall retrieval accuracy, compared to the more coarse span-based measures. Again, this research focuses on single terms instead of taxonomies as in our work.

III. Algorithm

The main goal of our approach is to identify tuples of words and or phrases, which originate from different taxonomies and rank these based on the distance of each involved word/phrase. The following Figure 5 illustrates our concept. In the example, two taxonomies as well as an exemplary text serve as input to the ranking algorithm. Words from the taxonomies which appear within the exemplary text are highlighted in the respective colors (blue for T_1, red for T_2). Below the exemplary text, an array-like data structure is shown, which keeps track of the position of the word in the text as well as the origin of the taxonomy. Our ranking algorithm will iterate over this list of hits to identify

 ISBN: 978-1-61208-311-7

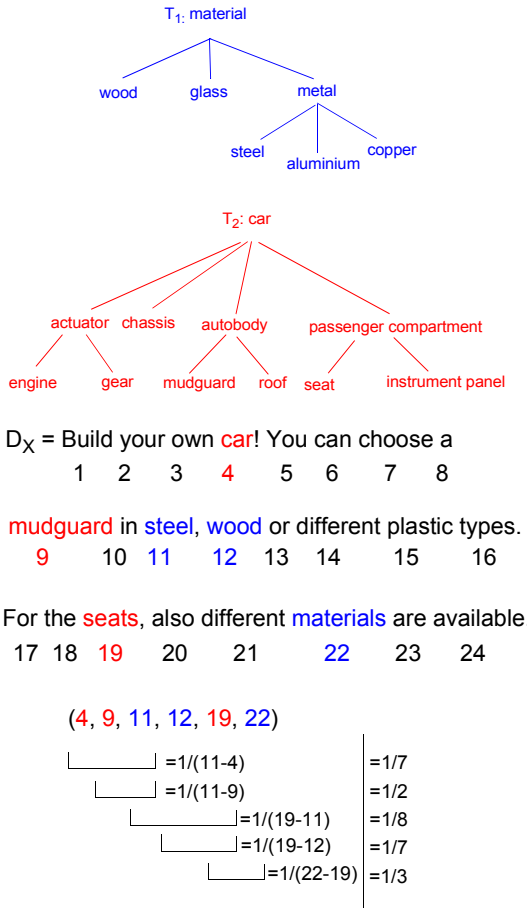

Figure 5. Ranking example using tuples from two taxonomies

tuples. A tuple is complete if words from all participating taxonomies are found. The first tuple is (car/steel) at position four and eleven. The distance value of a tuple plays a major role in the ranking formula. It is defined as the difference between the highest and lowest position value of the words of the tuple. In our case the distance value of the tuple (car/steel) is seven. This distance value embraces the fact that we are looking for cases in which words from all taxonomies appear very close to each other, as this indicates that the text is actually about a composite concept (see Figure 4). After all tuples have been identified as well as their distance is available, the ranking formula is calculated, which is shown in the following equation:

$$rank(T_1, T_2, D_x) = (\tfrac{1}{7} + \tfrac{1}{2} + \tfrac{1}{8} + \tfrac{1}{7} + \tfrac{1}{3}) * \tfrac{1}{24} = 0.052$$

The resulting ranking value is the sum of the inverse of each distance value divided by the length of the text. The inverse of the distance value is chosen to reduce the impact of higher distance values. Lower distance values indicate a high probability of the occurrence of a composite concept, which means the ranking should increase.

IV. Runtime Behavior

A naive approach to determine the ranking value is to iterate over the hit list while trying to find a closed tuple for each element. This makes it necessary to find complementary words from all other taxonomies for every element of the hit list, which requires three nested loops. This can become very costly depending on the size of the hit list as well as the amount of specified taxonomies as they determine when a closed tuple is achieved.

One potential optimization is to avoid the nested loops in some cases. This can be achieved by maintaining a dictionary of subsequent words from other taxonomies while calculating the ranking value across the entire hit list. This dictionary is used as a lookup table to complete tuples without searching for subsequent words for each element of the list. This dictionary is filled initially whenever the search for a closed tuple starts. The dictionary hereby serves as a memory of previous iterations in order to reduce the workload of subsequent stages of the processing.

Another idea is to stop the calculation once it is first encountered that no more tuple can be closed. Depending on the frequency distribution of words from each taxonomy, this can also reduce the required processing compared to the naive approach. However, this is very dependent on the dataset and might not be triggered at all in some cases, e. g., when a word from an infrequent taxonomy appears at the end of the hit list. We also tried to introduce a threshold value to limit the search space, in which tuples can be found. However, this technique actually decreased the performance of our algorithm, as words from infrequent taxonomies were not physically located within the area between the current index and the threshold value. This led to a lot of missing values in our lookup dictionary, which meant that the algorithm started to behave similar to the naive approach. We therefore decided to remove the search threshold and only keep the optimization approaches described before.

In the following, we will introduce performance metrics to illustrate the runtime behavior of our algorithm depending on the size of the hit list in Figure 6 as well as the amount of taxonomies under consideration in Figure 7. To run the benchmarks, we used a machine with a single 2,66 GHz Quad-Core Intel Xeon and 24 GB of main memory. The implementation is done in Java based on the 1.7.0u25 Oracle JDK in "-server" mode. We separate the benchmark in a warm-up and run phase of each 10.000 runs to reduce the impact of startup and JIT compilation effects. The Figures 6, 7 display the average runtime in msec. The first benchmark, is about the size of the hit list. A hit list is a data structure very similar to the example given in Figure 5. It is an ascending list of hits, which store the position of the word and the taxonomy it belongs to.

 ISBN: 978-1-61208-311-7

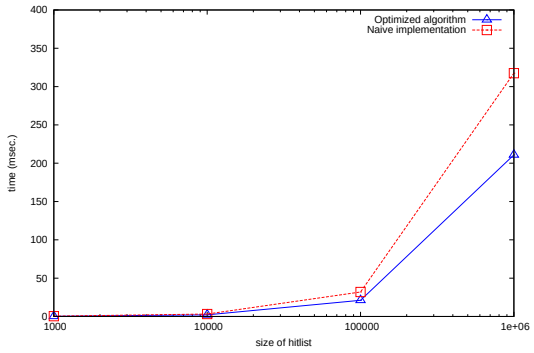

Figure 6. Runtime behavior depending on hit-list size

In the first benchmark, we used three taxonomies and manually generated a synthetic hit list. The frequency of words from the three different taxonomies is distributed by 70%, 25% and 5%. This is based on our observation from previous work [1], [2] in which we learned that hits are not distributed evenly among taxonomies, but rather follow Zipf's law. Based on this test setup, we generated hit lists of increasing sizes and calculated the ranking value using the naive and optimized algorithms. It can be concluded, that as the hit list grows in size, our optimizations have a bigger effect.

The following benchmark compares the algorithms based on various amounts of taxonomies, which are required to find a closed tuple. The frequencies are distributed in a similar fashion as the previous benchmark, which means that one taxonomy dominates the hit list, while others are rare in order to make the synthetic hit lists comply with our experiences from previous work. While the amount of taxonomies vary, each hit list has a size of 10.000 items.

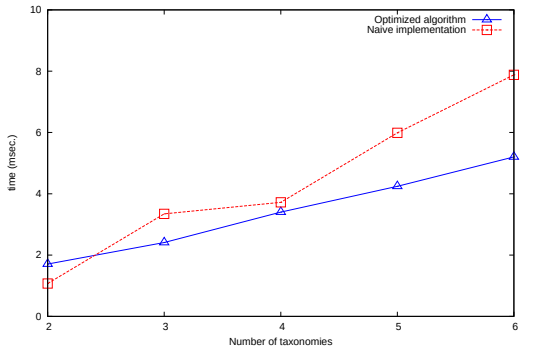

Figure 7. Runtime behavior depending on considered taxonomies

In the case of a small amount of taxonomies, the cost of maintaining the dictionary outweighs its benefits. However, once more taxonomies are considered, the optimizations become more effective. Although the gains do not meet our expectations, overall it is clear that our optimizations improve the required processing time in both test scenarios.

V. Conclusion and Outlook

We presented an algorithm for the ranking of documents based on taxonomy based queries. The algorithm calculates a similarity measure between the used taxonomies and a document which than can be used to rank the searched documents according to the used taxonomies. This measure is based on the occurrences of tuples containing words or phrases from all the different taxonomies and also the distance of the words found in the text.

Actually, we consider all the words with the same relevance. A more elaborate similarity function can give every word a different weight, so for example based on the term frequency and the inverse document frequency (weight: $w_{term,doc} = tf_{term,doc} * idf_{term}$) [6].

In our current implementation, the weight of every tuple is defined by the inverse of the distance of the words found in a tuple. Some (but not all) search results suggest that this measure probably favors tuples with words occurring consecutive inside a document too much. Optionally a measure which decreases the weight only logarithmically based on the distance could be more appropriate (i.e., $1/log(pos_{max} - pos_{min})$). But, to clarify this point, we need more input from our domain experts, when evaluating our ranked results. Another point for the future is to integrate our taxonomic based search inside the Lucene code base.

References

[1] A. Schmidt, D. Kimmig, and M. Dickerhof, "Search and graphical visualization of concepts in document collections using taxonomies," 46th Hawaii International Conference on System Sciences, 2013, pp. 1429–1434.

[2] A. Schmidt, D. Kimmig, and R. Senger, Poster: "Extraction and visualisation of semantic concepts from document-sets using taxonomies," First International Conference on Data Analytics (DATA ANALYTICS), 2012.

[3] E. Hatcher and O. Gospodnetic, Lucene in Action (In Action series). Greenwich, CT, USA: Manning Publications Co., 2004.

[4] A. Aksyonoff, Introduction to Search with Sphinx: From installation to relevance tuning. O'Reilly Media, 2011.

[5] R. Cummins and C. O'Riordan, "Learning in a pairwise term-term proximity framework for information retrieval," in Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, 2009, pp. 251–258.

[6] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, 1972, pp. 11–21.

[7] T. Tao and C. Zhai, "An exploration of proximity measures in information retrieval," in Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, 2007, pp. 295–302.

 ISBN: 978-1-61208-311-7