PROCESSING OVERLAPS

Georgina Ramírez
Yahoo! Research Barcelona
http://research.yahoo.com

SYNONYMS

Removing Overlap; Controlling Overlap

DEFINITION

In semi-structured text retrieval, processing overlap techniques are used to reduce the amount of overlapping (thus redundant) information returned to the user. The existence of redundant information in result lists is caused by the nested structure of semi-structured documents, where the same text fragment may appear in several of the marked up elements (see Figure 1). In consequence, when retrieval systems perform a focused search on this type of documents and use the marked up elements as retrieval objects, very often result lists contain overlapping elements. In retrieval applications where it is assumed that the user does not want to see the same information twice, it may be needed to reduce or completely remove this overlap and return a ranked list of no overlapping elements. Thus, depending on the underlying user model and retrieval application, different processing overlap techniques are used in order to decide, given a set of relevant but overlapping elements, what are the most appropriate elements to return to the user.

HISTORICAL BACKGROUND

Although the problem of overlap in semi-structured text retrieval is as old as the semi-structured documents themselves, no much work has been published on processing techniques for reducing or removing overlap. Some related work can be found in the area of passage retrieval, where approaches that use a varying window size for passage selection might produce result lists with overlapping passages. However, most of this work is performed on unstructured documents and the approaches taken for processing overlap tend to be simpler.

In the domain of semi-structured documents, there are several areas where different overlap issues are studied. For example, there is quite some work in the area of evaluation of XML systems that addresses the so called overlap problem (e.g., [1]). A different overlap problem is created by the possibility that standards like SGML provide of having multiple annotations (markups) on the same document (a.k.a. multiple hierarchies). In this case the overlap is produced by the structure of the different annotations. Since the multiple hierarchies complicate the use of standard retrieval techniques on this type of documents, work on this area is still focusing on addressing other indexing and retrieval issues. It is only recently that in the domain of XML documents several approaches have been presented that address the problem of processing overlap from result lists containing overlapping elements. The next section summarizes some of them.

SCIENTIFIC FUNDAMENTALS

One of the advantages of semi-structured documents is that retrieval systems can perform focused search by simply using the marked up divisions of the documents (elements) and retrieving those instead of the whole documents. However, since elements overlap each other (see Figure 1), when using traditional ranking techniques to independently rank these elements, result lists often contain many overlapping elements. This is due to the nested structure of semi-structured documents, where the same text fragment is usually contained in several of the

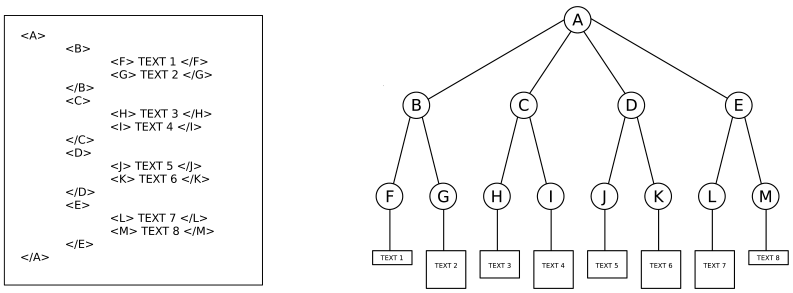


Figure 1: Example of a semi-structured document and its tree structure representation. Note that each fragment of the document is contained in three different elements (nodes in the tree).

marked up elements. Thus, when a specific element is estimated relevant to the query, all the elements containing this element (a.k.a. ancestors) will also be estimated to some degree relevant to the query. Furthermore, this element is probably estimated relevant because it contains several relevant elements (a.k.a. descendants). For example, if a section of a document is estimated highly relevant, it most probably contains several highly relevant paragraphs and it is contained in a relevant article. If all of these elements are returned to the user, the amount of redundant information contained in the result list will be considerable. In retrieval scenarios where users do not like to see the same information twice, retrieval systems need to decide which of these relevant but overlapping elements is the most appropriate piece of information to return to the user. The final decision on which elements the system should return depends on the search application and the underlying user model but a common goal is to reduce redundancy in the result lists. Although this can be done at indexing time (e.g., by selecting a subset of non-overlapping elements as potential retrievable objects), commonly this is done by removing overlapping elements from the result set, after retrieval systems have produced an initial ranking of all elements. Processing overlap techniques have recently been widely discussed in the domain of XML retrieval, where different approaches have been presented. The rest of this section presents and discusses some of them.

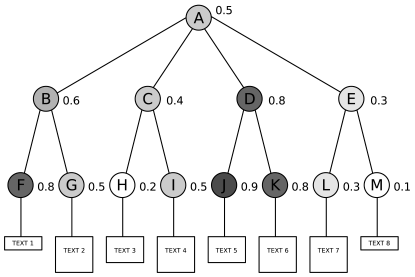
1 Using element types

A simple way to reduce overlap is to select a subset of element types and consider only these for retrieval. For example, if sections of documents are considered to be the most appropriate pieces of information, retrieval systems might want to use only these as retrievable objects and ignore all the other element types. This can be done at indexing time or by post-filtering the result lists. Depending on the number and element types selected, overlap is reduced at different degrees. Note that it might not always be possible to completely remove overlap; even if a single element type is selected as a unique retrievable object, it is still possible that elements of the same type overlap each other. The main drawback of this approach is that, since it is desirable to select element types that are likely to be relevant and useful to the user, it requires knowledge of the structure of the documents and its common usage.

2 Using paths

A common approach for processing overlaps keeps the highest ranked element on each path and removes its ancestors and descendants from the result list, i.e., all the elements in the result list that contain or are contained within it (e.g. [3], [4]). It is important to notice that depending on the order in which the different paths are processed different outputs might be produced.

In [3] the authors present a two steps algorithm to remove the overlap. The first step is used to select the highest scored element from each relevant path. Since their algorithm selects the elements from the different paths simultaneously, the output may still contain overlapping elements. That is why a second step is needed, to completely remove overlap. This is done by selecting again (this time from the output of the first step) the highest scored element from each path:



Original output: <J, D, F, K, B, A, G, I, C, E, L, H, M>

Algorithm 1:
output step 1: <J, F, D, B, A>
final output: <J, F>

Algorithm 2:
final output: <J, F, K, G, I, E, H>

Figure 2: Example of a retrieval run and the resulting outputs when removing overlapping elements with algorithm 1 and 2. The numbers on the tree indicate the initial retrieval scores obtained by each of the elements.

Algorithm 1
1.- Select highest scored element from each relevant path.
2.- Select highest scored element from each relevant path in output of step 1.

Another common way to remove overlap using paths [5] is to recursively process the result list by selecting the highest ranked element and removing any element from lower ranks that belongs to the same path (it contains the selected element or it is contained within it):

Algorithm 2
1.- Return highest ranked element from result list.
2.- Remove from result list all the elements belonging to the same path.
3.- Repeat step 1 and 2 until result list is empty.

The underlying assumption of this type of approaches is that the most appropriate piece of information in each path has been assigned a higher score than the rest and therefore, removing overlap is simply a presentation issue. These approaches rely completely on the underlying retrieval models to produce the best ranking. This could indeed be the case if the retrieval model would consider, when ranking, not only the estimated relevance of the element itself but also its *appropriateness* compared to other elements in the same path. However, since many retrieval models rank elements independently, the highest scored element may not be the most appropriate one, i.e., the one the user prefers to see.
To illustrate the different outputs of the previous algorithms, have another look at the example document from Figure 1. Imagine now that, given a query, the retrieval model estimates the relevance of each element in the document. Figure 2 shows the retrieval scores obtained by each of the elements and the outputs produced when removing overlap with the algorithms described above. Both algorithms produce a result list of non-overlapping elements. However, there are substantial differences. The main drawback of the first algorithm is that it might miss some relevant information. For example, one could argue that element K should also be contained in the output list. To be able to do that the algorithm should consider structural relationships between elements and re-add element K to the result list when it decides to remove element D in the second step.
Although the second algorithm produces a more complete list, someone could argue that the output produced is not the most desirable. For example, imagine that elements F and G are, respectively, the title and the abstract of the document. In this case, even if the title has been ranked high (it may contain most or all of the query terms), users might prefer to see a result item containing both, title and abstract (i.e., element B) instead of seeing both elements independently. In general, it can be argued that retrieval systems should only return those elements that have enough content information to be useful and can stand alone as independent objects. A similar example can be seen for elements D, J and K. Even if J is ranked higher, element D contains mostly relevant information and therefore, it might be more desirable (from a user perspective) to read element D than J and K independently. Furthermore, for elements E, L, and M it could be argued that it is better to return L than E because the relevance estimated for element E is due to the content of L and no extra benefit is obtained when returning E.

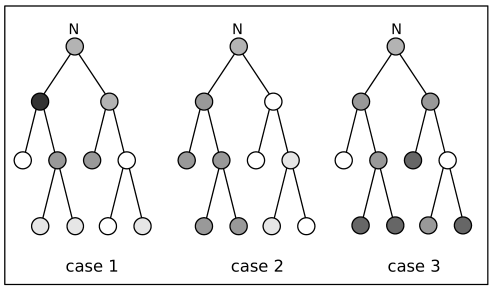


Figure 3: Illustration of the three cases considered in the approach presented in [6]. The grayer the node, the higher the relevancy estimated for that node.

In all these cases a better output might be produced if the algorithms would consider structural relationships between elements when deciding which elements to return to the user.

3 Using structural relationships for re-ranking

The following approaches present more advanced techniques that exploit the structural relationships within a document to decide which elements should be removed or pushed down the ranked list and which ones should be returned to the user (e.g., [2], [6], [5]). These techniques often modify the initial ranking predicted by the retrieval model.

In [2] elements are re-ranked by adjusting the element scores of the lower ranked elements according to their containment relationship with other higher ranked elements. The assumption underlying this approach is that the score of the elements that are contained or contain other elements that have been already shown to the user (i.e., are ranked higher) should be adjusted in order to reflect that the information they contain might be redundant. They do that by reducing the importance of terms occurring in already reported elements. The basic algorithm is similar to algorithm 2 but instead of removing the ancestors and descendants of the reported elements, their scores get adjusted:

Algorithm 3
1.- Report the highest ranked element.
2.- Adjust the scores of the unreported elements.
3.- Repeat steps 1 and 2 until m elements are reported.

The author also presents an extended version of the algorithm where different weighting values are used for ancestors and descendants and where the number of times an element is contained within others is considered. For example, a paragraph contained in an already seen section and in an already seen article is further punished because the user has already seen this information twice. Note that this algorithm is not designed to remove the overlap but to push down the result list those elements that contain redundant information.

In [6] a completely different approach is taken. The authors present a two step re-ranking algorithm for removing overlap. The first step identifies clusters of highly ranked results and picks the most relevant element from each cluster. The second round is used to remove any remaining overlap between the selected elements. The selection criteria for the first step is based on three different cases (illustrated in Figure 3): (1) if an element N has a descendant that is substantially more relevant, the element N is removed from the result list; (2) if case 1 does not hold and the element N has a child that contains most of the relevant information (the relevant elements are concentrated under this child), the element N is also removed from the result list, (3) if none of the previous cases hold and the results are evenly distributed under the element N, then the element N is kept and all its descendants removed from the result list. In the rest of the cases they do not do anything and leave the final overlap removal for the second phase. In the second step, they remove overlap by comparing the score of each element with the ones of its descendants. If the score of the element is bigger, all the descendants are removed and the element is kept. Otherwise, the element is removed.

In [5] the authors present an approach that makes use of an *utility* function that captures the amount of *useful* information contained in each element. They argue that to model the *usefulness* of a node three important aspects need to be considered: (1) the relevance score estimated by the retrieval model, (2) the size of the element, and (3) the amount of irrelevant information the element contains. They present an algorithm that selects elements according to the estimated *usefulness* of each element. If an element has an estimated *usefulness* value higher than the sum of the *usefulness* values of its children, then the element is selected and the children are removed. Otherwise, the children elements whose *usefulness* value exceeds some threshold are selected and the element is removed.

KEY APPLICATIONS

Processing overlap techniques are needed in any retrieval application where users need to be directed to specific parts of documents and there are not predefined retrieval units.

EXPERIMENTAL RESULTS

For every presented approach, there is an accompanying experimental evaluation in the corresponding reference. Commonly, the referred article provides an overview of the performance of the approach and of its variations. However, it is difficult to use the reported evaluations to compare approaches between articles. The main reason is that all approaches make use of different retrieval models for their initial runs, thus it is not clear whether the performance obtained after removing overlap is due to the underlying retrieval model or to the approach used to remove the overlap. Besides, not all the presented approaches experiment on the same dataset or use the same evaluation measures to report their numbers.

In [5] the authors present an experimental comparison between several of the approaches described above. They show that the best performing approach is the one that returns only paragraphs. As a general trend for the first type of approach (the ones that select a specific element type), the longer the element type selected, the worse the performance. The authors also show that their approach of estimating the *usefulness* of an element can help to improve retrieval performance (in terms of precision at low recall levels) when compared to the approach of using paths (algorithm 2).

DATA SETS

Since 2005 the *INitiative for the Evaluation of XML Retrieval* (INEX) provides a data-set that can be used to test processing overlap strategies (see http://inex.is.informatik.uni-duisburg.de/).

CROSS REFERENCE

XML Retrieval, Structured Document Retrieval, Evaluation Initiative for XML Retrieval (INEX).

RECOMMENDED READING

[1] G. Kazai, M. Lalmas and A.P. de Vries, *The overlap problem in content-oriented XML retrieval evaluation*, SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, 2004, isbn 1-58113-881-4, pages 72–79, ACM Press, New York, NY, USA.

[2] C.L.A. Clarke, *Controlling overlap in content-oriented XML retrieval*, SIGIR '05: Proceedings of the 28th annual international ATM SIGIR conference on Research and development in information retrieval, 2005, isbn 1-59593-034-5, pages 314–321, ACM Press, New York, NY, USA.

[3] K. Sauvagnat, L. Hlaoua and M. Boughanem, *XFIRM at INEX 2005: Ad-hoc and Relevance Feedback Tracks*, Advances in XML Information Retrieval and Evaluation. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX 2005), Lecture Notes in Computer Science, volume 3977, pages 88-103, issn 0302-9743 (Print) 1611-3349 (Online), Springer Berlin/Heidelberg, 2006.

[4] S. Geva, *GPX - Gardens Point XML IR at INEX 2005*, Advances in XML Information Retrieval and Evaluation. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX 2005), Lecture Notes in Computer Science, volume 3977, pages 240-253, issn 0302-9743 (Print) 1611-3349 (Online), Springer Berlin/Heidelberg, 2006.

[5] V. Mihajlović, G. Ramírez, T. Westerveld, D. Hiemstra, H. E. Blok and A. P. de Vries, *TIJAH Scratches INEX 2005: Vague Element Selection, Image Search, Overlap and Relevance Feedback*, Advances in XML Information Retrieval and Evaluation. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX 2005), Lecture Notes in Computer Science, volume 3977, pages 72-87, issn 0302-9743 (Print) 1611-3349 (Online), Springer Berlin/Heidelberg, 2006.

[6] Y. Mass and M. Mandelbrod, *Using the INEX Environment as a Test Bed for Various User Models for XML Retrieval*, Advances in XML Information Retrieval and Evaluation. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX 2005), Lecture Notes in Computer Science, volume 3977, pages 187-195, issn 0302-9743 (Print) 1611-3349 (Online), Springer Berlin/Heidelberg, 2006.