Improving the Accessibility of SGML Documents – A Content-analytical Approach*

Helena Ahonen Barbara Heikkinen
Oskari Heinonen Mika Klemettinen

{hahonen, bheikkin, oheinone, mklemett}@cs.helsinki.fi

Department of Computer Science, University of Helsinki
P.O. Box 26, FIN–00014 University of Helsinki, Finland

Abstract

Document retrieval based on string searches typically returns either the whole document or just the occurrences of the searched words. What the user often is after, however, is microdocument: a part of the document that contains the occurrences and is reasonably self-contained.

These microdocuments might, for instance, consist of several successive text paragraphs sharing a mutual subject. Single paragraphs, or corresponding close-to-leaf SGML elements, do not convey enough of the contextual information. On the other hand, sections or subsections of a text document, such as a book or an article, can discuss many heterogeneous topics, and thus be too large a unit for retrieval or assembly.

We claim that such microdocuments are both suitable retrievable units and appropriate units for document assembly, and that they can be reasonably well located using automatic techniques.

Optimal creation of microdocuments would require thorough semantic analysis of the text. However, it is possible to catch parts of the elementary semantic content by statistical term-frequency analysis.

Term-frequency distributions enable us to determine the locations of possible topic changes in the text. Based on this information, we can measure the similarity of two successive elements, and decide whether we wish to have them in the same microdocument. On the other hand, existing markup, for example classifying attributes, can be used in boundary detection. The microdocument, again, can be attributed with content information.

The results of our preliminary experiments show that the presented approach works well in user-assisted topic-oriented microdocument detection. We currently study the usefulness of this technique in document assembly, i.e., in generating new documents from a collection of existing text documents.

*To appear in *SGML Europe '97*, Barcelona, Spain, May 1997.

1 Introduction

In the early years of computerization, library catalogue files included only records stating, e.g., the title of the book and the name of the author. Searching with the title or the author name was fairly simple and effective. Nowadays, electronic full-text documents and digital libraries make the utilization of texts much more effective than before; yet, they pose new problems and requirements. For example, document retrieval based on string searches typically returns either the whole document or just the occurrences of the searched words. What the user often is after, however, is microdocument: a part of the document that contains the occurrences and is reasonably self-contained.

We claim that multi-paragraph microdocuments are both suitable retrievable units and appropriate units for document assembly, and that they can be reasonably well located using automatic techniques. With document assembly we consider computer-aided construction of a new document using several existing document sources. Assembly is usually an interactive process within which an author or editor uses various tools to find appropriate sources and to configure the intended document. The assembly process consists of two parts: (1) finding the interesting document fragments and (2) constructing a coherent document from these fragments. Meaningful and reasonably sized fragments, i.e. microdocuments, are essential in the assembly process.

In this article, we assume that the data consist of text documents having a typical section/subsection/paragraph structure, or a structure that can be mapped to a corresponding generalized structure [AHHK97]. The Standard Generalized Markup Language (SGML) is a natural way of representing the structure.

This paper is organized as follows. Section 2 introduces the concept of microdocument as we see it. Then in Section 3, we describe the process of detecting topical microdocuments. The detection is based on counting similarity measures for successive text paragraphs (Section 3.1), and is complemented by a fragmentation process which selects appropriate microdocument boundaries (Section 3.2). The utilization of microdocuments in document assembly is outlined in Section 4. Finally, Section 5 is a short conclusion.

2 Topical Microdocuments

We define microdocument as a reasonably self-contained fragment of the document. We do not see microdocuments as separately saved units only, but as semantically motivated units that can be represented internally as well as externally. Thus, creation of microdocuments can also be seen as additional markup that the document is seeded with.

We consider a topical microdocument to be semantically motivated by the topic the microdocument discusses. Topical microdocuments might, for instance, consist of several successive text paragraphs. Single paragraphs, or corresponding close-to-leaf SGML elements, do not convey enough of the contextual information. On the other hand, sections or subsections of a text document, such as a book or an article, can discuss many heterogeneous topics. Furthermore, sections are often longer than desired with respect to the intended purpose. It would not make sense, anyway, to break against the existing section/subsection structure of the document by, for example, joining paragraphs over section boundaries.

Text fragmentation is also studied in [Hea94], which introduces a method called TextTiling. Its approach, however, differs from the one that we have adopted. Passage retrieval, as part of information retrieval, attempts to improve precision and recall of query results by dividing documents into passages, in order to include some context around the hit words. The passages do not necessarily represent any semantically meaningful units, for instance in [Cal94], the size of passages is fixed. Moreover, text fragmentation is used for decomposition of continuous texts into hypertext [SSBM96]. In this application also the similarities of non-consecutive pieces of text are important, since hypertext links can clearly implement non-linear structures.

3 Detecting Microdocuments

Detecting microdocuments means basically breaking the document in smaller parts. If the document has a section/subsection-type structure, it is first divided into these parts. Probably many of the resulting parts are rather lengthy and require further processing. Naturally, this implies a need for paragraph-level handling. The paragraph-level elements are recognized using a predefined set of element mapping rules; the mapping rules can be either user-given or automatically generated (see Section 4). Paragraphs, again, can be very short; so, a microdocument can be created by combining short paragraphs that are related by content.

Optimal creation of microdocuments would require thorough semantic analysis of the text. However, it is possible to catch parts of the elementary semantic content by statistical term-frequency analysis. Term-frequency distributions enable us to determine the locations of possible topic changes in the text. Based on term frequencies, we can measure the similarity of two successive paragraph-level elements, and decide whether we wish to have them in the same microdocument. Yet, we never, as already determined by the above-mentioned section/subsection-level breaking, consider joining paragraphs of successive sections or subsections.

Since the similarity calculation and the fragmentation procedure are the kernel of our microdocument-detection process, let us now describe these methods.

3.1 Similarity Between Paragraphs

In order to be able to calculate the similarities between successive paragraphs, we must first preprocess the text and construct a term-frequency vector for each paragraph. That is, we employ stemming to get the stems of the words (optional), remove stopwords (typical non-content-bearing words) and count the frequencies of the remaining words, i.e. terms, in each paragraph. Then, we take a predefined number, e.g. 50, of the most frequent terms to represent the paragraph. We denote a term-frequency vector x as follows:

$$x = ((t_{x1}, f_{x1}), (t_{x2}, f_{x2}), \ldots, (t_{xl_x}, f_{xl_x})),$$

where $t_x i$ is a term, $f_x i$ the frequency of the term, and l_x is the length of the vector.

We have considered two methods for similarity calculation. A trivial approach looks at only two successive paragraphs and calculates the similarity using the well-known cosine coefficient formula (see, e.g. [Sal89]. The cosine coefficient, in turn, uses the paragraph term-frequency vectors. Intuitively, if the vectors contain exactly the same terms in the same proportions, their similarity value is 1.0; on the other extreme, the similarity of mutually exclusive term-frequency vectors is 0.0. The cosine coefficient of two vectors x and y, and thus the similarity of the corresponding paragraphs, is calculated with the following formula.

$$\mathrm{sim}(x, y) = \frac{\sum_{i\in[1,l_x];\, j\in[1,l_y];\, t_{xi}=t_{yj}} (f_{xi} f_{yj})}{\sqrt{(\sum_{i\in[1,l_x]}(f_{xi}{}^2))(\sum_{j\in[1,l_y]}(f_{yj}{}^2))}}.$$

A more elaborate method of calculating the similarity is to take a wider neighbourhood into account; that is, instead of one paragraph, e.g., three or four paragraphs on both sides of each paragraph boundary are considered. The paragraph term-vectors are weighted based on their relative distance from the paragraph boundary with the immediate paragraphs having the highest weight. For example, we might have a weighting function, which gives weight 1 for the nearest two paragraphs, for the next paragraphs on both sides weights 0.8, 0.64, 0.512, etc.

In the algorithm for window-based similarity calculation, we, as before, calculate a similarity value for all paragraph boundaries. If, e.g., a window size of 3 paragraphs is given, and the weights for the three nearest paragraphs on both sides of the boundary are 1, 0.8, and 0.64, we first multiply the term-frequencies in each of the vectors by their respective weights. Then, we termwise sum up the resulting three vectors on both sides of the currently considered boundary. These vectors are then used like simple paragraph term-frequency vectors to calculate the similarity using the cosine coefficient formula.

What we benefit from using a window is that we can smooth the effect of short paragraphs which are located between similar paragraphs. For example: we have four paragraphs that share some mutual subject, and they are separated by a short clarifying example that contains mostly numbers, etc. The similarity between the second paragraph and the example, as well as between the example and the third paragraph, would be close to zero; thus, they would not be similar at all. When using a window size of, say, 3 paragraphs, the four paragraphs and the example could still be clustered in the same fragment.

The results from our experiments with real data show that the window method is applicable in practice and gives reasonable results. In our implementation, we have used three functions to compare the effect of different weightings: a cosine-based function, a linear function, and an exponent function. A user can select a suitable function for the intended purpose. For example, the cosine-based function, compared with the linear function, favours even more those paragraphs that are close to the currently considered boundary.

Term-frequency based calculation of similarities is rather straightforward with languages like English. Even stemming is not a necessity, because words not that often take inflected forms, and there are not many different-looking inflected forms for a base form. The situation is more difficult in morphologically rich languages. For example, in the Finnish language, getting the base forms of the words is important since words most often appear inflected.

3.2 Fragmentation

After counting the similarities between successive paragraphs, we can discover topic changes by inter-paragraph similarities. Basically, every paragraph boundary is a potential microdocument boundary. To be able to get reasonably-sized microdocuments, the similarity information alone is usually not enough; also the lengths of the created fragments have an impact on the usefulness of the result. In this section, we briefly describe two alternative approaches for fragmenting the text into shorter units, namely similarity-based clustering and dynamic programming.

To begin with, simple similarity-based clustering goes roughly as follows. We first take the two adjacent paragraphs which are most similar, and cluster them together into the same fragment. Then, we take the following two most similar paragraphs or intermediate fragments and join them, until we have a desired amount of fragments. We can enhance this solution by taking the paragraph lengths into account while choosing the paragraphs to be joined, e.g., by not joining too lengthy paragraphs but joining the next most similar paragraphs if their combined length is below a given limit. We must also know when to stop the clustering process, i.e., when we have a suitable amount of fragments.

Let us then concentrate on another solution. In order to utilize both the similarities and the lengths, we have employed what is called dynamic programming (see, e.g., [CLR94, Chapter 16]). Dynamic programming is used for getting an optimal splitting of the section in hand by recursively defining the value for optimal solution.

The main idea behind resolving the optimal solution with minimal cost calculation for a certain splitting is as follows (see also the algorithm in Figure 1). We take all the paragraph boundaries, which are both potential splitting points and resulting microdocument boundaries, and the similarity information related to these boundaries. Then we start from the first boundary and calculate the cost for it, as if the first paragraph would be a single fragment in the final splitting. Then, we take the second boundary and attach to it the minimal cost from the two available possibilities: the cost of the first two paragraphs as if they were a single fragment in the final splitting and the cost given by the alternative that the first two paragraphs would have been splitted to separate fragments. In the following steps, the considered splitting point is moved on by one paragraph and all the possible locations of the previous splitting point are evaluated. We continue this procedure till the end of the text, and finally we can generate a list of splitting points that indicate the optimal microdocument fragmentation.

To evaluate the goodness between different splittings, a cost function for the text length must be defined. This cost function gives the lowest cost for the user-given preferred fragment length, say, e.g., 500 words. A fragment which is either shorter or longer gets a higher cost, i.e., is punished for its length. We have used two exemplary cost functions, second degree function (parabola) and a linear V-shape function, but other types of functions, with similar behaviour, could be considered as well.

After getting the fragmentation information, it is possible, among other ways of modelling, either to add new tags into the existing document that indicate the fragments, e.g., `<fragment fragid="1">...</fragment>`, or to complement the existing paragraph-level markup with an attribute which states the fragment that the paragraph-level element belongs to, e.g., `<para fragid="1">...</para>`. We prefer the latter approach, because it fully retains the hierarchical structure of the document and the document type definition (DTD).

```
similarity[0] = 0.0; # similarities are known
cost_of_boundary[0] = 0.0;
for current_paragraph = 1 to no_of_paragraphs
{
    length_sum = 0; # sum of preceding paragraphs
    smallest_cost = MAXREAL;
    for i = current_paragraph to 1
    {
        current_boundary = current_paragraph - 1;
        # boundary numbering: ... par3 bound3 par4 ...
        length_sum += length_of[current_paragraph];
        breaking_cost = cost_of_text_length(length_sum) +
                        cost_of_boundary[current_boundary] +
                        similarity[current_boundary];
        if breaking_cost < smallest_cost
        {
            smallest_cost = breaking_cost;
            smallest_cost_boundary = current_boundary;
        }
    }
    cost_of_boundary[current_paragraph] = smallest_cost;
    link_to_previous_boundary[current_paragraph] =
                                  smallest_cost_boundary;
}
j = no_of_paragraphs;
while link_to_previous_boundary[j] > 0
{
    add link_to_previous_boundary[j] into
    the set of chosen microdocument boundaries;
    j = link_to_previous_boundary[j];
}
output the set of chosen microdocument boundaries;
```

Figure 1: The dynamic programming algorithm for microdocument boundary detection.

3.3 Example

As test data, among others, we used *Mars* by Percival Lowell, 1895. The original text consists of six chapters, which are, in turn, divided into 1–4 sections. Furthermore, each section contains basically only paragraphs or elements similar to them. As an illustrative example, we present the analysis of Section I. *Evidence of it* of Chapter II. *Atmosphere.* The length of the section is approximately 6600 words and it contains 55 paragraphs or paragraph-level elements.

While creating term-frequency vectors for each paragraph, we used stemming to get the stems of the words and a list of stopwords to filter out uninteresting words. For similarity calculation we employed window lengths between 1 and 6, and the cosine-based weighting function to favour the closest paragraphs. For fragmentation, the dynamic programming method with a second degree cost function was used. The value of the preferred fragment length was set to one tenth, i.e. 660 words, of the total length of the section.

Using window length of 3, we found eight fragments, or microdocuments, separated by vertical lines in Figure 2. Each curve denotes the similarities with the respective window length (e.g. N3 = window length 3, etc.). For clarity reasons, curves for window lengths 5 and 6 are omitted; their behaviour is very close to the behaviour caused by window length 4. The fragment boundaries using window length of 3 were set between paragraphs 10 and 11, 15 and 16, 23 and 24, 28 and 29, 34 and 35, 44 and 45, and 49 and 50. The lengths of the fragments were 979, 642, 807, 560, 1020, 961, 961, and 682 words, respectively. Some fragments have an easily identifiable topic, like atmospheric chemistry in fragment 7. Fragments 2 and 3 seem to have roughly the same topic: measuring the diameter of the planet Mars. This can be explained by the requirement on the preferred fragment length.

We analysed the original text and compared the attained results with the text, as well as the results with the other window lengths. In general, the usage of moderate window sizes led to fragmentations close to the above-given example with window length 3. As the length increased or decreased significantly, the more inaccurate or unintuitive the results seemed to evolve. Actually, what affected most to the attained fragmentation, was the parametrized cost function used for the text length.

4 Microdocuments in Document Assembly

The topical microdocuments which are the end product of our fragmentation procedure, serve as retrievable units for document assembly. With document assembly we consider computer-aided construction of a new document using several existing document sources. The assembly process consists of two parts: (1) finding the interesting document fragments and (2) constructing a coherent document from these fragments.

Although one of the strengths of SGML technology is to allow several representations and formats to be generated from the same documents, multiuse of documents by dynamic assembly of structure elements is still an unsolved problem, and hardly any techniques or tools exist. The necessity of assembly is, however, clearly seen and considered to be of great importance [McF96, Sal89].

In our approach, assembling documents from a collection of SGML documents has the following steps.

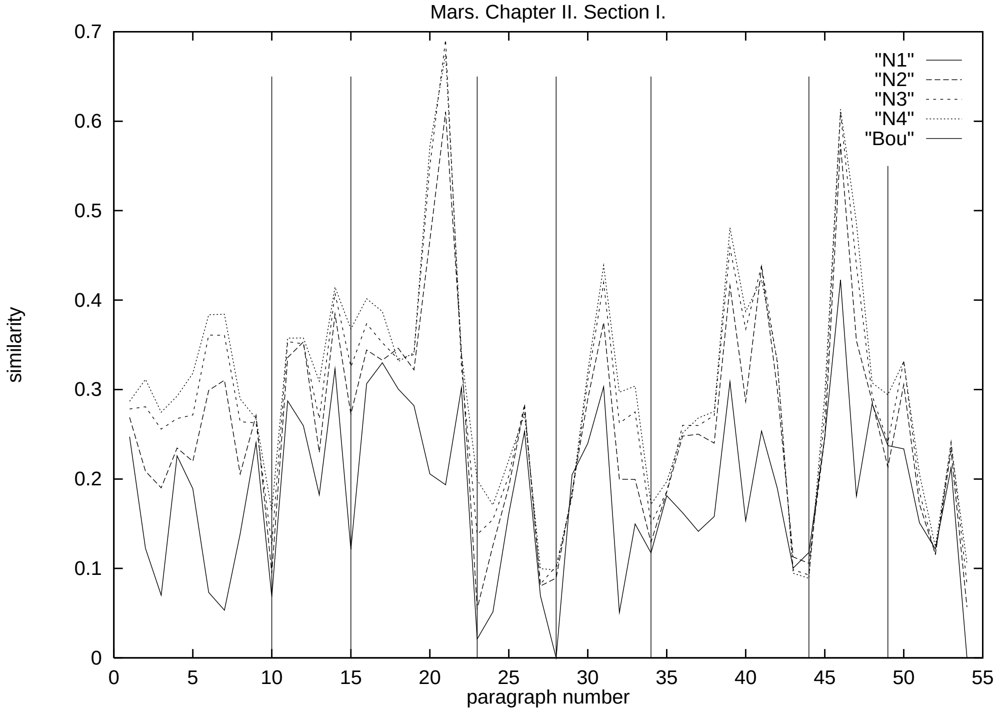

Figure 2: Similarity curves using different window lenghts and detected fragment boundaries.

1. The user selects interesting document fragments by querying and browsing.
2. A new DTD is formed.
3. A new document is constructed from the generalized elements.
4. The formatting rules are formed for the elements.

An obvious problem in document assembly is that the source documents may have differing structures, and hence, an arbitrary composition of their fragments does not belong to any known document class. Therefore, no formatting rules are available, i.e., the documents cannot be browsed or printed in a formatted form.

The fragmentation procedure classifies each element to some general element, mainly a section or a paragraph, to decide which elements form meaningful microdocuments. The same classification can be utilized also in the assembly, since the general elements have well-known semantics. We have defined a set of general elements that often appear in texts. We also give them element declarations and simple semantics, e.g., how the element should be printed.

Actually the classification needed in the assembly process is often more fine-grained than in fragmentation: at least sections, paragraph containers, paragraphs, and strings are usually found. Additionally, some elements, like tables and figures, do not match well semantically any of the general elements, although they can be interpreted as, e.g., paragraphs for the fragmentation purposes. These elements are in the assembly preserved as such, as well as the elements they contain. The elements can be identified using simple heuristics, e.g., in tables the proportion of tags compared to the text content itself is high, and the DTD usually gives enough hints for recognizing figures and formulas.

Construction of the new DTD is straightforward: it expresses essentially nothing more than the inclusion relationships of the general elements, e.g., a section is allowed to contain paragraphs but not vice versa. If some element structure is left unclassified, its original definition is added to the DTD. Additionally, the element has to be contained in the definition of the parent.

The actual assembly process composes a valid SGML document from the fragments. After the user's selection of interesting fragments, we have an ordered list of fragments containing classified elements. First, a new document root is constructed for the new document. Our intended top-level structure contains a list of sections, possibly preceded by a couple of paragraphs or paragraph containers. Hence some paragraphs and paragraph containers may need to be combined by generating new sections. Some heuristics can also be used to expand the content, e.g., some upper-level titles, metatext or referred figures can be included.

The formatting rules can be constructed in a straightforward manner. The general elements have their formatting rules that are independent of the documents assembled, whereas an unclassified element is output using the rules attached to the original document.

5 Conclusions

A microdocument can be defined as a reasonably self-contained fragment of the document. We do not see microdocuments only as separately saved units, but as semantically motivated units that can be represented internally as well as externally. Thus, creation of microdocuments can be seen as additional markup that the document is seeded with.

We consider a topical microdocument to be semantically motivated by the topic the microdocument discusses. Topical microdocuments might, for instance, consist of several successive text paragraphs. Single paragraphs, or corresponding close-to-leaf SGML elements, do not convey enough of the contextual information. On the other hand, sections or subsections of a text document, such as a book or an article, can discuss many heterogeneous topics. Furthermore, sections are often longer than desired with respect to the intended purpose, such as document retrieval or assembly.

In this article, we presented a method for detecting microdocuments based on term-frequency distributions. The detection process has two phases: similarity calculation and fragmentation. In general, the results of our preliminary experiments show that the presented approach works well in user-assisted topic-oriented microdocument detection. We currently study the usefulness of this technique in document assembly, i.e., in generating new documents from a collection of existing text documents.

Acknowledgements

This work has been supported by the Finnish Technology Development Centre (TEKES) and industrial partners including leading Finnish enterprises involved in electronic printing and publishing. We would like to thank Juha Kärkkäinen for suggesting the dynamic programming approach for the fragmentation phase of the microdocument detection.

References

[AHHK97] Helena Ahonen, Barbara Heikkinen, Oskari Heinonen, and Pekka Kilpeläinen. Assembling Documents from Digital Libraries. Unpublished manuscript, March 1997.

[Cal94] James P. Callan. Passage-Level Evidence in Document Retrieval. In Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, July 1994.

[CLR94] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. The MIT Press, Cambridge, Massachusetts, USA, 1990.

[Hea94] Marti Hearst. Multi-Paragraph Segmentation of Expository Text. In Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, Las Cruces, New Mexico, USA, June 1994.

[Kim96] W. Eliot Kimber. Re-usable SGML: Why I Demand SUBDOC. In SGML '96, Boston, Massachusetts, USA, 1996.

[McF96] John McFadden. Hybrid Distributed Database (HDDB) and the Future of SGML. In SGML Europe '96, Munich, Germany, 1996.

[Sal89] Gerard Salton. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1989.

[SSBM96] Gerard Salton, Amit Singhal, Chris Buckley, and Mandar Mitra. Automatic Text Decomposition Using Text Segments and Text Themes. In Proceedings of the Hypertext '96 Conference, Washington D.C., USA, 1996.