CLUSTERING DOCUMENTS TO COMPRESS INVERTED INDEX PROJECT

INFORMATION RETRIEVAL AND WEB SEARCH

[CM0473]

Nicolas Pietro Martignon (870034)

Index

1. Dataset used and stop word file.
2. Language and main data structure used.
3. Steps of the program
4. How to compile and run
5. Result of test
6. Conclusions

1

Dataset

It was decided to use a dataset of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005.Natural Classes: 5 (business, entertainment, politics, sport, tech), from: - D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.

Stopword

To further improve the result of the experiment, it was decided to use a stop word file with the main English stop-words. This file was created by joining several stop word files find on web.

2

Language used

It was decided to use **C++** a fast and memory efficient language, because some algorithms (tsp heuristic, Jaccard), turn out to be very heavy due to the numerous documents involved (2225).

Main data structure used

```
class Doc {
public:
    int docId;
    int NewDocId;
    int lenght;
    std::set<std::string> docVocabulary;

    bool operator < (const Doc& other) const{
        return (lenght > other.lenght);
    }
};
```

This class contains all the information of a single document. Its DocId, the new DocId that will be assigned to it, the length in words of the document. Finally, there is a method to make the comparison of the length with other documents.

std::map<std::string, std::vector<int>> posting;
This data structure stores the index, more precisely in the key field it's store a word, while in the value field we save the posting list belonging to the word saved in the key value. In the posting list we save the raw DocId.

std::vector<Doc> vectorDoc;
All documents are saved in this vector

std::set<std::string> stopword;
This set contains all stopwords read from stopwords file

3

Steps of the program

- After reading and loading the stopwords file into memory, the program proceeds by reading all the files one by one. For each file, each word is read and if this is not present in the stopword file and survives the filtering phase, the word will be inserted in the vocabulary (if not present) and the DocId of the current document will be inserted in the posting list. Furthermore the word is inserted in the internal set of doc object (if not present).

- Then, the gap is calculated and the average bit for gap is print using (VB (2 Byte, Byte, Nibbles, Half Nibbles), ELIAS GAMMA CODE, ELIAS DELTA CODE).

- Docs is reorder by length and then Jaccard/Jaccard Normalized is used to create medoid. If the distance between one doc to all the medoids is larger than a certain threshold we set the doc as new medoid.

- After finding all the Medoids, Jaccard/Jaccard Normalized is reapplied between the Medoids themselves.

- Now TSP is calculated on medoid. For implement this it was decided to use a heuristic method, the nearest neighbour (NN) algorithm, which is very fast and offers a solution that's on average yields a path 25% longer than the shortest possible path. The exact method is infeasible because the complexity is O(n!).

- Once we have the minimum path between the Medoids, we iterate through it and through doc into medoid, to reassign the new contiguous DocIds to the documents. The new DocIds is saved in the attribute NewDocId in every Doc Object. So for this we have to rescan the posting list to rewrite new doc id.

- Once we do this, the posting list is it no longer in order, so we we have to reorder it.

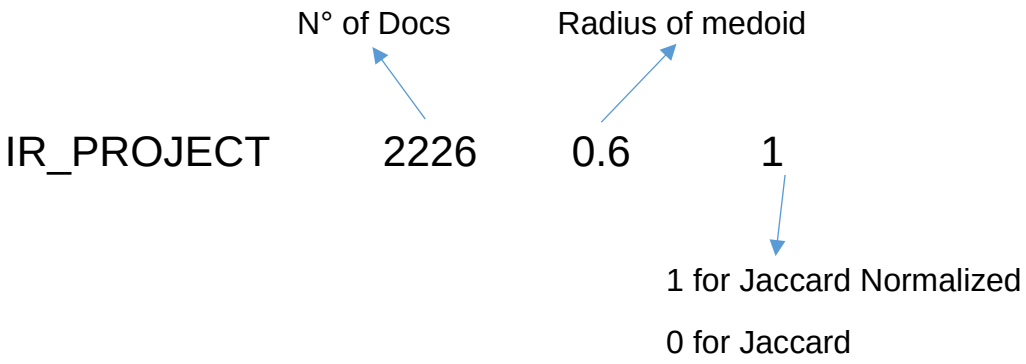
- At the end, the gap is calculated and the average bit for gap is print using (VB, ELIAS GAMMA CODE, ELIAS DELTA CODE).

4

How to compile

```
g++ main.cpp -o IR_PROJECT
```

How to run



In the same folder as the executable put the text files on which to run the test, they must be numbered from 1 to N with extension .txt, example -> (1.txt). In the same folder as the executable put also the stop-word file (STOPWORD.txt).

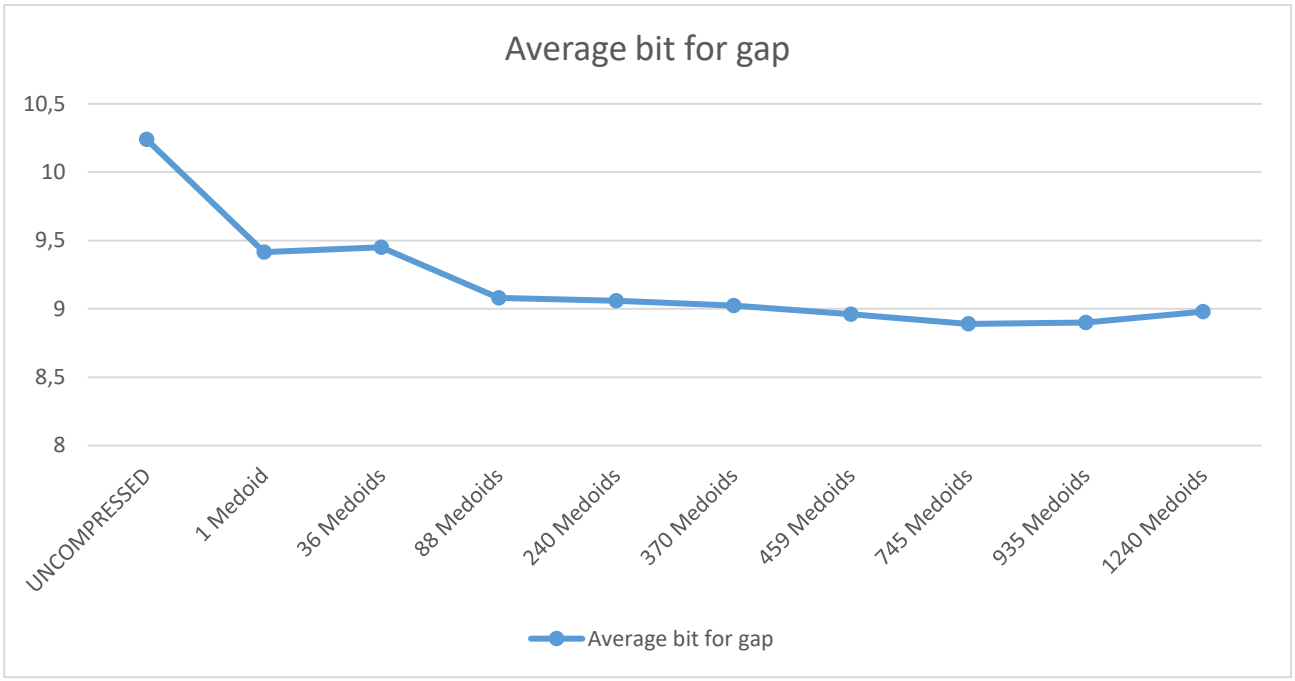
5

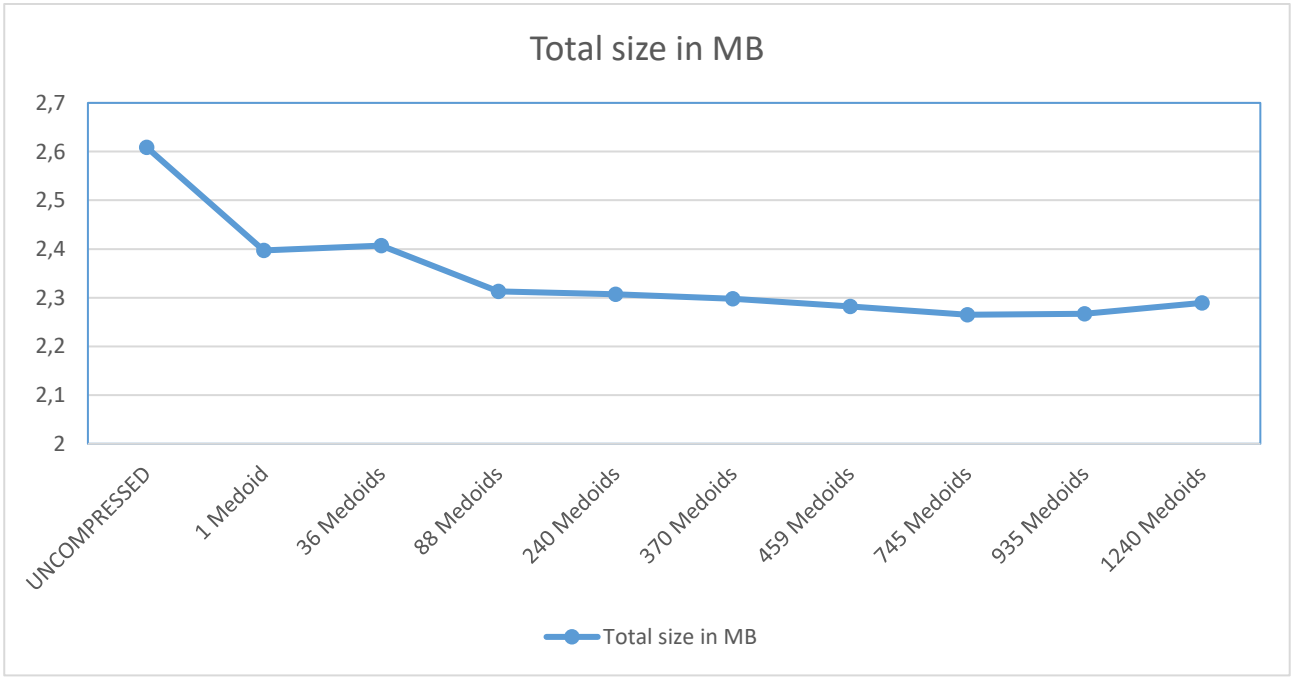
The tests were conducted with all 2225 documents using Jaccard normalized.

For each test the radius of the medoids is decreased until there is no longer any improvement.

We take in consideration the improvements with Elias gamma code.

Result of test





With 1 medoid there is already a significant improvement, but this is not due to the medoid but to the fact that the documents are reordered according to their length in words, this helps the compression a lot.

6

Conclusion

From the tests conducted with different radii and therefore different Medoids, we have been able to observe that there have been visible and notable improvements.

There was a significant improvement only by reordering the documents on the basis of decreasing length, in fact the improvement from the uncompressed index (**2.61 MB**) to the compressed index with only 1 Medoid (**2.4 MB**) was about **8.1%.**

There was another significant improvement from 1 medoid (Only reordering doc) to 735 medoids. In fact the improvement was from **2.4 MB** to **2,26 MB** that is about **5.9%.**

From the results we can declare that clustering documents and reassigning the DocIds following the tsp induced order, helps a lot in index compression, more precisely we have a total improvement of **14%**.