Project

CLUSTERING DOCUMENTS TO COMPRESS INVERTED INDEX

Inverted index to be compressed

```
SPIMI-Invert(token_stream)
  output_file = NewFile()
  dictionary = NewHash()
  while (free memory available)
  do token ← next(token_stream)
      if term(token) ∉ dictionary
        then postings_list = AddToDictionary(dictionary, term(token))
        else postings_list = GetPostingsList(dictionary, term(token))
      if full(postings_list)
        then postings_list = DoublePostingsList(dictionary, term(token))
      AddToPostingsList(postings_list, docID(token))
  sorted_terms ← SortTerms(dictionary)
  WriteBlockToDisk(sorted_terms, dictionary, output_file)
  return output_file
```

Empty posting list

docID are naturally sorted, but they can be reassigned

DocID reasssignment

- The **clustering property** of posting lists indicates that term occurrences are not uniformly distributed.
  - **Some terms** are more **frequently used** in some **parts of a collection** than in others.
  - For example, in a news collection, the term "Ucraine" may occur much more frequent in the documents of 2022 than in other subsets.
- By transforming the posting lists into **d-gaps**
  - The corresponding part of the inverted list will mainly be **small d-gaps clustered**.
  - To obtain **small d-gaps** it should be better to assign consecutive DocIDs to that **part of a collection** where frequently **co-occur groups of words**

DocID reasssignment

- At the end, small **d-gaps** are much more frequent (high probability) than large ones within postings lists
  - variable-length encoding schemes allow indexes to be compressed very well by using **shorter codes** for **small d-gaps**
- Research Question: May we permute the DocID assignment to increase the frequency of small d-gaps, thus exploiting clustering property of the index?
  - If yes, we may increase the compression of the index
  - Issue: we must apply the same order to all the posting lists

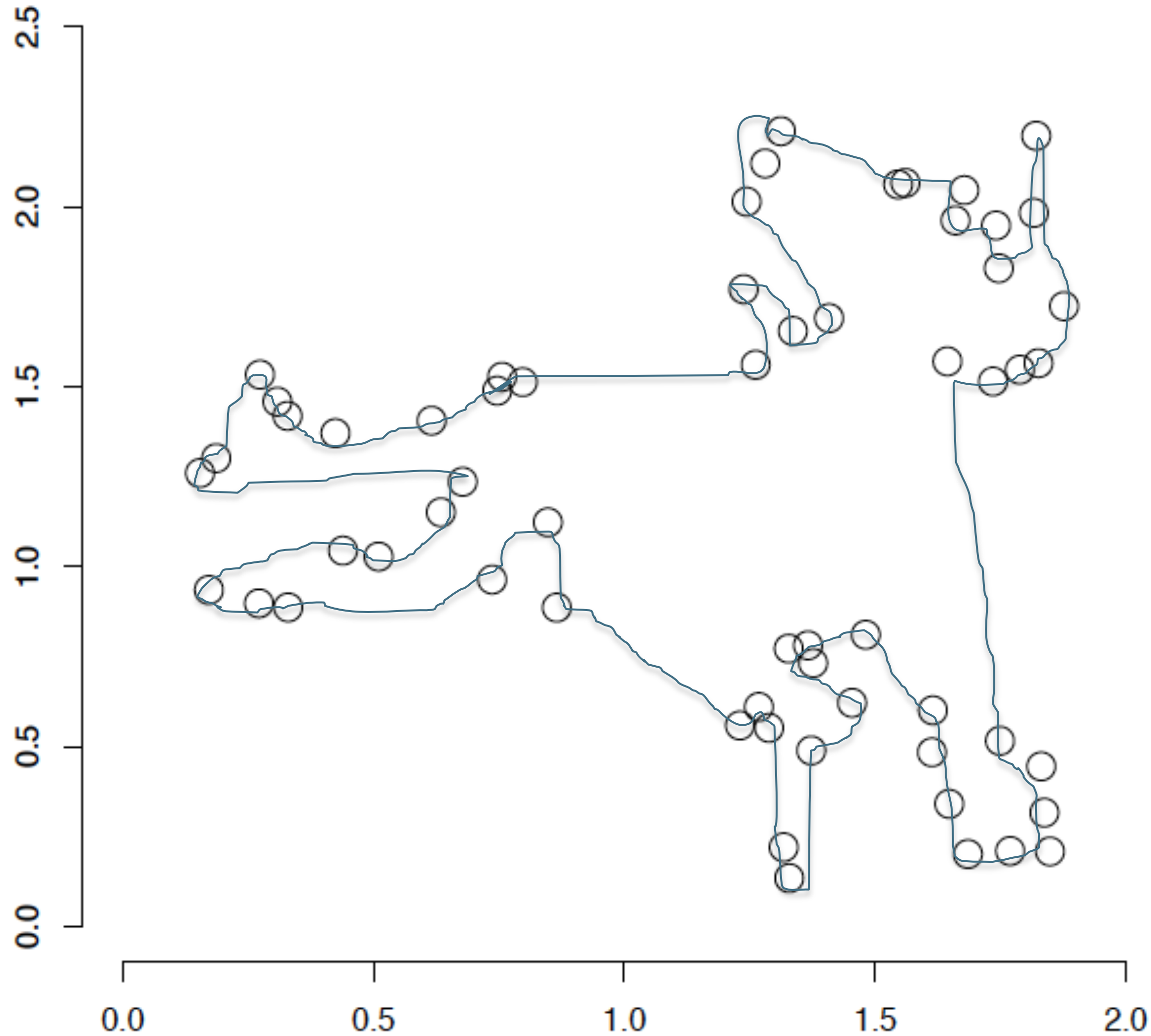
DocID reassignment - TSP

- A technique proposed in the literature is based on the traveling salesman problem (TSP)
  - The heuristic computes a *pairwise distance* between every pair of documents (<u>complete</u> *distance matrix*!!)
  - Proportional to the number of shared terms (documents as sets)
  - e.g., **Jaccard distance** = *1 – JaccardSim*
    - *1: max distance*   *0: identical documents*
- If two docs $\mathbf{d_i}$ and $\mathbf{d_j}$ have a high value of *JaccardSim*
  - If they share many terms.
  - Let $\mathbf{T_{shd}}$ be the set of shared terms
  - Limit case:
    - Assign two consecutive DocIDs to $\mathbf{d_i}$ and $\mathbf{d_j}$
    - For each $\mathbf{t} \in \mathbf{T_{shd}}$ the gaps in the associated posting lists between the two docs $\mathbf{d_i}$ and $\mathbf{d_j}$ will be equal to **1**, i.e., max compression

DocID reassignment - TSP

- Indeed, TSP is used to approximate the **shortest cycle** traversing all documents in the graph.
  - The cycle is finally broken at some point
  - the DocIDs are reassigned to the documents according to the ordering established by the cycle
  - Close documents in the cycle share many terms

TSP



- Example of TSP library for Routing Optimization:
  - https://developers.google.com/optimization/routing/tsp?hl=en#search_strategy

DocID reassignment - TSP

- The rationale of TSP usage
  - the TSP cycle preferably traverses edges connecting documents sharing a lot of terms (characterized by a small Jaccard distance)
  - if we assign close DocIDs to these documents, we expect a reduction in the average value of *d-gaps* (<u>in the posting lists of the shared terms</u>) and thus in the overall size of the compressed inverted index

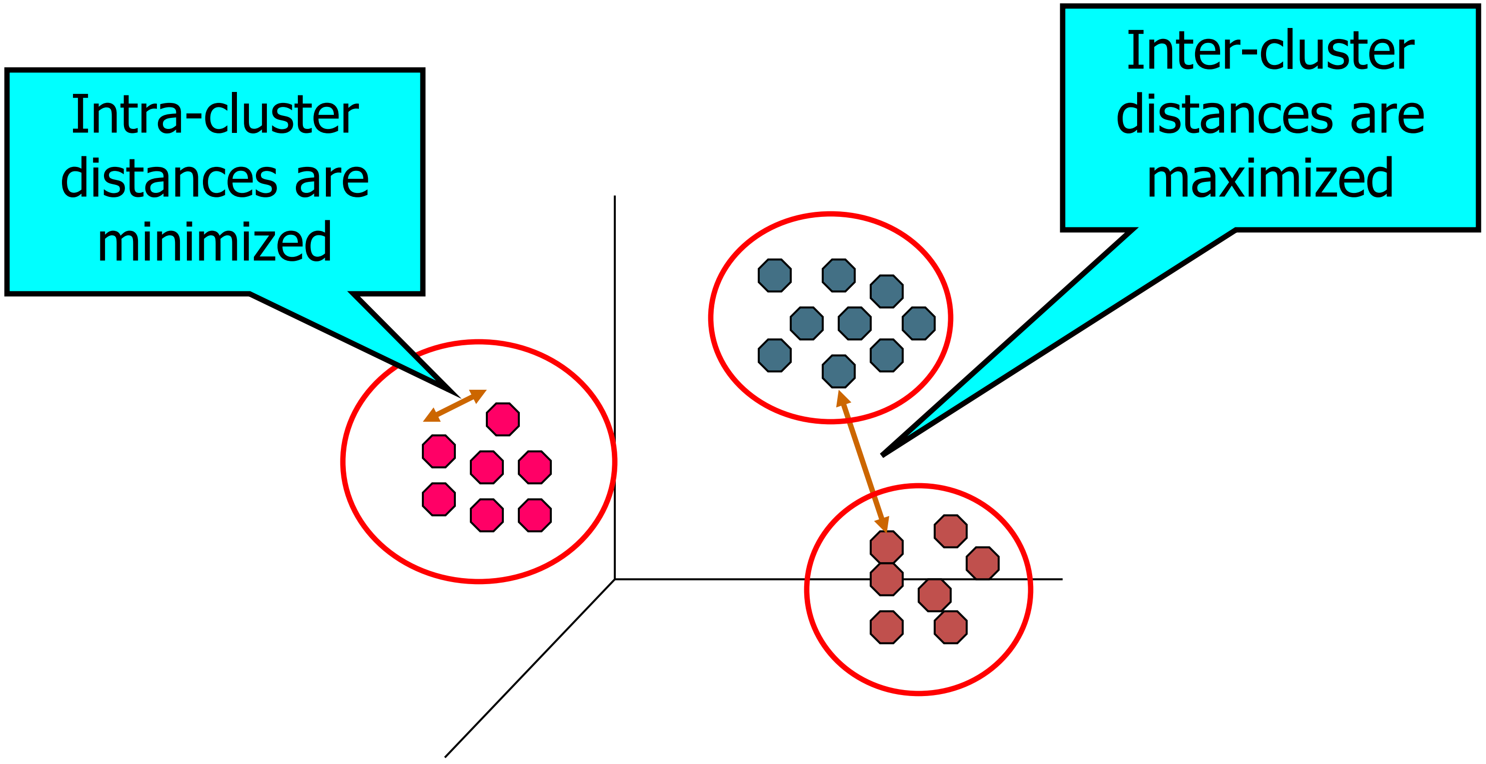
- However, this TSP approach doesn't scale

What is clustering?

- Clustering: the process of grouping a set of objects into classes of similar objects
  - Documents within a cluster should be similar.
  - Documents from different clusters should be dissimilar.
- The commonest form of *unsupervised learning*
    - Unsupervised learning = learning from raw data, as opposed to supervised data where a classification of examples is given
  - A common and important task that finds many applications in IR and other places

What is Cluster Analysis?

- Finding groups of objects such that the objects in a group will be similar (or related, or less distant of) to one another and different from (or unrelated to, ore more distant of) the objects in other groups



DocID reassignment: possible scalable solution

- (1) First **cluster the document collection**
- (2) **Reorder clusters** (rather than single documents) by exploiting TSP, using the representative document of each cluster
- (3) Assign the DocIDs linearly, cluster by cluster, using the TSP-induced order. Within each cluster the order is arbitrary.

DocID reassignment: possible scalable solution

- Possible clustering algorithm (single scan)
  - **scan linearly** the documents, sorted in reverse order of length
    - a doc with many terms should be closer to much more docs, measured by Jaccard distance
  - Each cluster returned will be identified by a **medoid**, i.e., a document that represents all the others in the cluster
  - The medoid should be the **most centrally located** point in the cluster. *However, the stream nature of the clustering algorithm does not guarantee this property of medoids*

DocID reassignment: possible scalable solution

- Transform each document into a set of **termIDs**
- Reorder the collection according to the document length (in reverse order)
- Scan linearly the collection of document to clustering them using **the Jaccard distance = 1 - JaccardSim**

```
C = Stream_cluster(SortedCollection, Radius)
```

  where `C` is the returned set of clusters, each cluster represented by its *Medoid*.
- Apply TSP to the Medoids of each cluster, using the Jaccard distances between each pair of Medoids
- Assign the DocIDs linearly, cluster by cluster, using the TSP-induced order. Within each cluster the order is arbitrary.
- For each **postings list**, reassign the docIDs, compute the d-gaps, and determine the total **size of all posting lists**, along the **avg no. of bits per posting**
  - In this phase, it suffices to determine the average bits per d-gap.
  - For example, for VB, the **bits for a posting *G*** are: $\left\lceil \frac{\lfloor \log G \rfloor + 1}{7} \right\rceil * 8$

DocID reassignment:

possible scalable solution (single scan k-means)

- The pseudo-code of the stream algorithm that visits each document only once is the following, where **Radius** is the hyperparameter:

Stream_cluster(SortedCollection, Radius)

$C = \emptyset$

for each d in **SortedCollection**

Dist_c = *Min* (JaccardDistance(c, d), for each medoid c in C)

if (Dist_c < **Radius**) then

add d to cluster c

else

make d a new medoid, and add it to C: $C = C \cup d$

return C

DocID reassignment:

alternative solutions (two-scans o K-medoids)

- Stream algorithm visiting each document two times is the following, where **$\text{Radius}_1 > \text{Radius}_2$**

  C = **Stream_cluster(SortedCollection, Radius_1)**

  foreach c_i **in** C:

          Stream_cluster(c_i**, Radius_2)**

- Using **K-Medoid**s on a sample of the dataset
  - Use a suitable *k*
  - Assign the rest of the dataset to the closest centroids, still using *cosine*

DocID reassignment:

evaluations

- **Execution time vs. number of clusters**
- **Compression rate (n. bits per posting)** as a function of the **number of clusters (and the clustering methods)**, using:
  - Gamma, Delta, Variable Byte, PForDelta
  - How **compression rate changes** as **the clustering is modified**?
- The RCV1 small collection doesn't contain stop words, so we have fewer opportunities for compression.
  - So, it should be interesting to use another real collections
- Store/materialize the index in a compressed binary form, e.g., Variable byte and PForDelta
  - Evaluate the **decompression time** for the first 10 or 100 most frequent terms by using the different methos

DocID reassignment:
alternative solution (kMeans)

- Using K-Means on a sample of the dataset
  - Need to use a distance that allows computing the mean vector (centroid)
  - *Cosine*, using normalized doc vectors (weights computed as TF-IDF, normalized by dividing by the doc lengths)
- Use a suitable *k*
- Assign the rest of the dataset to the closest centroids, still using *cosine*
- Apply TSP to the centroids, and assign DocIDs as in the previous case

- Compute the Upper Bound to the **impact weight** of each posting list (as used in WAND)
  - Maximum weight contribution of each posting list
  - MAX BLOCK also uses the maximum impact of each block (of 64 or 128 DocIDs)
  - In “Faster Top-k Document Retrieval Using Block-Max Indexes”, SIGIR ’11, Ding and Suel states
    - DocID reassignment gives some benefits, as the distribution of impact values in each block becomes more even
  - Measure the evenness of impact values per block before and after the DocID reassignment

DocID reassignment: alternative solution (kMeans)

- The use of TF-IDF weights in clustering opens to a new analysis

- WAND uses an Upper Bound to the **impact weight** of each posting list
  - It's simply the maximum weight contribution of each posting list

- MAX BLOCK uses the maximum impact of each block (of 64 or 128 DocIDs)
  - In "Faster Top-k Document Retrieval Using Block-Max Indexes", SIGIR '11, Ding and Suel make the following statement:
    - *DocIS reassignment gives some benefits, as the distribution of impact values in each block becomes more even*
  - Measure the evenness of impact values per block before and after the DocID reassignment with a suitable statistical measure