Project

CLUSTERING DOCUMENTS TO COMPRESS INVERTED INDEX

FROM "INDEX CONSTRUCTION" PDF

Remaining problem with sort-based algorithm

but if not possible?

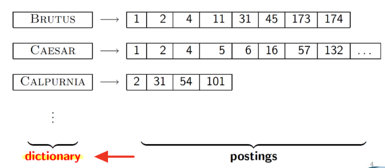
- Our *implicit assumption* was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term → termID mapping.
- Actually, we could work with term,docID postings instead of termID,docID postings . . .
- . . . but then intermediate files become very large. (We would end up with a scalable, but very slow index construction method.)

So,

SPIMI: (use memory in more clever way)
Single-pass in-memory indexing

- Key idea 1: Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur (appending) (parse document in order and assign an incrising id)
  - DocIDs are generated and assigned sequentially to documents
- With these two ideas we can generate a complete inverted index for each block. (generate directly a complete index for each block)
- and These separate indexes can then be merged into one big index.

REMEBER THAT:



SPIMI-Invert

Size of block (aimed to store the index of a chuck of documents)

Empty posting list

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*, *postings_lists*, *output_file*)
return *output_file*

ordino i termini nel dizionario (chiavi) alla fine, dopo averlo costruito

mi serve poi per il merge dei termini dei dizionari (due blocchi possono infatti contenere lo stesso termine) e la rinomirazione degli id

- Merging of blocks is analogous to BSBI, but we have to merge vocabulary and may rename docIDs

in fact.

primo non era necessario
perché il dict. era globale

SPIMI: Compression

- Compression makes SPIMI even more efficient.
  - Compression of terms
  - Compression of postings
- See next lecture

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
```

→ FOR THIS TECHNIQUE DOCID REMAP IS NEEDED *

Empty posting list

docID are naturally sorted, but they can be reassigned

REMAP DOC ID:

1 2 1
apple → 1, 3, 4
100 101 90
90 10 1
→ 90, 100, 101

⇒ Exists a remapping that increase the compression? (that reduce the gap ON AVERAGE) *

DocID reasssignment

- Small **d-gaps** are much more frequent (high probability) than large ones within postings lists
  - this feature of posting lists is called **Clustering property**, and is passively exploited by compression algorithms
  - 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?
  - If yes, we may increase the compression of the index
  - Issue: we must apply the same order to all the posting lists *

(vedo disegno sotto)

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 (complete *distance matrix*!!)
  - proportional to the number of shared terms (documents as sets)
    (se condividono molti termini sono simili (euristica))
  - e.g., **Jaccard distance** = *1 – JaccardSim*
    (can be used also COSINE SIMILARITY)
    - *1: max distance*   *0: identical documents*

successivamente

- Then use TSP to find the **shorted 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

*

dx dy {a, b, c, d, e}

a → … dx … dy

b → … dx … dy

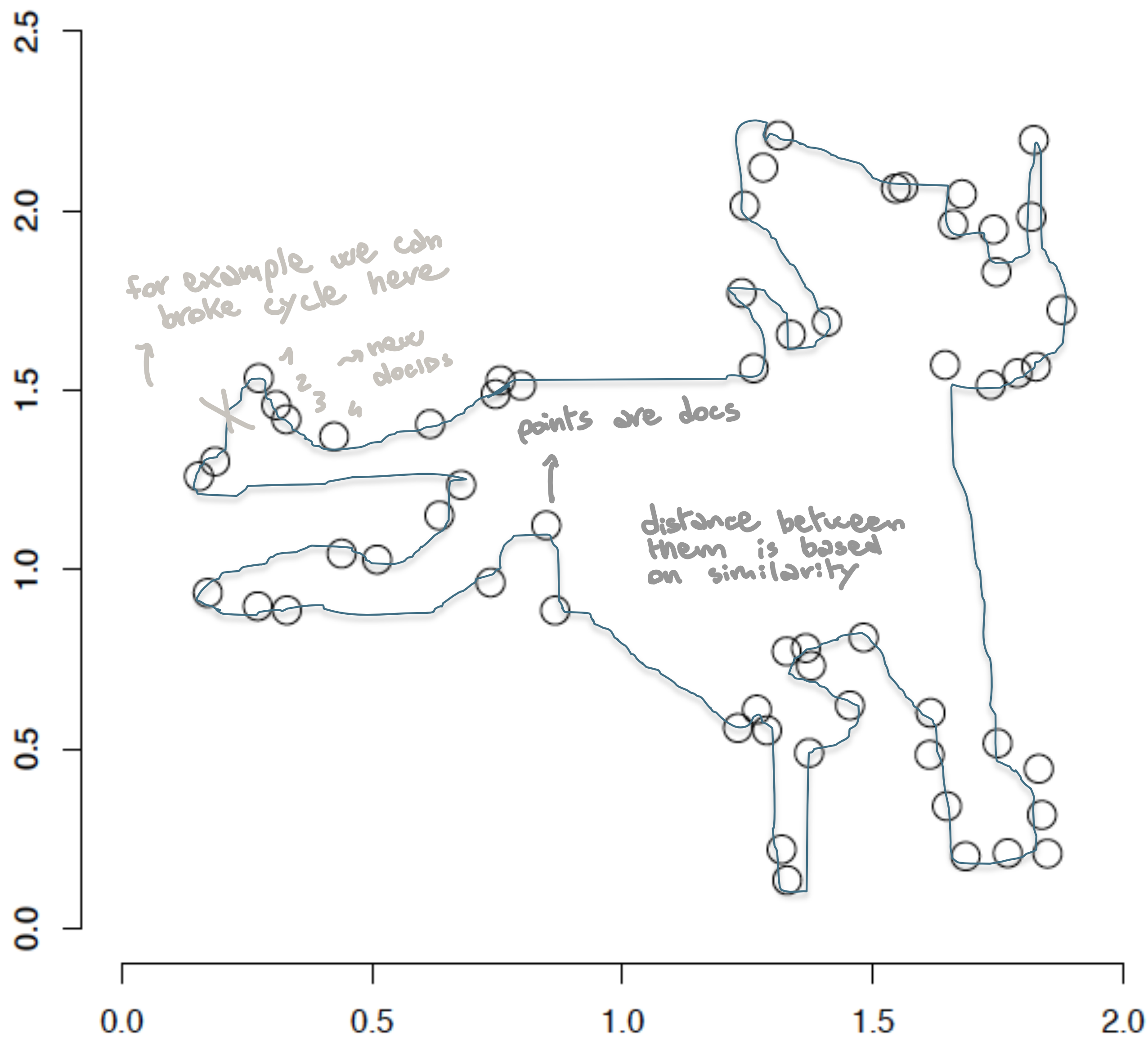
very small gap

⋮

e → … dx … dy

if they are similar they share a lot of terms so if we remap docID with ID close for these doc the gap between doc is smaller

$$J = \frac{|dx \cap dy|}{|dx \cup dy|}$$



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

EXAMPLE DATASET THAT CAN BE USED "RCV1"

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* (in the posting lists of the shared terms) and thus in the overall size of the compressed inverted index

- However, this TSP approach doesn't scale

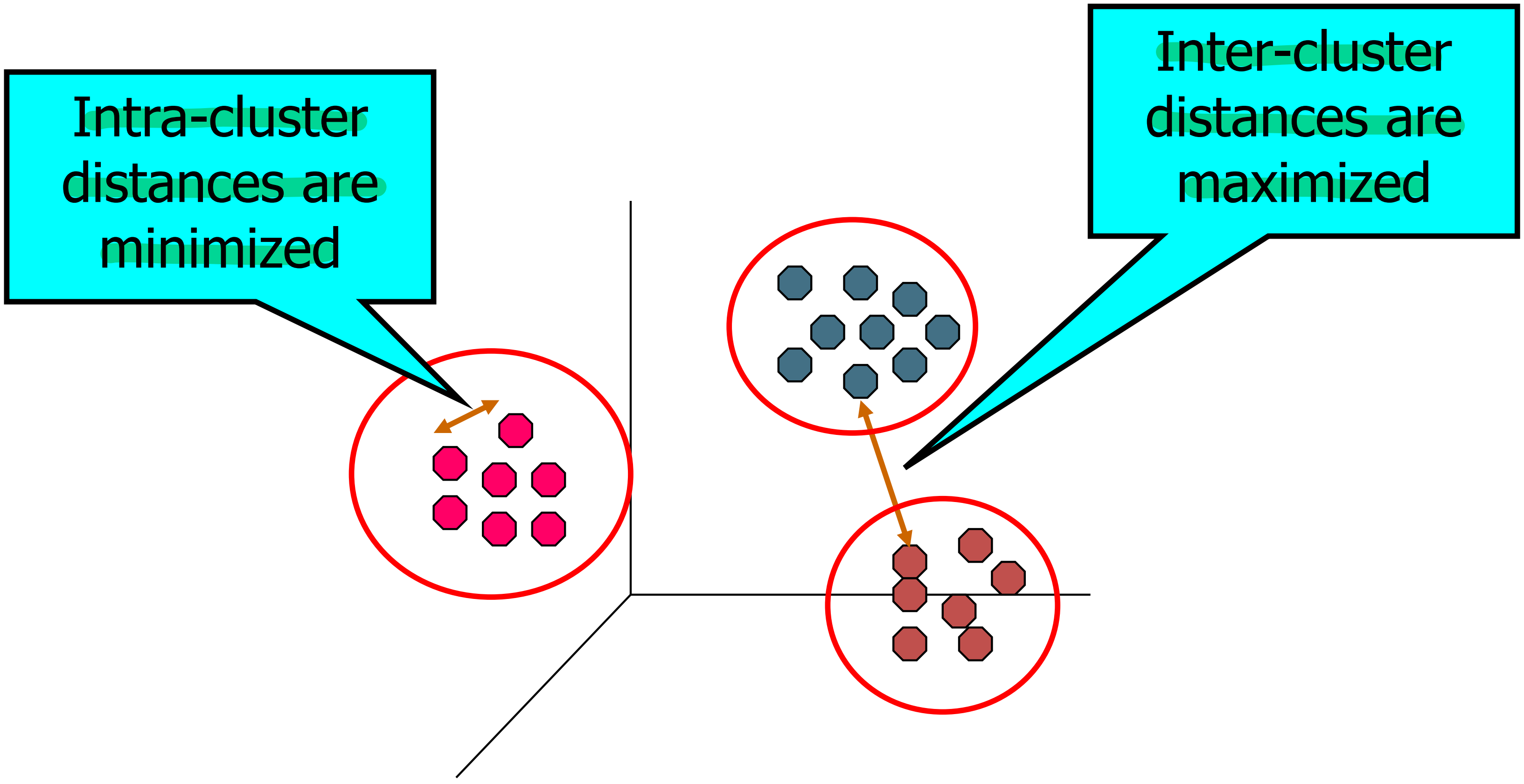
↳ TSP is NP-hard and it's expensive to consider a lot of graph's points

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

(So use TSP NOT ON SINGLE DOCS, but to CLUSTER of DOCS)

ex. use centroid to represent the cluster

cycle of centroid,
cut the cycle and assign
the docids

DocID reassignment: possible scalable solution

- Possible clustering algorithm (single scan)
  - **scan linearly** the documents, sorted in reverse order of length ⇝ an heuristic to select good medoids
    - 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

doclength: cardinality of doc. terms

$doc_i = \{term_{i_1}, term_{i_2}, \dots\}$

IN DETAILS:

- Transform each document into a set of **termIDs** (to simplify jaccard)
- 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)
```

~> next slide!

  where C is the returned set of clusters, each cluster represented by its *Medoid*. (if radius is small we have a lot of cluster)
- 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.
  It is not needed to materialize the compressed posting lists, but it suffices to determine the average bits per d-gap.
  - Compute avg bit for posting, e.g., for VB, the bits for a posting *G* are: $\left\lceil \frac{\lfloor \log G \rfloor + 1}{7} \right\rceil * 8$

perché 1 Byte = 8 bits

DocID reassignment:

possible scalable solution (single scan k-means)

ALG. CITATO SOPRA:

- The pseudo-code of the stream algorithm that visits each document <u>only once</u> is the following:

in our case [0,1] because we have Jaccard dist.

Stream_cluster(SortedCollection, Radius)

$C = \emptyset$

for each *d* in **SortedCollection**

Dist_c = *Min* (JaccardDistance(c, *d*), for each <u>medoid</u> c in *C*)

if (Dist_c < **Radius**) then

add *d* to cluster c

else

make *d* a new <u>medoid</u>, and add it to *C*: $C = C \cup d$

return *C*

in the first iteration "d" becomes the medoids

DocID reassignment:

alternative solutions (two-scans o K-medoids)

- Stream algorithm visiting each document two times is the following, where **$Radius_1 > Radius_2$**

↳ in the second step apply alg. with smaller radious (a kind of hierarichical clustering)

$r_1 > r_2$: bigger and smaller radious

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

foreach c_i **in** *C*:

  Stream_cluster(c_i, $Radius_2$)

DocID reassignment:

alternative solution (kMeans) or K-Medoids

ANOTHER VARIANT IF DATASET IS HUGE:

- 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

DOCID reassignment can be beneficial also if WAND is used (can be analyzed also this value in project)

- Upper Bound to the **impact weight** of each posting list (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

- WANS 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 <u>maximum impact</u> 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 <u>more even</u>*
  - Measure the evenness of impact values per block before and after the DocID reassignment with a suitable statistical measure