Hierarchical Indexing for Interactive Zooming of Document Clusters

Hala Saadeh[1] and Joe Tekli[1*]✉

[1] E.C.E. department, School of Engineering, Lebanese American University
36 Byblos, Mont Lebanon, Lebanon
hala.saadeh@lau.edu, joe.tekli@lau.edu.lb

Abstract. The increasing availability of large collections of textual data online gives rise to the challenge of developing methods to explore and organize large document collections in a meaningful and pragmatic way. Thematic exploration solutions allow users and organizations to effectively visualize document clusters and browse them thematically or topically. Existing solutions provide methods to topically visualize document clusters. However, they do not allow the user to dynamically explore the clusters online by zooming in and out of the cluster hierarchy. Also, most studies do not provide a solution for persisting the document clusters offline to be re-used and visualized at a later time. In this study, we design and implement a hierarchical indexing solution that allows dynamic and interactive zooming of document clusters. Our solution consists of two major components: 1) a dynamic indexing component that produces a topical and hierarchical index for document clusters, and 2) a visualization component that provides interactive and thematic exploration functionalities. The index is dynamically updated online using incremental clustering following the user's exploration. Users can seamlessly zoom in and out of clusters to explore topically related data. Our solution serves as a baseline for an interactive cluster zooming solution and can be extended to support semantic zooming and more advanced querying functionalities at a later stage.

Keywords: Data indexing, Hierarchical clustering, Incremental clustering, Topical organization, Thematic exploration, Data zooming.

1 Introduction

Thematic exploration is the concept of allowing users and organizations to effectively visualize document clusters and browse them thematically or topically. A common technique in thematic exploration is grouping similar documents into clusters of related topics. These clusters are then organized to produce effective visual exploration methods for users. Few existing methods have addressed cluster-based thematic exploration, e.g., [4-6]. While they organize groups of documents into thematic clusters, nonetheless, most existing methods do not allow the user to dynamically explore the clusters online by zooming in and out of the cluster hierarchy. Also, most studies do not provide a solution for persisting the document clusters offline to be re-used and visualized at a later time.

In this study, we present a new solution for cluster-based document exploration. Our solution consists of two main contributions: i) an indexing component that

[*] The author is co-founder of the United Nations ESCWA Knowledge Hub (UNEKH), which framework provided the main use case for this project.

accepts as input a set of unstructured text documents and produces as output a hierarchical and topical index structure of document clusters, and ii) an adapted visualization component that provides seamless, interactive, and thematic exploration of document clusters. Different from existing solutions, our proposal explicitly addresses the issue of cluster formation, by i) building a novel cluster index structure adaptively following a two-phase process: hierarchical (offline) clustering for priming; and incremental (online) clustering following the user's exploration, and ii) allowing interactive cluster exploration online using the generated cluster index structure, where iii) the index is specifically designed to allow seamless zooming in and out of the clusters, following the user's needs. Our design is modular and allows the decoupling of the indexing and visualization components as separate and fully functional services, according to the user's needs. Empirical results highlight the effectiveness, efficiency, and practicality of our solution.

Section 2 reviews related works. Section 3 describes our proposed solution. Section 4 presents our experimental tests. Section 5 concludes with future directions.

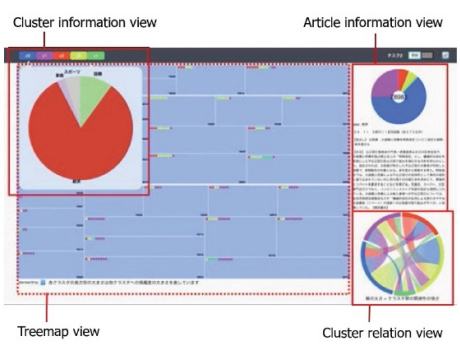
2 Related Work

2.1 Topical Exploration of Web Data

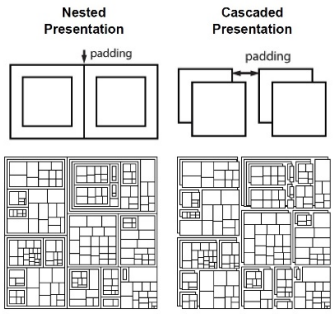
Providing techniques for Web data search, exploration, and visualization is gaining importance, where topical exploration and result organization become essential to allow easier and more effective access to the data. In [4], the authors introduce the concept of *inCloud* to transform the flat organization of data into a special structure of clustered topics using dedicated data aggregation and abstraction techniques. An inCloud provides a high-level topical view of the data and consists of 3 main components: i) a circle box, representing a cluster, which is a group of data that focus on a specific topic, ii) a square box which representing a summary of the contents of the cluster, and iii) an arrow, which represents the relationship between clusters. The thicker the arrow is, the more connected the two clusters are. To build an inCloud, the first step is to perform similarity evaluation on the input data and produce a similarity graph where similarity links are added to connect the data nodes together. The second step is to perform topical aggregation to identify a set of topical clusters within the connected similarity graph. Clusters are formed by going through the similarity graph and detecting those data nodes that are highly interconnected. To do this, the authors in [7, 8] rely on the clique percolation method (CPM). CPM divides the graph into sub graphs, where each sub-graph has k connected nodes. Two sub graphs are defined as adjacent if they share k-1 nodes. In a subsequent study in [6], the authors introduced *inWalk*, an interactive system for the exploration of linked data based on the concept of inCloud from [4]. In [5], the authors build on the concept of inClouds from [4] to define inClouds as entity-driven collections of Web resources aiming at providing information organization structures. An inCloud is used to collect information relevant for a given target entity. Hence, resources considered prominent to the target entity are properly arranged and presented to the user. In [12], the authors describe LDAExplore, a solution for organizing topic distributions using an unsupervised LDA (Ladent Dirichlet Allocation) topic modeling method. It organizes the generated topics according to their word distributions in the corpus, and allows topical filtering and set operations, displaying different views of the document corpus,

and grouping similar documents together according to their shared topics. In [10], the authors introduce a dimensional clustering approach capable of selecting the set of features to use for data clustering, which are then packaged into topical dimensions. This provides a description of the similarity value that generates each cluster. Hence, resources with the same degree of similarity but with different sets of matching features are put in different clusters, resulting in more accurate clustering.

Most of the above approaches describe how to organize data clusters. Yet they only briefly cover cluster exploration. They do not address dynamic navigation or zooming within the clusters, nor do they address cluster loading for visualization.



a. Two-level zooming approach from [27]



b. Cascading cluster presentation from [18]

Fig. 1. Sample cluster visualizations from the literature

2.2 Cluster Visualization and Zooming

Few approaches have specifically addressed data cluster visualization and exploration. Most of them revolve around the treemap visualization tool, e.g., [21]. In [21], the authors describe a treemap based visualization that provides a rapid layout for graph structures. They introduce the *Focus+ context* paradigm, in which a user presses on a node in the treemap to view its surrounding nodes. They also introduce the fisheye view zooming functionality allowing a user to select a cluster and zoom-in to display its children clusters [26]. In [27], the authors present a treemap-based visualization that applies Fuzzy c-means clustering to organize the data, and visualizes the results on the basis of the highest and the second-highest membership values within the treemap. Fuzzy c-means is applied iteratively to create a hierarchical structure, where the first layer shows the clustered objects based on highest membership value, and the second layer shows the clustering objects based on the second-highest membership values (cf. Fig. 1). The hierarchy is limited to two levels, allowing the user to click on a cluster and view its child clusters, and vice-versa. In [18], the authors introduce the concept of cascading treemaps, using cascading rectangles instead of the traditional nested rectangles. Cascading uses less space to present the same containment relationship, and the space savings enable a depth effect and natural padding between siblings in complex hierarchies. The authors provide one level of parent-child zooming in their implementation [18].

In short, most existing methods i) do not address the issue of cluster formation, i.e., whether the documents are clustered offline or online, ii) do not discuss the issue of cluster visualization loading, which also depends on cluster formation and the lack of persistence of the clusters offline for re-use and visualization, and iii) do not address dynamic navigation or zooming within the data clusters.

3 Proposal

The overall architecture of our solution is shown in Fig. 2. It consists of two main components: i) an indexing component consisting of a two-phase process: hierarchical clustering conducted offline for priming the index structure; and fast incremental clustering conducted online to update the index structure following the user's exploration, and ii) a visualization component which is run online, and involves cluster exploration and interactive zooming. We describe each of the components in the following sub-sections.

3.1 Indexing Component

Our indexing component accepts as input a set of text documents, and performs document preprocessing, clustering, and topical extraction from the generated clusters, and then produces as output an index structure describing the topical organization of the document clusters. The initial index structure is generated offline using hierarchical clustering, and is subsequently updated online using fact incremental clustering following the user's exploration. The index structure is used as input for the online visualization component to perform document cluster exploration and zooming.

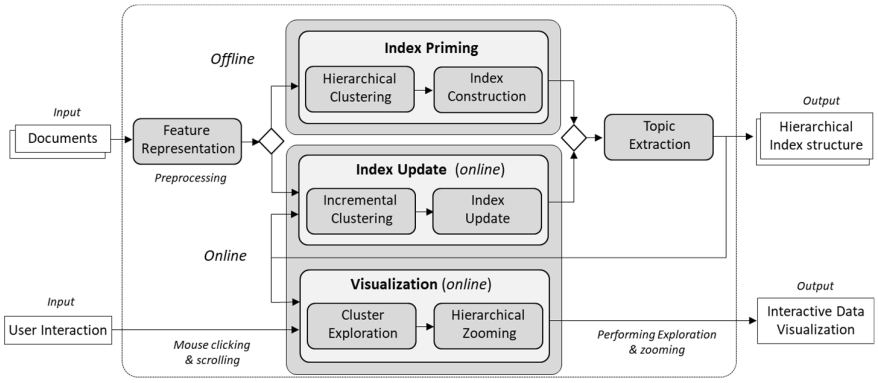


Fig. 2 Overall architecture of our solution

3.1.1 Document Preprocessing and Feature Representation

First, the documents are serialized to extract their raw textual contents from the input pdf documents. Second, all words are converted to their lowercase form, and all stop-words and punctuations are removed from the text. Third, words are reduced to their stems or roots, using stemming or lemmatization, following the users' preference: i)

stemming converts all the words to their original syntactic forms (stems) using syntactic stemming rules[1]; ii) lemmatization transforming words into their original lexical forms using a lexical reference[2]. The documents are then processed for feature representation, where each document is represented as a multi-dimensional feature vector, where vector weights are computed using supervised TF-IDF weighting. We introduce a supervised weighting scheme derived from a modified version of the Term-Frequency Inverse-Category-Frequency (TF-ICF) scheme from [3]. Different from existing approaches which are designed for document representation, we adapt TF-ICF to produce weighted representations for the generated clusters. We augment each document with the cluster features to improve clustering quality. To do so, we represent each cluster by a TF-ICF vector where the dimensions represent distinctive terms and the weight of each dimension reflects the frequency of occurrence of the term in the documents belonging to the cluster. We then embed the new weighting scheme within the documents' original TF-IDF feature vectors to capture the relationships between document terms and the generated clusters.

Table 2. Term Frequency - Inverse Cluster Frequency (TF-ICF) model adopted in our study

Variable	Description
TF	$TF_i^j = freq_{c_j}(t_i) = \sum_{d_k \in c_j} freq_{d_p}(t_i)$
CF	$CF(t_i) = freq_{t_i}(C)$

Variable	Description
ICF	$ICF_i = \log\left(\frac{\lvert C \rvert + 1}{CF(t_i) + 1}\right) + 1$
TF-ICF	$TF-ICF_i^j = TF_i^j \times ICF_i$

We designate by $D = \{d_1, d_2, \ldots, d_{|D|}\}$ the set of documents, $T = \{t_1, t_2, \ldots, t_{|T|}\}$ the set of terms that occur in the documents in D (the vocabulary of D), and $C = \{c_1, c_2, \ldots, c_{|C|}\}$ the set of generated clusters. We compute the TF-ICF of a term $t_i \in T$ in cluster $c_j \in C$ as shown in Table 2. TF represents the frequency of a term inside the set of documents pertaining to the cluster c_j, where more recurrent terms are assigned higher TF scores. ICF represents the fraction of clusters that contain term t_i, where less recurrent terms are assigned higher ICF scores. The less clusters term t_i occurs in, the more descriptive it will be in distinctively describing the clusters it occurs in, and vice versa (the more clusters term t_i occurs in, the less expressive it will be in distinguishing the clusters)[3].

3.1.2 Hierarchical Document Clustering

One of the motivations of our study is to highlight the hierarchical relationships between document clusters, to allow interactive cluster exploration and zooming. Here, we perform clustering using the well-known Unweighted Pair-Group Method with Arithmetic mean (UPGMA) average link hierarchical clustering method [13, 14], although any form of hierarchical clustering can be utilized. Given n documents, we construct a fully connected graph G with n vertices and $\frac{n \times (n-1)}{2}$ weighted edges. The

[1] We adopt the Porter Stemmer [32] since it's one the most effective and commonly used in the literature.

[2] We adopt the WordNet lexical dictionary [20] to perform lemmatization, since it's one of the most commonly used machine-readable lexical knowledge bases in the literature.

[3] Our approach is not restricted to TF-IDF/ICF (which performance is widely acknowledged, e.g., [25, 29]) and can utilize other indexing schemes, e.g., LSI (Latent Semantic Indexing) [22], Word2Vec [9], and kernel functions [15].

weight of an edge corresponds to the similarity (distance) between the connected vertices. We adopt an agglomerative clustering approach where each edge in the connected graph initially represents an individual cluster. Consequently, the nearest two clusters (i.e., edges/documents) are combined into a higher-level cluster. This is repeated iteratively at every step to combine the most similar clusters into higher-level clusters, where the similarity between the clusters is computed as the average of all similarities between their constituent edges (i.e., documents), i.e., the mean pair-wise similarity between all pairs of matching documents from both clusters. We utilize the cosine measure to compute document feature vector similarity, due to its prominent usage in the literature, yet other vector similarity measures can be used (e.g., Pearson Correlation Coefficient, Dice, Euclidian, e.g., [15, 30]). Fig 5 shows the dendrogram produced for a sample set of 10 documents, grouped using hierarchical clustering. Each internal node represents a cluster, and each leaf node represents an individual document. For example, one can observe that documents #8 and #9 are clustered together first, which means they're most similar compared with all other documents in the dataset. We can also observe that the root cluster, which contains all 10 documents, is split into two child clusters. These are examples of the kinds of structural relationships we seek to highlight through our index structure.

Algorithm: ***Incremental_Clustering***

Input: C // Set of initial document clusters
S // Stream of newly accessed documents
R // Set of initial cluster representatives
Thresh // Document similarity threshold

Output: Set of updated document clusters: C

```
Begin                                                                          1
  For each new doc ∈ S                                                         2
    Find max(Sim(doc, Rep_i)) where Rep_i ∈ R   // Find maximum similarity document   3
    If max(Sim(doc, Rep_i)) > Thresh   // If similarity above threshold,       4
      Add doc to cluster c_i having Rep_i as representative   // then add document to existing cluster   5
      Update_Cluster_Representative(c_i)   // and update cluster representative   6
    Else                                                                       7
      Create new cluster c_new   // Otherwise, form new cluster               8
      Add doc to c_new                                                         9
      Designate doc as Rep_New                                                 10
      Add c_new to C                                                           11
    EndIf                                                                      12
  If user wishes to update threshold                                           13
    Set Thresh as average similarity of all images in C                        14
  For each new cluster c ∈ C                                                   15
    Find max(Sim(c, c_i)) where c_i ∈ C                                        16
    Combine c with c_i                                                         17
Return C
End
```

Fig 3. Pseudo-code of incremental document clustering algorithm

3.1.3 Incremental Document Clustering

Following the initial hierarchical clustering phase, which is executed offline to prime to index structure, a second phase of fast incremental clustering [1, 24] is executed online to update the clusters produced in the previous phase by processing newly accessed documents (cf. pseudo-code in Fig. 3). The new document is compared with each of the existing cluster representatives using the aggregated similarities of their

constituent documents from the previous clustering phase (cf. Fig. 3, lines 1-1). The algorithm identifies the cluster having the highest similarity with the new document (lines 3-5), and then decides whether the document should be placed in the cluster; or whether a new cluster should be created around the document; based on a user chosen (or system-generated) similarity threshold (lines 6-11). The newly formed clusters are compared and combined with their most similar hierarchical counterparts (lines 14-16), integrating seamlessly into the cluster hierarchy structure. The incremental process continues online in the same manner as new documents are being explored.

```
Algorithm: Index_Construction
Input:
    dendrogram_root: dendrogramNode    // root node of the dendrogram
    root_id: string                    // id of the root being passed to the function
    doc_id_index: hashTable            // documents and their respective IDs
    cluster_topic_index: hashTable     // extracted topics for every cluster
Variables:
    children = dendrogram_root.children: list of dendrogramNode
Output:
    doc_id_index: hashTable            // documents and their respective IDs
Begin
 cluster_topic_index[root_id] = dendrogram_root.topic                                  1
 If length(children) != 0                                                              2
      higher_child_id = root_id + ".0"                                                 3
      lower_child_id = root_id + ".1"                                                  4
          If children[0].distance > children[2].distance                               5
                higher_child = children[0]                                             6
                lower_child = children[2]                                              7
        Else                                                                           8
                higher_child = children[2]                                             9
                lower_child = children[0]                                              10
     Index_Construction(higher_child, higher_child_id, doc_id_index)                   11
     IndexConstruction(lower_child, lower_child_id, doc_id_index)                      12
    Else                                                                               13
     doc_id_index[root_id]=dendrogram_root.document     // reference to the document  14
 Return doc_id_index                                                                   15
End
```

Fig 4. Pseudocode of our index construction algorithm

3.1.4 Index Construction

The purpose of our indexing component is to develop a structure that will store all the necessary hierarchical cluster relationships offline, allowing for seamless online cluster exploration and zooming by the user. These relationships include for each cluster: i) the parent cluster, ii) the child clusters, iii) the sibling clusters, and iv) the cluster's order w.r.t. (with respect to) its siblings. To do so, we put forward an index based on the Dewey numbering scheme [28], which allows reflecting all the above mentioned relationships. Note that other labelling schemes such as pre-order labelling (POL) and level-order father labelling (LAF) can be adapted to the task, e.g., [11, 17]. In Dewey order, each node is given a vector that describes the path from the (dendrogram) structure's root to the target (document) node. Consider our running example dendrogram in Fig. 5, where each component of the path indicates the local order of the ancestor node. Dewey order is considered to be lossless since each path uniquely defines the absolute position of the node within the structure [28]. Also, the ancestor path is implicitly encoded within the node id, which simplifies the retrieval of the parent and child nodes [19], thus making it suitable for our index structure. The pseudocode of our index construction algorithm is shown in Fig 4. It accepts as input the dendrogram structure produced by the hierarchical document clustering algorithm, and produces as output the hierarchical document index. The algorithm starts at the

root node of the dendrogram and assigns it id =0 (line 1). It then checks if the node has children nodes and generates their ids by appending the id of the parent to those of the children (lines 2-4), taking into account the children's relative order in the dendrogram (lines 5-10). This is achieved by assigning the id concatenation .0 to the child that is higher-up on the dendrogram hierarchy, and .1 to the lower child. The algorithm is then invoked iteratively on each of the children nodes (lines 11-14) until all nodes of the dendrogram have been assigned their ids.

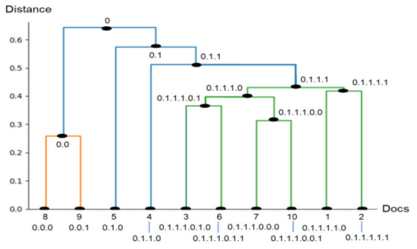
The Dewey labelling of our running example dendrogram is shown in Fig 3.b. One can observe that the root node, i.e., the cluster which encompasses all documents, is labelled with id=0. This id defines the cluster as the root of the index. The two child clusters of the root (i.e., the first-level nodes of the structure) are labeled as 0.0 and 0.1, according to their relative ordering in the dendrogram. Similarly, the id of any given cluster is equal to the id of its parent concatenated with either 0, 1, …, or *n*, where *n* is the number of children nodes of the parent. This enables efficiently determining the parent-child relationships and sibling relationships between the nodes. Also, the index keeps track of the relative ranking between cluster siblings. This labelling allows determining all the required cluster relationships:

i. Determining the *parent cluster*: by removing the last id concatenation from the target cluster label (e.g., the parent of node 0.0.0.0.0 is 0.0.0.0 by removing the last .0).
ii. Determining the *child clusters*: by concatenating the target cluster label with the relative order of the children clusters (e.g., the children nodes of 0.0.1 are 0.0.1.**0**, 0.0.1.**1**, …, 0.0.1.***n*** considering *n* children nodes).
iii. Determining the *sibling clusters*: by toggling the final id concatenation with the relative order of the sibling clusters (e.g., the siblings of cluster 0.0.0.1 are 0.0.0.**0**, 0.0.0.**2**, …, 0.0.0.***n*** considering *n* sibling nodes).
iv. Determining the *order between sibling clusters*: based on the arithmetic order of last id concatenation in the sibling labels (e.g., cluster 0.0.**1** is higher on the dendrogram that its sibling 0.0.**2**, since the **.1** child comes first in terms of relative ordering).

Doc_id	Source file	Title
1	goal_1.pdf	Ending poverty
2	goal_3.pdf	Healthy lives & well-being
3	goal_4.pdf	Quality education
4	goal_5.pdf	Gender equality
5	goal_6.pdf	Clean water and sanitation
6	goal_8.pdf	Decent work and economic growth
7	goal_10.pdf	Reduced inequality
8	goal_14.pdf	Life below water
9	goal_15.pdf	Life on land
10	goal_17.pdf	Partnerships to achieve the goal

Available at: https://www.unescwa.org/ehandbook-sdg-framework-metadata

a. Sample documents from the UN's SDG dataset



b. Dendrogram, with our Dewey-labelled nodes

Fig 5. Dendrogram structure produced for 10 sample documents selected from the UN's Sustainable Development Goals (SDGs) dataset, labeled following our Dewey labeling scheme

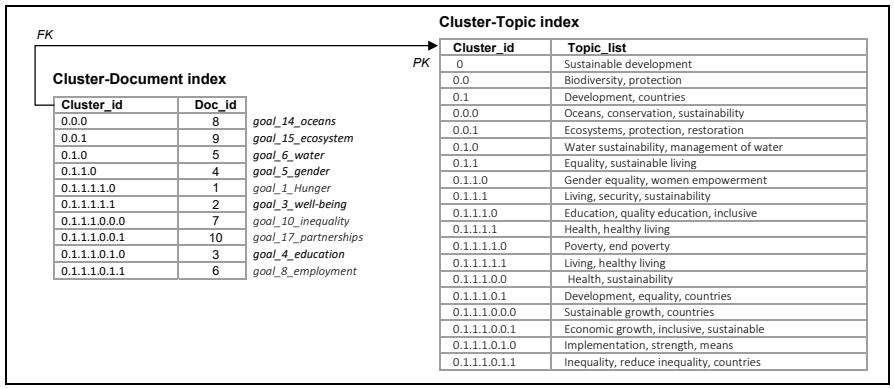
The index structure of our running example dendrogram is shown in Fig. 6. The first table represents the *cluster-document index*, which stores the ids of the leaf nodes (i.e., the individual documents). We do not store the ids of the internal nodes in this

table (i.e., the document clusters) since they are determined from the ids of the leaf nodes, as illustrated previously. The second table represents the *cluster-topic index*, which stores the topics associated with every (internal and leaf) node (i.e., clusters and individual documents), following the topic extraction task described in Section 3.1.5. Note that our index tables are sorted and indexes themselves, using binary search tree indexing, to allow logarithmic search and retrieval time when performing cluster exploration and zooming functionalities.

3.1.5 Topic Extraction

Topic extraction in our approach consists in identifying the most important keywords describing the document clusters. The first step consists in producing the feature vector representations describing every cluster. This is achieved by aggregating the term/expression frequency (and augmented co-occurrence frequency) vectors of their constituent documents. Consequently, we represent every cluster by its top-*k* keywords, ranked following their term weights in the cluster feature vector.

Recent solutions have suggested using external knowledge sources such as Wikipedia or Yago [23, 31], where the topics representing the clusters are identified based on their semantic meaning, and might be selected from outside the clusters themselves [30] (e.g., topic *air pollution* might be selected to describe a cluster containing terms *CO2 emissions* and *greenhouse gases*, such that *air pollution* never appears in the cluster). Yet we restrict our current study to syntactic topical extraction, and report semantics to a dedicated study.



Cluster_id	Doc_id	
0.0.0	8	*goal_14_oceans*
0.0.1	9	*goal_15_ecosystem*
0.1.0	5	*goal_6_water*
0.1.1.0	4	*goal_5_gender*
0.1.1.1.1.0	1	*goal_1_Hunger*
0.1.1.1.1.1	2	*goal_3_well-being*
0.1.1.1.0.0.0	7	*goal_10_inequality*
0.1.1.1.0.0.1	10	*goal_17_partnerships*
0.1.1.1.0.1.0	3	*goal_4_education*
0.1.1.1.0.1.1	6	*goal_8_employment*

Cluster_id	Topic_list
0	Sustainable development
0.0	Biodiversity, protection
0.1	Development, countries
0.0.0	Oceans, conservation, sustainability
0.0.1	Ecosystems, protection, restoration
0.1.0	Water sustainability, management of water
0.1.1	Equality, sustainable living
0.1.1.0	Gender equality, women empowerment
0.1.1.1	Living, security, sustainability
0.1.1.1.0	Education, quality education, inclusive
0.1.1.1.1	Health, healthy living
0.1.1.1.1.0	Poverty, end poverty
0.1.1.1.1.1	Living, healthy living
0.1.1.1.0.0	Health, sustainability
0.1.1.1.0.1	Development, equality, countries
0.1.1.1.0.0.0	Sustainable growth, countries
0.1.1.1.0.0.1	Economic growth, inclusive, sustainable
0.1.1.1.0.1.0	Implementation, strength, means
0.1.1.1.0.1.1	Inequality, reduce inequality, countries

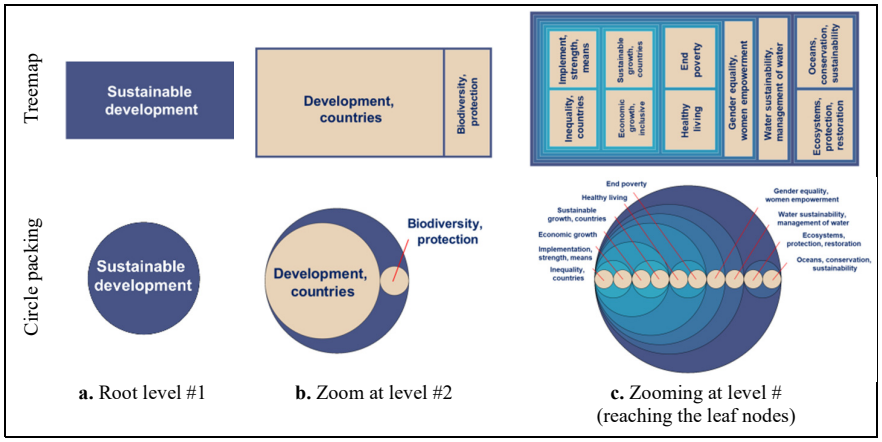
Fig 6. Index structure for sample dendrogram structure produced in Fig. 5

3.2 Visualization Component

The visualization component takes as input our index structure, and uses it to allow interactive exploration and seamless zooming functionalities of the document clusters. Our visualization component is decoupled from the index structure, where the index can be used with other visualization tools following the user's preferences.

3.2.1 Cluster Exploration

We adopt two types of hierarchical visualizations in our current tool (other visualizations can be added in the future): treemap and circle packing (cf. Fig. 7). Both visualizations are designed to initially display the root of the dendrogram as the most comprehensive and zoomed-out cluster, highlighting the most generic information and topical representation of the document dataset (cf. Fig 7.a). Then, as the user scrolls to zoom-in, the inner nodes of the dendrogram representing the more specific levels of the cluster hierarchy are displayed (cf. Fig 7.b and c) along with their representative topics extracted from the index. Zooming in and out is done in a seamless way using the mouse scrolling action. Users can also click on any target cluster to zoom inside of it and acquire its children clusters and their topics, providing a more detailed description of the target cluster.



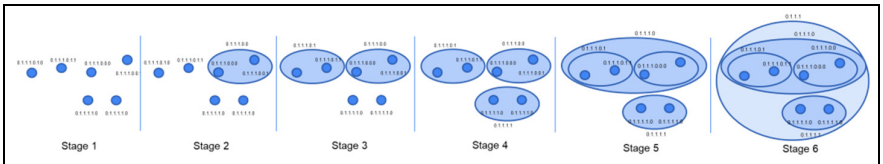
a. Root level #1 **b.** Zoom at level #2 **c.** Zooming at level # (reaching the leaf nodes)

Fig 7. Cluster visualizations using treemap and circle packing tools

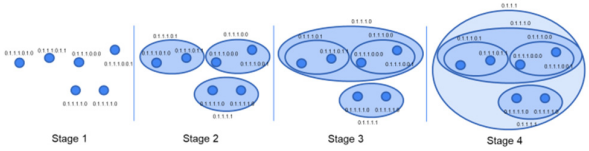
3.2.2 Hierarchical Zooming

We consider two approaches for zooming in and out of the hierarchical clusters: i) *sequentially* according to the dendrogram structure, and ii) *by grouping* sibling clusters, bypassing the dendrogram structure (cf. Fig. 8). Using the former approach, each zoom (or scroll) operation consists in increasing or decreasing the dendrogram's hierarchical level by 1 (i.e., ±1 level depending on the direction of the zoom/scroll, cf. Fig. 8.a). Using the latter approach, each zoom (or scroll) operation groups the existing siblings together, leaving the lone clusters untouched. The siblings to be grouped are identified through their Dewey labels from the index (Fig. 8.b).

While zooming sequentially may be more straightforward, following the structure of the dendrogram, yet it might also be tedious and might take too long for the user to zoom-in or out of the data, especially when a large number of deeply nested clusters are involved. Zooming by grouping may alleviate the task of zooming in and out of largely nested clusters, allowing for faster zooming operations focused on the cluster siblings, rather than the dendrogram hierarchy. Nevertheless, users can choose the zooming approach that is most adapted to their needs.



a. Zooming sequentially according to dendrogram structure



b. Zooming by grouping sibling clusters

Fig 8. Hierarchical zooming approaches: *sequential* (a) and *by grouping* (b)

```
Algorithm: Zooming_Out
Input:
    current_clusters: list              // clusters displayed on the screen
    doc_id_index: hashTable             // documents and their IDs
    cluster_topic_index: hashTable      // cluster topics
    zoom_type: string                   // sequential or grouping
Variables:
    zoom_seq: list                      // clusters to zoom sequentially
    max_depth: int                      // max depth of current clusters
Output:
    current_clusters: list              // updated clusters displayed on screen
Begin
  If zoom_type == grouping
    If length(current_clusters) != 1
        For every cluster_j in current_clusters
            sibling= getSibling(cluster_j)
            If sibling in current_clusters Then
                append getParent(cluster_j) to current_clusters
                remove cluster_j from current_clusters
                remove sibling from current_clusters
        For every cluster_k in current_clusters
            displayCluster (cluster_k)

  Else if zoom_type == sequential
    If length(current_clusters) != 1
        For every cluster_j in current_clusters
            If cluster_j.depth > max_depth
                max_depth = cluster_j.depth
        For every cluster_j in current_clusters
            sibling= getSibling(cluster_j)
            If sibling in current_clusters Then
                append getParent(cluster_j) to zoom_seq
        next_parent= getLowestParent (zoom_seq)
        append next_parent to current_clusters
        remove cluster_j from current_clusters
        remove sibling from current_clusters
        For every cluster_k in current_clusters
            displayCluster (cluster_k)
End
```

a. Zooming-out algorithm

```
Algorithm: Zooming_In
Input:
    current_clusters: list
    doc_id_index: HashTable
    cluster_topic_index: HashTable
    zoom_type: string
Variables:
    max_depth: int      // max depth of current clusters
Output:
    current_clusters: list
Begin
  If zoom_type == grouping
    If length(current_clusters) != rows(doc_id_index)
        For every cluster_j in current_clusters
            append(cluster_j+.0) to current_clusters
            append(cluster_j+.1) to current_clusters
            remove cluster_j from current_clusters
    For every cluster_k in current_clusters
        displayCluster (cluster_k)

  Else if zoom_type == sequential
    If length(current_clusters) != rows(doc_id_index)
        For every cluster_j in current_clusters
            If cluster_j.depth > max_depth
                max_depth = cluster_j.depth
        next_parent= getLowestParent (zoom_seq)
        append(next_parent +.0) to current_clusters
        append(next_parent +.1) to current_clusters
        remove next_parent from current_clusters
        For every cluster_k in current_clusters
            displayCluster (cluster_k)
End
```

b. Zooming-in algorithm

Fig 9. Pseudocodes of our zooming algorithms

The pseudocodes for our zooming algorithms are shown in Fig. 9. They accept as input the index structure produced by the indexing component, the list of clusters currently displayed on the screen, and the zoom type (i.e., sequential or by grouping). The algorithms are invoked following mouse-scroll events. In case of zooming out while grouping, the algorithm will determine the sibling IDs of all the currently displayed clusters (cf. Fig. 9.a, line 4). It will then check whether these siblings exist

on the screen and belong to the current clusters list (line 5). If a pair of siblings exists in the current cluster list, it is grouped. All sibling pairs are grouped at once to zoom out. A similar logic is applied for zooming-in with groups (cf. Fig. 9.b), where the IDs of the children are generated and those new children will be displayed. For zooming out sequentially, the algorithm will start by determining the set of currently displayed clusters with the highest depth value (Fig. 9.a, lines 13-15). These clusters are candidates for zooming. Next, the algorithm checks whether these clusters currently have their siblings on the screen. After determining the existing siblings, the algorithm compares the IDs of the potential parents to determine the parent that is lowest on the dendrogram (Fig. 9.a, line 20). The corresponding sibling pair is grouped. Similarly, when zooming in sequentially (Fig. 9.b), the cluster with the most depth and the lowest ID is split into its child clusters to zoom in.

4 Experimental Evaluation

4.1 Prototype Implementation

We have implemented our indexing component using the Python programming language. We perform i) data serialization using *PDFToText*, ii) text preprocessing and feature extraction using *NLTK*, iii) clustering and dendrogram building using *SciPy*. We store the index tables in SQLite, a lightweight relational database commonly used for on-disk data storage. We have implemented our visualization component using JavaScript, creating the visualizations using D3.js, a popular visualization framework designed for efficiently handling large datasets. Our implementation and test data are available online on github[1].

4.2 Experimental Protocol

4.2.1 Performance Experiments

We conducted experiments to assess both our indexing and visualization components.

Indexing component: we evaluate offline indexing by measuring: i) index *building time*: the time to preprocess, cluster, extract the topics, build and update the index structure, while varying the size of the dataset; ii) index *size in-memory*: the size of the index tables stored in-memory, while varying the size of the dataset.

Visualization component: we evaluate the online visualization by measuring: i) cluster *visualization time*: the time needed to load the interface and document clusters, while varying the size of the input index; ii) cluster *zooming time*: the time needed to perform the zooming action and load the zoomed cluster visualizations, while varying the size of the index.

4.2.2 Qualitative Experiments

We created an online survey[2] to evaluate the quality of our solution. We considered five evaluation criteria to evaluate the overall tool's usability, including: i) *stability*,

[1] https://github.com/HalaSaadeh/LAU-ECE-Research-cluster-indexing and https://github.com/HalaSaadeh/LAU-ECE-Research-cluster-visualization

[2] Available at: https://forms.gle/gugpZcM9jyzrvuPG9

ii) *look and feel*, iii) *ease of use*, iv) *responsiveness*, and v) *user interface* (cf. Table 3) We considered four additional criteria to evaluate the usefulness of key functionalities, including: i) *zooming intuitiveness*, ii) *topic descriptiveness*, iii) *treemap exploration intuitiveness* and iv) *circle packing exploration intuitiveness* (cf. Table 4). We also requested from the testers to perform a set of predefined tasks and asked them to rate their satisfaction accordingly: *Task 1:* Find a document and a cluster where the document does not belong; *Task 2:* Find a cluster that has a weak relationship with other clusters; *Task 3:* Find two clusters that have a strong relationship with each other; *Task 4:* Find two documents that have similar topics; *Task 5:* Find two documents that have distinct topics. Testers were instructed to separately rate every evaluation criterion and every task on an integer scale from 0 to 4 (i.e., from *highly dissatisfied* to *highly satisfied*).

Table 3. Overall tool's usability criteria

Criterion	Description	Evaluation question
Stability	It is the ability of the software tool to function over a long period of time without crashing.	Given the criterion's description, how satisfied are you with the stability of the software tool?
Look and Feel	It refers to the first impression a user has after using the software tool.	Given the criterion's description, how satisfied are you with the look and feel of the software tool?
Ease of Use	It describes how easy and straightforward it is to use and manipulate the software tool.	Given the criterion's description, how satisfied are you with the ease of use of software tool?
Responsiveness	It refers to the time it takes the software tool to perform a certain action or behavior such as the cluster zooming.	Given the criterion's description, how satisfied are you with the responsiveness and overall speed of this application functionalities?
User Interface	It is the means through which a user controls a software application and interacts with it.	Given the criterion's description, how satisfied are you with the interface of this software tool?

Table 4. Key functionalities' usefulness criteria

Criterion	Description	Evaluation question
Zooming Intuitiveness	It reflects how easy it is for a user to zoom in and out of the document clusters using the tool	Given the criterion's description, how satisfied are you with the tool's zooming functionality?
Topic Descriptiveness	It reflects how easy it is for a user to understand the topic associated with a document or cluster.	Given the criterion's description, how satisfied are you with the topics generated by this tool?
Treemap exploration	It refers to the visualization exploration of the document clusters using treemap.	How satisfied are you with the visualizing the clusters using treemap?
Circle packing exploration	It refers to the visualization fexploration of the document clusters using circle packing	How satisfied are you with the visualization of the clusters using circle packing?

4.3 Experimental Results

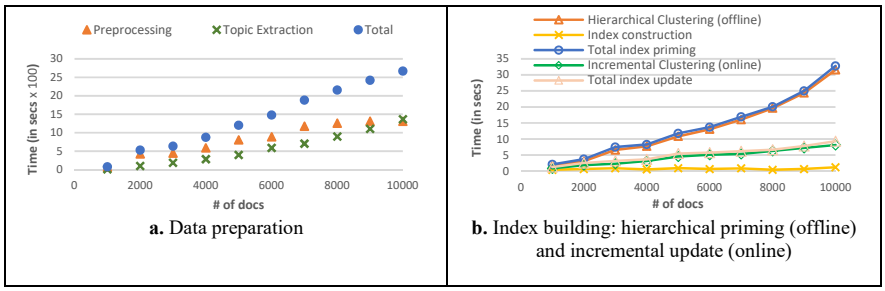
4.3.1 Performance Experiments

The time complexity of our offline hierarchical clustering and index priming simplifies to $O(N^2)$ time where N represents the number of documents. The complexity of our online incremental clustering and index update comes down to $O(N \times C)$ where C represents the number of clusters, and it simplifies to $O(N)$ since C is usually $<< N$. Space complexity also simplifies to $O(N)$, and comes down to the size of the *cluster-document index* table.

We conducted our performance experiments on a set of United Nations (UN) documents acquired from the UN's Manara portal[1], by varying the dataset size from

[1] https://manara.unescwa.org/home

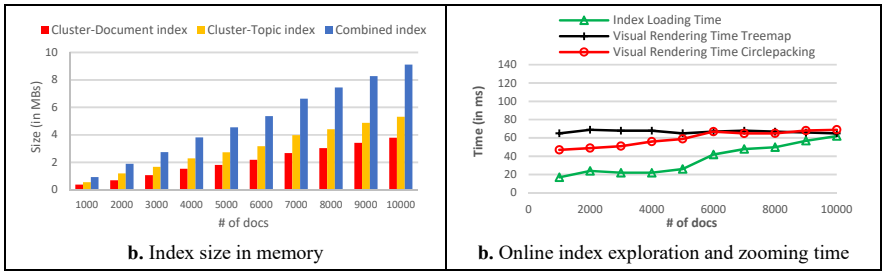
1000-to-10,000 documents, with document size averaging 2,630 KB per document. Tests were conducted on a personal computer with Intel(R) Core(TM) i7-8750H processor with 2.4 GHz processing speed and 16 GB of RAM. Results in Fig. 10.a show almost linear time for data preparation. Results in Fig. 10.b show polynomial time for offline index priming which is mainly due to the quadratic time of the hierarchical clustering process, and linear time for online index update following the linear time performance of incremental clustering. Results in Fig. 11.a show that index size increases linearly with the number of documents in the dataset. On the one hand, index priming is conducted offline and will not affect the user's experience. On the other hand, index update occurs online and is seamless w.r.t. user exploration given i) the efficiency of our incremental clustering process and most importantly ii) the limited number of documents being explored online and being added to the index (e.g., while our time experiments in Fig. 10.b and Fig. 11.b varies the number of documents in the thousands, yet a typical user usually explores tens of documents at a time and those are incrementally integrated into the index almost instantaneously). Results in Fig. 11.b highlight our solution's logarithmic (almost constant) time in loading the clusters and performing cluster zooming.



a. Data preparation

b. Index building: hierarchical priming (offline) and incremental update (online)

Fig 10. Data preparation and index build time



b. Index size in memory

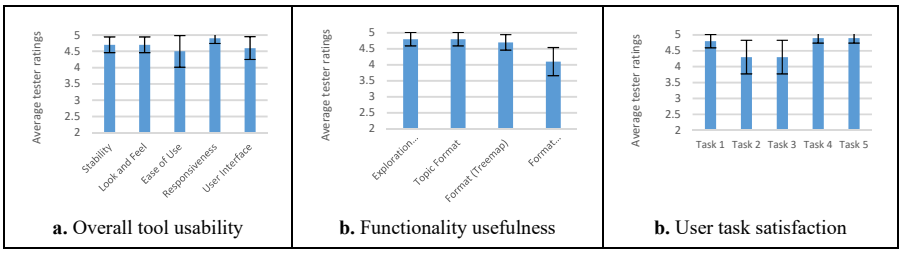
b. Online index exploration and zooming time

Fig 11. Index size in memory (a) and online exploration and zooming time (b)

4.3.2 Qualitative Experiments

A total of 20 testers (senior engineering students) were invited to contribute to the experiment. Testers were invited to run the tool using an executable version shared online, presented with the same use case consisting of a dataset of 50 documents representing the UN's Sustainable Development Goals (SDGs) extended from our

running example[1]. They were subsequently asked to fill the online survey criteria and task ratings. Results in Fig. 12, aggregated for every criterion and every task, show that most testers are satisfied with the tool's usability and the usefulness of its cluster exploration and zooming functionalities, producing overall average ratings of 4.68/5 and 4.60/5 considering the combined usability and usefulness criteria respectively. Results also show that users were mostly satisfied with the assigned tasks, producing an overall average 4.64/5 rating.



a. Overall tool usability **b.** Functionality usefulness **b.** User task satisfaction

Fig 12. Average ratings for tool quality, functionality usefulness, and user task satisfaction

5 Conclusion

This study introduces a hierarchical indexing solution for dynamic and interactive zooming of document clusters. Different from existing solutions which do not address the issue of cluster formation and do not allow dynamic exploration and zooming, our solution i) builds a novel cluster index structure adaptively using hierarchical (offline) clustering for index priming; and incremental (online) clustering for index update, ii) allows interactive cluster exploration online, where iii) the index is designed for seamless zooming in and out of the clusters. The design is modular and allows decoupling the indexing and the visualization as separate services. Empirical results highlight the efficiency and practicality of the solution. We are currently building a querying layer on top of the indexing and visualization components. We are also extending our TF-IDF text features to consider cluster-based features using term-cluster relationships [2]. Extending TF-IDF to consider supervised learning schemes [3] and machine learning (ML) based techniques such as BERT, transformers, and Large Language Models (LLMs) is another direction [33]. We also plan to investigate different clustering algorithms (e.g., constrained agglomerative and spectral [16, 29]), to allow speed-ups and reduced index building time.

References

[1] Ahmad A. and Khan S., *Survey of State-of-the-Art Mixed Data Clustering Algorithms.* IEEE Access 2019. 7: 31883-31902.

[2] Attieh J. and Tekli J., *Supervised Term-Category Feature Weighting for Improved Text Classification.* Knowledge Based Systems 2023. 261:110215.

[3] Attieh J. and Tekli J., *Fast Text Classification using Lean Gradient Descent Feed Forward Neural Network for Category Feature Augmentation.* TrustCom'23 conf., 2023. 2341-2348.

[4] Castano S., et al., *Thematic Exploration of Linked Data.* VLDS'21 workshop, 2011, 11-16.

[1] https://www.unescwa.org/ehandbook-sdg-framework-metadata

[5] Castano S., et al., *Structured Sata Clouding across Multiple Webs.* Information Systems 2012. 37(4): 352-371.
[6] Castano S., et al., *inWalk: Interactive and Thematic Walks inside the Web of Data.* Inter. Conf. on Extended DataBase Technology (EDBT'14), 2014. pp. 628-631.
[7] Castano S., et al., *Exploratory Analysis of Textual Data Streams.* Future Generation Computer Systems, 2017. 68: 391-406.
[8] Castano S., et al., *Topic Summary Views for Exploration of Large Scholarly Datasets.* Journal of Data Semantics, 2018. 7(3): 155-170.
[9] Di Gennaro G., et al., *Considerations about learning Word2Vec.* Journal of Supercomputing, 2021. 77(11): 12320-12335.
[10] Ferrara A., et al., *Dimensional Clustering of Linked Data: Techniques and Applications.* Transactions on Large Scale Data and Knowledge Centered Systems, 2015. 19: 55-86
[11] Fraigniaud P. and Korman A., *An Optimal Ancestry Labeling Scheme with Applications to XML Trees and Universal Posets.* Journal of ACM, 2016. 63(1): 6:1-6:31.
[12] Ganesan A., et al., *LDAExplore: Visualizing Topic Models Generated Using Latent Dirichlet Allocation.* 2015. CoRR abs/1507.06593.
[13] Gower J. and Ross G., *Minimum Spanning Trees and Single Linkage Cluster Analysis.* Applied Statistics, 18, 1969. pp. 54-64.
[14] Halkidi M. et al., *Clustering Algorithms and Validity Measures.* SSDBM'21, 2001, 3-22.
[15] Haraty R. and Nasrallah R., *Indexing Arabic Texts using Association Rule Data Mining.* Library Hi Tech, 2019. 37(1): 101-117.
[16] Haraty R. and Assaf A., *DG-means: a superior greedy algorithm for clustering distributed data.* Journal of Supercomputing, 2024. 80(2): 1990-2024.
[17] Liu J., et al., *Efficient Labeling Scheme for Dynamic XML Trees.* Info. Sc., 2013. 221: 338-354.
[18] Lü H. and Fogarty J., *Cascaded Treemaps: Examining the Visibility and Stability of Structure in Treemaps* Graphics Interface, 2008. pp. 259-266.
[19] Lu J., et al., *Indexing and Querying XML using Extended Dewey Labeling Scheme.* Data and Knowledge Engineering (DKE), 2011. 10(1):35-59.
[20] Miller G. and Fellbaum C., *WordNet Then and Now.* Lang. Res. & Eval., 2007. 41(2): 209-214.
[21] Muelder C. and Ma K., *A Treemap Based Method for Rapid Layout of Large Graphs.* IEEE Pacific Visualization Symposium (PacificVis'08), 2008. pp. 231-238.
[22] Qin W., et al., *Text and metadata extraction from scanned Arabic documents using support vector machines.* Journal of Information Sciences, 2022. 48(2): 268-279.
[23] Rajendran T. and Gnanasekaran T., *Multi-level Object Relational Similarity based Image Mining for Improved Image Search using Semantic Ontology.* Cluster Computing, 2019. 22: 3115-3122.
[24] Salameh K., et al., *Unsupervised Knowledge Representation of Panoramic Dental X-ray Images using SVG Image-and-Object Clustering.* Multimedia Systems, 29(4): 2293-2322, 2023.
[25] Sarkissian S. and Tekli J., *Unsupervised Topical Organization of Documents using Corpus-based Text Analysis.* Inter. MEDES'21 conf., 2021. pp. 87-94.
[26] Schaffer D et al., *Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks.* Graphics Interface, 1993. 12 p.
[27] Takama Y., et al., *Treemap-Based Cluster Visualization and its Application to Text Data Analysis.* Journal of Adv. Comput. Intell. Intell. Informatics, 2021. 25(4): 498-507.
[28] Tatarinov I., et al., *Storing and Querying Ordered XML using a Relational Database System.* Inter. ACM SIGMOD Conference, 2002. pp. 204-215.
[29] Tekli J., et al., *(k, l)-Clustering for Transactional Data Streams Anonymization.* Information Security Practice and Experience, 2018. pp. 544-556.
[30] Tekli J., *An Overview of Cluster-based Image Search Result Organization: Background, Techniques, and Ongoing Challenges.* Knowl. Inf. Syst., 2022. 64(3): 589-642.
[31] Tu N.A., et al., *Featured Correspondence Topic Model for Semantic Search on Social Image Collections.* Expert Systems Applications, 2017. 77: 20-33.
[32] Willett P., *The Porter Stemming Algorithm: Then and Now.* Program, 2006. 40(3): 219-223.
[33] Zhao W., et al., *Dense text retrieval based on pretrained language models: A survey.* ACM Transactions on Information Systems (TOIS), 2024. 42(4):1–60.