University of Crete

Computer Science Department

Results Clustering in Web Searching

Kopidaki Styliani

Master's Thesis

Heraklion, June 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ομαδοποίηση Αποτελεσμάτων στις Μηχανές Αναζήτησης του Ιστού

Εργασία που υποβλήθηκε από την

Στυλιανή Εμμ. Κοπιδάκη

ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Στυλιανή Κοπιδάκη, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Ιωάννης Τζίτζικας, Επίκουρος καθηγητής, Επόπτης

Δημήτρης Πλεξουσάκης, Καθηγητής, Μέλος

Γρηγόρης Αντωνίου, Καθηγητής, Μέλος

Δεκτή:

Πάνος Τραχανιάς, Καθηγητής

Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Ιούνιος 2009

Results Clustering in Web Searching

Kopidaki Styliani

Master's Thesis

Computer Science Department, University of Crete

Abstract

This thesis elaborates on the problem of providing efficient and effective methods for results clustering in Web searching. In brief, results clustering is useful for providing users with overviews of the search results and thus allowing them to restrict their focus to the desired parts of the returned answer. In addition, results clustering alleviates the problem of ambiguity of natural language words. However, the task of deriving (single-word or multiple-word) names for the clusters (usually referred as *cluster labeling*) is a difficult task, because they have to be syntactically correct and predictive (should allow users to predict the contents of each cluster). Furthermore, results clustering is an online task therefore efficiency is an important requirement.

This thesis surveys the methods that have been proposed and used for results clustering and focuses on the *Suffix Tree Clustering (STC)* approach. STC is a clustering technique where search results (mainly snippets) can be clustered fast (in linear time), incrementally, and each cluster is labeled with a phrase. This thesis proposes two novel results clustering methods: (a) a variation of the STC, called STC+, with a scoring formula that favors phrases that occur in document titles and differs in the way base clusters are merged, and (b) a novel non merging algorithm, called NM-STC that results in hierarchically organized clusters. The comparative user evaluation showed that both STC+ and NM-STC are significantly more preferred than STC, and that NM-STC is about two times faster than STC and STC+. These methods where applied over `Mitos` Web search engine and over Google. Moreover, NM-STC was integrated with the *Dynamic Faceted Taxonomies* interaction scheme of `Mitos`. The dynamic coupling of results clustering with dynamic faceted taxonomies results to an effective, flexible and efficient exploration experience. Finally, the thesis reports experimental and empirical results from applying these methods over `Mitos` and over Google.

Supervisor: Yannis Tzitzikas

Assistant Professor

Ομαδοποίηση Αποτελεσμάτων στις Μηχανές Αναζήτησης του Ιστού

Κοπιδάκη Στυλιανή
Μεταπτυχιακή Εργασία
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Η συγκεκριμένη εργασία ασχολείται διεξοδικά με το πρόβλημα εύρεσης αποτελεσματικών και αποδοτικών μεθόδων για την ομαδοποίηση αποτελεσμάτων στις μηχανές αναζήτησης του Ιστού. Εν συντομία, η ομαδοποίηση αποτελεσμάτων παρέχει στους χρήστες μια γενική επισκόπηση των αποτελεσμάτων αναζήτησης, επιτρέποντας τους να εστιάσουν την προσοχή τους σε εκείνα τα τμήματα της απάντησης που ανταποκρίνονται στις πληροφοριακές τους ανάγκες. Επιπλέον, η ομαδοποίηση αποτελεσμάτων μετριάζει το πρόβλημα της αμφισημίας των λέξεων της φυσικής γλώσσας. Εντούτοις, η σύνταξη (μονολεκτικών ή περιφραστικών) ονομάτων παρουσίασης για τις παραγόμενες ομάδες είναι ένα δύσκολο πρόβλημα αφού τα ονόματα πρέπει αφενός να είναι συντακτικά ορθά και αφετέρου να επιτρέπουν στο χρήστη να προβλέψει τα περιεχόμενα των ομάδων. Συνάμα, η ομαδοποίηση αποτελεσμάτων αποτελεί τμήμα της διαδικασίας απάντησης επερωτήσεων επομένως η αποδοτικότητα είναι μια σημαντική απαίτηση.

Η εργασία αυτή κάνει μια επισκόπηση των μεθόδων που έχουν προταθεί και έχουν χρησιμοποιηθεί για την ομαδοποίηση αποτελεσμάτων και εστιάζει στον αλγόριθμο *Suffix Tree Clustering (STC)*. Ο STC είναι μια τεχνική στην οποία τα αποτελέσματα αναζήτησης (κυρίως τμήματα των κειμένων) ομαδοποιούνται γρήγορα (σε γραμμικό χρόνο), αυξητικά, και η κάθε ομάδα έχει μια φράση σαν όνομα. Η εργασία αυτή προτείνει δύο νέες μέθοδους: (α) μια παραλλαγή του STC, που λέγεται STC+, η οποία χρησιμοποιεί μια συνάρτηση βαθμολόγησης που ευνοεί τις φράσεις που εμφανίζονται στους τίτλους των εγγράφων και διαφέρει στον τρόπο με τον οποίο συγχωνεύονται οι υποψήφιες ομάδες, και (β) ένα νέο αλγόριθμο, που λέγεται NM-STC, ο οποίος καταλήγει σε μια ιεραρχική δομή από ομάδες. Η συγκριτική αξιολόγηση με χρήστες έδειξε ότι οι χρήστες προτιμούν περισσότερο τους STC+ και NM-STC από τον STC, και ότι ο NM-STC είναι δύο φορές πιο γρήγορος από τους STC και STC+. Οι μέθοδοι αυτοί εφαρμόστηκαν πάνω στη μηχανή αναζήτησης Μίτος και το Google. Επιπλέον, τα αποτελέσματα του NM-STC ενσωματώθηκαν στο μοντέλο αλληλεπίδρασης των Δυναμικών Πολυδιάστατων Ταξινομιών που υποστηρίζει η μηχανή Μίτος, ως μια επιπλέον διάσταση που συμπληρώνει τις

υπόλοιπες διαστάσεις (που είναι ανεξάρτητες περιεχομένου). Η ζεύξη αυτή έχει σαν αποτέλεσμα μια αποτελεσματική, ευέλικτη και αποδοτική πλοηγητική εμπειρία. Τέλος, περιγράφονται και αναλύονται τα πειραματικά και εμπειρικά αποτελέσματα από την εφαρμογή αυτών των μεθόδων πάνω στη μηχανή Μίτος και στο Google.

Επόπτης Καθηγητής: Γιάννης Τζίτζικας
Επίκουρος Καθηγητής

Ευχαριστίες

Θα ήθελα να ευχαριστήσω πάρα πολύ τον επόπτη καθηγητή μου κ. Γιάννη Τζίτζικα για όλη τη βοήθεια του για την περάτωση αυτής της εργασίας τον τελευταίο ενάμιση χρόνο, καθώς και για την καθοδήγηση του ως συμβούλου μου τον πρώτο ένα χρόνο των μεταπτυχιακών μου σπουδών.

Επίσης, οφείλω ένα μεγάλο ευχαριστώ στον Παναγιώτη Παπαδάκο για τις πολύτιμες γνώσεις του πάνω στη μηχανή Mίτος και για όλη τη βοήθεια και την υποστήριξη του.

Ακόμα, θα ήθελα να ευχαριστήσω τους γονείς μου, Μανώλη και Μαρία, την αδερφή μου, Νίτσα, για την αγάπη τους και την υποστήριξη τους, τις φίλες και τους φίλους μου που με αντέχουν ακόμα και όσους ήταν εδώ και όσους ήταν μακριά.

Τέλος, θα ήθελα να ευχαριστήσω τους μεταπτυχιακούς φοιτητές Γιάννη Μαρκετάκη, Μύρο Παπαδάκη, Νίκο Αρμενατζόγλου, Σοφία Κλεισαρχάκη, Μαρία Καμπουράκη, Μαρία Ψαράκη, Τσιαλιαμάνη Πέτρο, και τους διδακτορικούς φοιτητές Παναγιώτη Παπαδάκο, Γιάννη Θεοχάρη, Αντώνη Μπικάκη και Θοδωρή Πάτκο για τη συμμετοχή τους στην αξιολόγηση των αλγορίθμων.

Η εργασία αυτή είναι αφιερωμένη στους προπαπούδες μου Νίκο και Χατζήνα (Αναστασία).

Άνθρωπος που παινεύεται
για την πολλή του γνώση
δε ξέρει ο θεός
τι έχει να του δώσει.

Contents

Table of Contents **iv**

List of Figures **viii**

1 Introduction **1**

1.1 Introduction to Clustering 1

1.1.1 Flat (Partitioning) clustering 2

1.1.2 Hierarchical clustering 3

1.2 Clustering in Information Retrieval 4

1.2.1 Information Retrieval Systems 5

1.3 Distance/Similarity Functions 7

1.3.1 Euclidean distance measures (L_p-Norms) 7

1.3.2 Non-Euclidean distance measures (Similarity Measures) 8

1.4 Results Clustering 9

1.4.1 Motivation 9

1.4.2 Approaches and Problems 10

1.4.3 General Requirements 11

1.5 Contribution of this thesis 12

1.6 Organization of the thesis 13

2 Related Work **15**

2.1 The Results Clustering Process 15

2.2 Web Meta-Search Engines 16

2.2.1 Commercial Engines 16

2.2.2 Research Prototypes . . . 18
2.2.3 Open-source Systems . . . 21
2.3 Index-based approaches . . . 23
2.3.1 Lexical Affinities . . . 24
2.3.2 Frequent Itemset Hierarchical Clustering (FIHC) . . . 25
2.3.3 SCuBA - Subspace Clustering . . . 26
2.4 Snippet-based approaches . . . 27
2.4.1 Suffix Tree Clustering (STC) . . . 27
2.4.2 TermRank algorithm . . . 29
2.4.3 Deep Classifier . . . 31
2.4.4 Salient phrases extraction . . . 32
2.4.5 Automatic construction from text information bases . . . 33
2.5 STC-based approaches . . . 35
2.5.1 STC based on True Common Phrase Label Discovery . . . 35
2.5.2 STC with X-gram . . . 36
2.5.3 Extended STC (ESTC) . . . 38
2.5.4 Findex . . . 39
2.5.5 Link-based Clustering . . . 40
2.5.6 Semantic, Hierarchical, Online Clustering (SHOC) . . . 41
2.6 Synopsis and Comparison . . . 42
2.6.1 Discussion . . . 46

3 Our Approach **51**
3.1 Problem Statement and Notations . . . 51
3.1.1 Configuration Parameters . . . 52
3.1.2 Notations . . . 52
3.2 STC and Extensions . . . 53
3.2.1 The Original STC . . . 53
3.2.2 STC+: A Variation of STC . . . 55
3.2.3 A New Clustering Algorithm (NM-STC) . . . 58
3.2.4 Notes . . . 61

3.3 Comparative Evaluation . . . 62
3.3.1 Efficiency . . . 62
3.3.1.1 Time Performance . . . 62
3.3.2 Effectiveness - Quality . . . 62
3.3.2.1 UI Examples . . . 62
3.3.2.2 Evaluation by Users . . . 63
3.3.2.3 Clustering Evaluation Metrics . . . 67
3.4 Synopsis . . . 68

4 Implementation and Applications **71**
4.1 Application over a Web Search Engine . . . 71
4.1.1 Software Design Diagrams . . . 73
4.2 Snippet-based Clustering Component . . . 74
4.2.1 Sequence Diagrams . . . 74
4.3 Implementation of STC . . . 78
4.4 Preprocessing . . . 89
4.4.1 Problems in Detecting the Right Sentence Boundaries . . . 91
4.5 Combining Results Clustering with Metadata Exploratory through Dynamic Taxonomies . . . 92
4.5.1 On-Demand Integration . . . 95
4.5.2 Application over `Mitos` . . . 96
4.5.3 Incremental Evaluation Algorithm . . . 96
4.5.3.1 Using the initial suffix tree . . . 99
4.5.3.2 Using the pruned suffix tree . . . 100
4.5.4 Experimental Results . . . 101
4.5.4.1 Clustering Performance . . . 101
4.5.4.2 Overall Performance . . . 103
4.6 Admin Parameters . . . 105
4.7 Application over Google . . . 106

5 Conclusion **109**
5.1 Synopsis . . . 109

5.2 Directions for further work and research 110

List of Tables

2.1 Snippets set 37
2.2 Features comparison of clustering search engines and algorithms 45
2.3 Pros and cons summary of algorithms and search engines 48
2.4 Complexity comparison of clustering algorithms 48
2.5 Explanation for each parameter of Table 2.4 49

3.1 Base clusters identified from STC using only snippets 57
3.2 Base clusters identified from STC using titles and snippets 57
3.3 Base clusters identified from STC+ 58
3.4 Comparative Evaluation by Users 66
3.5 Relative Ranking by Users 66
3.6 Questionnaire 67
3.7 Evaluation Metrics 67
3.8 Comparative Results 68

4.1 Base clusters identified from Figure's 4.12 suffix tree. 84
4.2 Merged Clusters 86
4.3 Execution times (in seconds) for query q= `kernel` with BT1 and similarity threshold 0.4. 90
4.4 Execution times (in seconds) for query q= `kernel` with BT2 and similarity threshold 0.4. 90
4.5 Differences in number of clusters between the best text approaches and number of common labels. 91
4.6 Top-C Snippet Generation and Clustering Times (in seconds) 103

4.7 Top-C Comparison of Incremental/Non-Incremental Algorithms (in seconds) 105

List of Figures

2.1 Clusty search engine user interface 17
2.2 Quintura search engine user interface 18
2.3 Grouper prototype user interface 19
2.4 Scatter/Gather results for the top 250 documents that contain the word star. 20
2.5 Carrot clustering engine 22
2.6 Snaket clustering engine user interface 24
2.7 The suffix tree of the strings "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too". 28
2.8 Base clusters derived from the suffix tree of Figure 2.7. 28
2.9 The base cluster graph of the example given in Figures 2.7 and 2.8. 29
2.10 A fragment of a relational graph of apple data. 30
2.11 Ranks of terms based on TermRank and TF×IDF. 31
2.12 An example of Salient phrases extraction algorithm 33
2.13 Suffix tree with X-gram 37
2.14 Suffix tree with X-gram 38
2.15 Suffix tree with X-gram after complement 38

3.1 Two examples of NM-STC 60
3.2 Results clustering for the query q=ηράκλειο 63
3.3 Results clustering for the query q=uml 64
3.4 Result Sizes 64
3.5 Evaluation system user interface 65
3.6 Questionnaire 66
3.7 Evaluation per query 69

4.1 The Component Model of `Mitos` . . . 72
4.2 Component diagram of `Mitos` search engine. . . . 73
4.3 Sequence diagram of the results clustering process. . . . 75
4.4 Sequence diagrams for the generation of clusters by the original STC and STC+. . . . 75
4.5 Sequence diagrams for the generation of clusters by the original STC and STC+. . . . 76
4.6 Sequence diagrams for the generation of clusters by the original STC and STC+. . . . 76
4.7 Sequence diagrams for the generation of clusters by NM-STC. . . . 77
4.8 Sequence diagrams for the generation of clusters by NM-STC. . . . 77
4.9 Sequence diagrams for the generation of clusters by NM-STC. . . . 78
4.10 The suffix tree of the Snippets 1, 2 and 3. . . . 80
4.11 Suffix tree pruning example for two nodes a,b. . . . 81
4.12 The pruned suffix tree of Figure 4.10 . . . 81
4.13 Base clusters and final cluster . . . 84
4.14 Faceted Taxonomies interface on `Mitos` . . . 94
4.15 Time to load results to `FleXplorer` . . . 95
4.16 Faceted Taxonomies based on Clustering interface on `Mitos` . . . 97
4.17 Example of using the initial suffix tree . . . 100
4.18 Example of using the pruned suffix tree . . . 101
4.19 Elimination of $A_{c,old}$ . . . 102
4.20 Example of using the pruned suffix tree . . . 102
4.21 Steps (a)-(c) of running scenario . . . 104
4.22 Clustering over Google user interface . . . 107
4.23 Clustering over Google user interface . . . 107

Chapter 1

Introduction

1.1 Introduction to Clustering

Clustering is a process of partitioning a set of objects (or data elements) into subsets, called clusters, such that an object belonging to a cluster is more similar to objects belonging to the same cluster than to objects belonging to other clusters. Partitioning is based on a (dis)similarity measure that is always a pair-wise measure. Clustering is a form of unsupervised learning compared to classification (or categorization) that is based on predefined categories.

Clustering can be separated into various categories. Based on the relation between clusters, there can be either *flat* or *hierarchical* clustering. If Obj denotes the set of objects to be clustered, flat clustering generates a flat set of clusters $C_1, \ldots, C_k$ (where $C_i \subseteq Obj$), that are not related to each other, while hierarchical clustering generates a hierarchy of clusters that are correlated to each other. Furthermore, according to the relationship between objects and clusters clustering can be divided into:

- *Exhaustive:* each object is assigned to at least a cluster (i.e. $\cup_{i=1}^{k} C_i = Obj$), otherwise it is called *non-exhaustive*.

- *Overlapping:* an object can belong to more than one cluster ($\exists i, j$ s.t. $C_i \cap C_j \neq \emptyset$), otherwise it is called *non-overlapping* or *disjoint* clustering.

A clustering $C_1, \ldots, C_k$ of Obj is called *partition* clustering, if it is exhaustive and non-overlapping, i.e. if

$$\cup_{i=1}^{k} C_i = Obj \text{ and } C_i \cap C_j = \emptyset \ \forall i \neq j$$

1.1.1 Flat (Partitioning) clustering

Flat (Partitioning) clustering is linear in the number of objects and requires the number of clusters to be predetermined. The most commonly used flat clustering algorithm is K-means that generates non-overlapping clusters.

- *K-means*

  At first, initial K cluster centers (centroids) are selected randomly. Afterwards, each object is assigned to its most similar centroid. Next, centroid vectors are calculated again. Finally, these two steps are repeated until there are no object movements from one cluster to another or a halting criterion is reached (e.g. max number of iterations).

  Quality of created clusters is significantly influenced by the selection of the initial clusters. As mentioned in [24] a robust method that works well for a large variety of objects distributions is to select i (e.g. i = 10) random vectors for each cluster and use their centroid as the seed for this cluster. Also, there are effective heuristics for seed selection which include (i) excluding outliers[1] from the seed set, (ii) trying out multiple starting points and choosing the clustering with lowest cost, and (iii) obtaining seeds from another method such as hierarchical clustering.

  Regarding time complexity, K-means is linear in all relevant factors: iterations, number of clusters, number of vectors and dimensionality of the space.

- *Bisecting K-means*

  Bisecting K-means is a variant of K-means. At first, all objects are assigned to one cluster. Afterwards, three steps are repeated until the desired number of clusters is reached. These steps are, the selection of the cluster to split, the separation of this cluster into two sub-clusters using basic K-means algorithm and the repetition of the

[1] Outliers are the objects that are far from any other objects and therefore do not fit well into any cluster.

bisecting step for a number of iterations in order to choose the separation that creates the clusters with the highest overall similarity.

Experimental results have proved that Bisecting K-means technique is better than the standard K-means approach and as good or better than the hierarchical approaches [32].

1.1.2 Hierarchical clustering

Hierarchical clustering is quadratic in the number of objects and does not require the number of clusters to be predetermined. The hierarchical methods can be further divided into agglomerative (bottom-up) or divisive (top-down) methods. The hierarchical agglomerative clustering methods are most commonly used.

- *Hierarchical agglomerative (bottom-up) clustering (HAC)*

  At first, each object constitutes a different cluster. Afterwards, the similarity between each pair of clusters is computed. Next, the pair of clusters with the highest (inter-cluster) similarity is detected and these clusters are merged. This process is repeated recursively until there is only one cluster left or a halting condition has been met. Based on the way of selection of clusters to be merged, there are four different approaches, single-link, complete-link, group-average and centroid similarity.

  - In single-link, inter-cluster similarity of the cluster's pair is the similarity of the most similar objects. Thus, two clusters are similar if some pair of objects are similar.
  - In complete-link, inter-cluster similarity of the cluster's pair is the similarity of the most dissimilar objects. Thus, two clusters are similar if every pair of objects is similar.
  - In group-average agglomerative clustering (GAAC), inter-cluster similarity of the cluster's pair is defined as the average of all similarities between objects.
  - In centroid clustering, inter-cluster similarity of the cluster's pair is the similarity of cluster's centroids (equivalent with group-average).

- *Hierarchical divisive (top-down) clustering*

  At first, all objects are assigned to a cluster. Afterwards, the cluster is divided using a flat clustering algorithm, which is linear in the number of objects. This process is repeated recursively for each cluster with cardinality greater than a certain threshold or until each object belongs to its own cluster.

  Top-down clustering has the advantage of being more efficient when it does not generate a complete hierarchy all the way down to individual object leaves. For a fixed number of top levels, using an efficient flat algorithm like K-means, top-down algorithms are linear in the number of documents and clusters. So they run much faster than HAC algorithms, which are at least quadratic.

1.2 Clustering in Information Retrieval

Document clustering is a way of grouping documents with similar content. Documents' similarity can be estimated by adopting a similarity measure (e.g. Cosine similarity) or a distance metric (e.g. Euclidean distance). Note that the measure that is used for evaluating queries (i.e. for identifying and ranking the documents that are relevant to user query) could also be used for the purposes of clustering. There are several applications of clustering in Information Retrieval. Below we describe in brief some of them.

Clustering can be used to **speed up the query evaluation**. The computation of similarity between a query and each document of the collection could be a slow process. A faster query evaluation method that involved clustering is the following. Documents of the collection are clustered and a cluster representative is selected for each cluster. Now, each query is compared only with the cluster representatives and the documents that their representatives are similar to the query are retrieved. The set of cluster representatives is definitely a smaller set than the entire collection, thus query evaluation becomes faster. Correctness of this method is based on the fact that each cluster contains documents that are related.

Another aspect of clustering in information retrieval is search **results clustering**. Search results are referred to the documents retrieved in response to a query. Results clustering groups together similar documents. Instead of presenting a list of the relevant documents,

a more usable and more effective interface, that includes the groups of documents-clusters, is presented. Users choose a cluster and see only the documents that belong to this cluster. This application is especially useful when the query has more than one senses as the information presented can help the user to detect them.

Moreover, clustering can be applied on the entire collection in order to **increase the precision and/or recall** of the search results. The idea is that in the initial set of documents that are similar to the query, some documents that belong to the same cluster are added, even if they have low similarity.

Another application of clustering is to provide an **overview of the contents** of a collection. For example, Google News[2] provides an overview of the recent news. Clustering is repeated frequently so as to incorporate the breaking news into the information presented to the user.

Even Latent Semantic Indexing (LSI) [14, 22] is a kind of clustering. Both LSI and clustering are techniques that reduce dimensionality. LSI is an information retrieval model that uses methods of linear algebra in order to reduce the size of the term-document matrix. LSI copes with the problem of synonymy. Synonymy is used to describe the fact that we can make reference to an object or concept with many ways. In LSI the reduced dimensions characterize the documents by context, so documents that do not share keywords with the query but are relative, can be retrieved. LSI can not be used in dynamic systems like Internet because the insertion of new documents is a hard and expensive task.

1.2.1 Information Retrieval Systems

In information retrieval models, documents are considered as sets or bags (depending on the retrieval model) of terms. These terms are the result of a preprocessing phase. This phase usually comprises the following steps. Each document is lexically analyzed, and the words with low discrimination power (like pronouns, articles), which are called stop-words, are removed. For the remainder words their stem is computed (e.g. by eliminating suffixes). Finally, the words that will be used for creating the index are chosen. Some systems consider that only the nouns are necessary, as they have their own meaning while adjectives, adverbs, and verbs are complementary. Roughly, the logical structure of the index of a collection of n documents D=$\{d_1, d_2, ..., d_n\}$ and a set of t terms K=$\{k_1, k_2, ..., k_t\}$ has the form of a

[2] http://news.google.com/

$n \times t$ matrix.

	k_1	k_2	k_3	...	k_t
d_1	$w_{1,1}$	$w_{2,1}$	$w_{3,1}$	...	$w_{t,1}$
d_2	$w_{1,2}$	$w_{2,2}$	$w_{3,2}$	...	$w_{t,2}$
d_3	$w_{1,3}$	$w_{2,3}$	$w_{3,3}$	...	$w_{t,3}$
...	...	...	...	...	...
d_n	$w_{1,n}$	$w_{2,n}$	$w_{3,n}$	...	$w_{t,n}$

Each document d_j is represented by a vector $d_j = \{w_{1,j}, w_{2,j}, ..., w_{t,j}\}$, where $w_{i,j}$ is the weight of term k_i in document d_j. $w_{i,j}$=0 when the term k_i does not appear in document d_j, otherwise it is a positive number and its value is defined according to the adopted retrieval model. For instance, according to the Boolean retrieval model, $w_{i,j} \in \{0, 1\}$, and when the term k_i appears in the document $w_{i,j}$=1, otherwise $w_{i,j}$=0. According to VSM (Vector Space Model), $w_{i,j} \in [0, 1]$, and its value depends on the number of occurrences of k_i in the document d_j (this is expressed by $freq_{i,j}$), and on the number of documents of the collection that contain term k_i (this is expressed by df_i). Specifically:

$$w_{i,j} = tf_{i,j} * idf_i$$

where $tf_{i,j} = \frac{freq_{i,j}}{max_t\{freq_{t,j}\}}$ is the number of occurrences of term k_i in the document d_j, normalized by the max number of occurrences of a term in the document d_j. $idf_i = log(\frac{N}{df_i})$, is a measure of the discreet ability of a term. N is the total number of documents and df_i is the number of documents that term k_i appears.

TF×IDF is a good weight measure because it favors the terms that appear in a few documents a lot of times, and penalizes the terms that appear a lot of times in many documents.

Each query is represented by a vector q=$(w_{1,q}, w_{2,q}, ..., w_{t,q})$, where $w_{i,q} = tf_{i,q} * idf_i$. A document is relevant if its similarity with the query is positive or over a threshold. The similarity between each document and the query is defined as the cosine similarity.

$$sim(d_j, q) = \cos(\vec{d_j}, \vec{q}) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

The rank of relevant documents is defined by their similarity with the query.

Recall that a clustering algorithm aims at separating a collection of objects into various clusters. The simpler form of this problem is to separate a collection of objects into two

clusters. The information retrieval problem, which deals with the searching of documents that are relevant with a query, could be conceived as a clustering problem aiming at deriving two clusters: one comprising all documents that are relevant with the query, and another one containing all documents that are non-relevant with the query.

1.3 Distance/Similarity Functions

Similarity-measures (dually, distance-measures) are very important in information retrieval and in clustering.

Let X be a set of elements. A function

$$d : X \times X \rightarrow R$$

(where R is the set of real numbers) is called a *metric* if for all x, y, z in X, the following conditions are satisfied:

$$\begin{array}{lcll} d(x,y) & > & 0 & \text{(non-negativity)} \\ d(x,y) & = & 0 \text{ iff } x = y & \text{(identity)} \\ d(x,y) & = & d(y,x) & \text{(symmetry)} \\ d(x,y) & < & d(x,z) + d(z,y) & \text{(triangle inequality)} \end{array}$$

The pair (X, d) is called a metric space. Note that vector spaces are a special case of metric spaces.

We could distinguish two major classes of distance measures:

1. *Euclidean*

   A Euclidean distance is based on the locations of points in a Euclidean space.

2. *Non-Euclidean*

   A Non-Euclidean distance is based on properties of points, but not their "location" in a space.

1.3.1 Euclidean distance measures (L_p-Norms)

Assume two vectors $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$.

- L_1 **or Block or Manhattan distance**: The L_1 or Block distance is calculated by the sum of the differences in each dimension.

$$L_1(x,y) = L_1((x_1,...,x_k),(y_1,...,y_k)) = \sum_{i=1}^{k} |x_i - y_i|$$

- L_2 **or Euclidean distance**: The L_2 distance is calculated by the square root of the sum of the squares of the differences between x and y in each dimension. This is the most common notion of "distance".

$$L_2(x,y) = L_2((x_1,...,x_k),(y_1,...,y_k)) = \sqrt{\sum_{i=1}^{k} |x_i - y_i|^2}$$

- L_∞ **distance**: L_∞ is the maximum of the differences between x and y in any dimension. The maximum is the limit as n goes to ∞ of what you get by taking the n^{th} power of the differences, summing and taking the n^{th} root.

$$L_\infty(x,y) = L_\infty((x_1,...,x_k),(y_1,...,y_k)) = \lim_{n\to\infty} \sqrt[n]{\sum_{i=1}^{k} |x_i - y_i|^n}$$

1.3.2 Non-Euclidean distance measures (Similarity Measures)

Functions of this class origin in measuring similarity between sets based on the intersection of the two sets.

Assume two sets of elements X,Y and two vectors $t_i = (t_{i1},\ldots,t_{ik})$, $t_j = (t_{j1},\ldots,t_{jk})$, where weights t_{ih}, t_{jh} are not binary.

- **Dice's coefficient**: Relates the overlap to the average size of the two sets.

$$DiceSim(X,Y) = \frac{2|X \bigcap Y|}{|X| + |Y|} \tag{1.1}$$

or

$$DiceSim(t_i,t_j) = \frac{2\sum_{h=1}^{k} t_{ih}t_{jh}}{\sum_{h=1}^{k} {t_{ih}}^2 + \sum_{h=1}^{k} {t_{jh}}^2}$$

- **Jaccard's coefficient**: Relates the overlap to the size of the union.

$$JaccardSim(X,Y) = \frac{|X \bigcap Y|}{|X \bigcup Y|} \tag{1.2}$$

or

$$JaccardSim(t_i,t_j) = \frac{\sum_{h=1}^{k} t_{ih}t_{jh}}{\sum_{h=1}^{k} {t_{ih}}^2 + \sum_{h=1}^{k} {t_{jh}}^2 - \sum_{h=1}^{k} t_{ih}t_{jh}}$$

- **Cosine coefficient**: Relates the overlap to the geometric average of the two sets.

$$CosSim(X,Y) = \frac{|X \bigcap Y|}{|X|^{1/2} \times |Y|^{1/2}} \tag{1.3}$$

or

$$CosSim(t_i,t_j) = \frac{\vec{t_i} \bullet \vec{t_j}}{|\vec{t_i}| \ \times \ |\vec{t_j}|} = \frac{\sum_{h=1}^{k} t_{ih}t_{jh}}{\sqrt{\sum_{h=1}^{k} {t_{ih}}^2 \sum_{h=1}^{k} {t_{jh}}^2}}$$

- **Overlap**: Determines to which degree the two sets overlap.

$$Sim(X,Y) = \frac{|X \bigcap Y|}{min(|X|,|Y|)} \tag{1.4}$$

or

$$Sim(t_i,t_j) = \frac{\sum_{h=1}^{k} t_{ih}t_{jh}}{min(\sum_{h=1}^{k} {t_{ih}}^2, \sum_{h=1}^{k} {t_{jh}}^2)}$$

Equations (1.1), (1.2), (1.3), (1.4) are used when the elements of the compared sets are not weighted. Rest of the equations are used when the elements' weights are not binary. For example, in case that similarity between documents must be estimated, t_i, t_j are the document vectors and k is the number of terms.

1.4 Results Clustering

1.4.1 Motivation

In our days, search engines are the most powerful tools for searching and retrieving information from the (constantly-growing) Web. Web search engines (WSEs) typically return a ranked list of documents that are relevant to the submitted query and users have to explore the answer linearly (from the first page to the second, and so on). For each document, its title, URL and a small fragment of the text that contains the searched keywords are usually presented. This fragment of the document, which depends on the query, is called *snippet*. Ranked list presentation and low precision of the results, require from the user to try a lot in order to find the information he needs. It is observed that most users are impatient and look at the first results only. Consequently, when either the documents with the intended (by the user) meaning of the query words are not in the first pages, or there are a few dotted

in various ranks (and probably different result pages), then the user has to try hard to find and collect the information he really wants. The problem becomes harder if the user can not guess additional words for restricting his query, or the additional words he chooses are not the right ones for restricting the result set.

In addition it is difficult for the user to identify the discrete notions of ambiguous queries (e.g. jaguar, apple) and to guess additional discriminative keywords to make the query more specific.

A solution to these problems is *results clustering* which provides a quick overview of the search results. It aims at grouping the results into topics, called *clusters*, with predictive names (labels), aiding the user to locate quickly one or more documents that otherwise he wouldn't practically find especially if these documents are low ranked (and thus not in first result pages).

Currently only a few engines provide result clustering services.

1.4.2 Approaches and Problems

Original versus Snippet-based approaches

Clustering can be applied either to original documents or to snippets. Clustering meta-search engines use the results of one or more search engines (e.g. Google[3], Yahoo![4], Live Search (formerly MSN Search)[5]), in order to increase coverage/relevance. Since different search engines return different search results as it is proved by several research. A recent research study[6] estimated that the percent of total first page results shared by the top four search engines is 0.6%. Web search engines reply to a query by returning a ranked list of snippets. Each web *snippet* is a small summary of the web page contents that contain the search keywords. Clustering the snippets rather than the whole documents makes clustering algorithms faster. Also, the algorithm's speed can be improved even more by processing snippets incrementally; starting from the first snippet that is received rather than processing the snippets altogether.

[3] www.google.com
[4] www.yahoo.com
[5] www.live.com
[6] http://www.infospaceinc.com/onlineprod/Overlap-DifferentEnginesDifferentResults.pdf

Cluster Labeling

Cluster labeling is the task of deriving readable and meaningful (single-word or multiple-word) names for clusters, in order to help the user to recognize the clusters/topics he is interested in. Such labels must be predictive (should allow users to guess the contents of each cluster), descriptive, concise and syntactically correct. So, the user will not look at the typical list of documents that can be very long, but will look only the documents in the topics he is interested in. In general the user can browse the result in a non-linear manner.

Efficiency

Search engines should use efficient and scalable clustering algorithms. Scalability is very important because the number of documents can vary. Usually only the top-L documents are clustered in order clustering to be fast and the resulting labels not to be too general. Some engines, like Clusty and $Carrot^2$, give to the user the option to choose the number of results to be clustered. Clusty by default clusters 200 results but user can change it to 100 or 500 results.

1.4.3 General Requirements

The key requirements of a results clustering algorithm are:

- High intra-cluster similarity

  The produced clusters must consist of relevant documents.

- Concise and Accurate Presentation of each Cluster

  The users should detect quickly what they need.

- Snippet-based

  It should be possible to provide high quality clusters based on document snippet rather than the whole documents.

- Efficient and Progressive Algorithms

  Algorithms must be fast enough, clustering up to one thousand snippets in a few seconds, and incremental, processing each snippet once it is received from the Web.

Some of these are based on [38].

Other desired properties:

- Cluster size uniformity

  Distribution of documents among folders of the same level must be balanced.

- Language-independent

- Not too much overlapping

  Clusters should not overlap too much, on average a search result appears in only 1.2 to 1.5 clusters[7] in order the main distinct themes to be shown.

1.5 Contribution of this thesis

- We provide a detailed survey of the results clustering methods that have been applied or described in the literature.

- We propose two novel results clustering algorithms, called STC+ and NM-STC. STC+ is a variation of the STC, uses a scoring formula that favors phrases that occur in document titles and differs in the way base clusters are merged. NM-STC (Non Merging - STC) is a novel algorithm that adopts a different scoring formula, it does not merge clusters and results in hierarchically organized labels.

- We introduce a new approach for enhancing exploratory web searching with the dynamic coupling of dynamic faceted taxonomies with results clustering.

- We describe an incremental approach of NM-STC that is beneficial for the coupling of dynamic faceted taxonomies with results clustering.

- The results of this thesis have already been applied over `Mitos` Web search engine, and over Google.

- Finally, the results of this thesis will be presented on the 13^{th} European Conference on Digital Libraries [26] and on the 10^{th} International Conference on Web Information Systems Engineering [21].

[7] http://searchdoneright.com/2007/03/how-to-evaluate-a-clustering-search-engine/

1.6 Organization of the thesis

Chapter 2 provides a survey of the algorithms used in document clustering and various search engines that offer on-line results clustering.

Chapter 3 describes two novel results clustering algorithms and their evaluation.

Chapter 4 describes the application of our approaches over `Mitos` search engine, as an independent component of results clustering and as a combination with the component of faceted taxonomies.

Chapter 5 summarizes and identifies topics that are worth of further research.

Chapter 2

Related Work

This chapter examines the state of the art of the algorithms used for web document clustering and provides an overview of the search engines that support results clustering.

It is organized as follows: Section 2.1 describes in brief the results clustering process. Section 2.2 describes some meta-search engines that support results clustering. These engines forward user queries to several other search engines and/or databases, cluster the results obtained by the latter and finally display a topic hierarchy. Section 2.3 analyzes various results clustering algorithms that are based on the vectors of the documents while Section 2.4 analyzes clustering algorithms that are based on the snippets of documents that are returned from the search engines and Section 2.5 describes approaches that are variants of the Suffix Tree Clustering (STC) clustering algorithm. Finally, Section 2.6 presents a comparison of all these clustering approaches.

2.1 The Results Clustering Process

In general, the process of results clustering comprises of the following steps:

(A) Fetch the **representatives** of the top-L documents of the query answer $Ans(q)$.

(B) Construct a vector for each one of them.

(C) Run a clustering algorithm using a specific similarity measure.

(D) Construct and return a tree to present the clusters to the end users where each cluster has been given an appropriate name (or label).

The *representative* of a document d, at Step (A), could be:

- The *snippet* of d as described in Section 1.4.
  We may denote it $snip(d, q)$. Note that we include q because the snippet of a document depends on the submitted query. In case of meta engines, the snippets are provided by the underlying engines. In case of stand alone engines these snippets are computed during query answering.
- The title and URL of d.
- The vector representation of d.
  This is possible for stand alone engines.
- A part of the vector representation of d, e.g. a vector comprising of the F (where $F \leq |K|$, where K is the vocabulary of the collection) biggest coordinates of $\vec{d}$.
  This is possible for stand alone engines.

2.2 Web Meta-Search Engines

2.2.1 Commercial Engines

By definition, a clustering engine analyzes the top (say 200-500) search results from a query and displays the main themes, typically as folders that may consist of subfolders.

- **Vivisimo/Clusty** [6, 2]. Vivisimo is probably the most famous commercial clustering search engine. It calls other search engines like Yahoo! and MSN, extracts the relevant information (titles, URLs, and short descriptions) from the answers retrieved and groups them based on the summarized information. The output is a hierarchical folder structure, allowing users to avoid link overload and to click only on the specific category of information that they need. **Clusty** is an extension of Vivisimo. Figures 2.1(a) and (b) show the interface of Clusty and the clusters derived when submitting the query q= `java` and q= `result clustering algorithms` respectively. Clusty

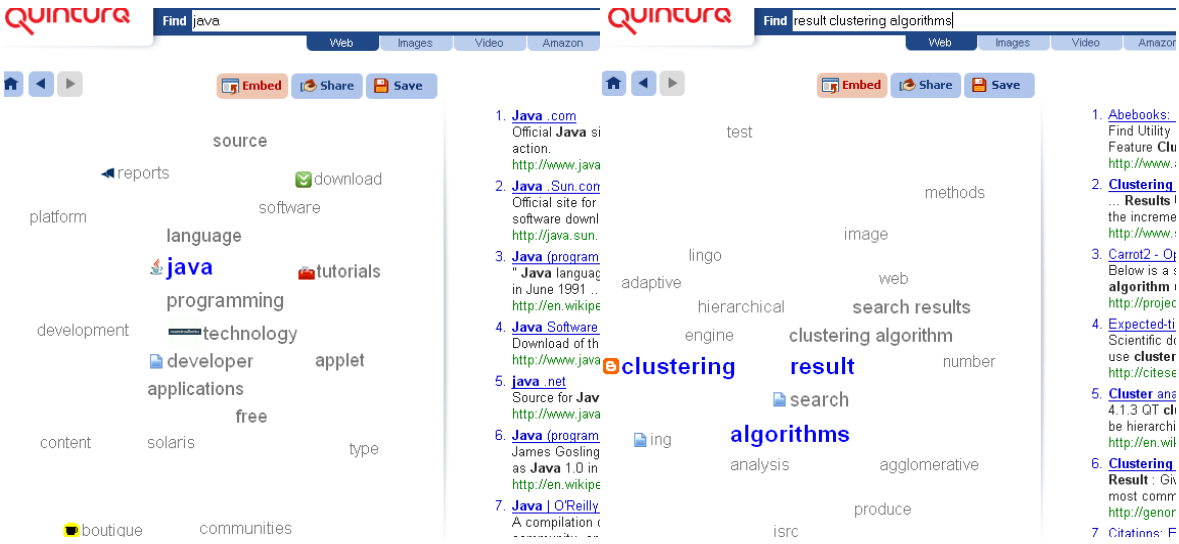
(a) (b)

Figure 2.1: Clusty search engine user interface

offers a new feature, called *remix clustering* that works by clustering again the same search results but ignoring the topics that the user just saw. Hence, it is an interaction model (history-aware) that it is based on/utilizes clustering.

- **Quintura** [4]. Quintura is a visual search engine. It extracts keywords from search results and builds a *word cloud* (visual map). The name of each cluster is placed in a 2D area. The position of the names in the 2D area is based on their distance, while font size is used for indicating the size of the cluster. By clicking words in the cloud, the user query is refined. The user is also able to remove search results i.e. restrict his/her focus by selecting a cluster name and clicking on the *Exclude* icon. Quintura analyzes contextual relationships among keywords, helping to define the context or meaning of the keywords. At present, it builds the map based on information contained in links and summaries of those links returned by the underlying search engines[1]. Figures 2.2 (a) and (b) show the interface of Quintura and the clusters derived when submitting the query q= `java` and q= `result clustering algorithms` respectively.

[1]It is planed to use a Web index of a search engine to allow even faster searches for more relevant results. Also, it is planed to have its own Web index that will be based on contextual relationships among words.



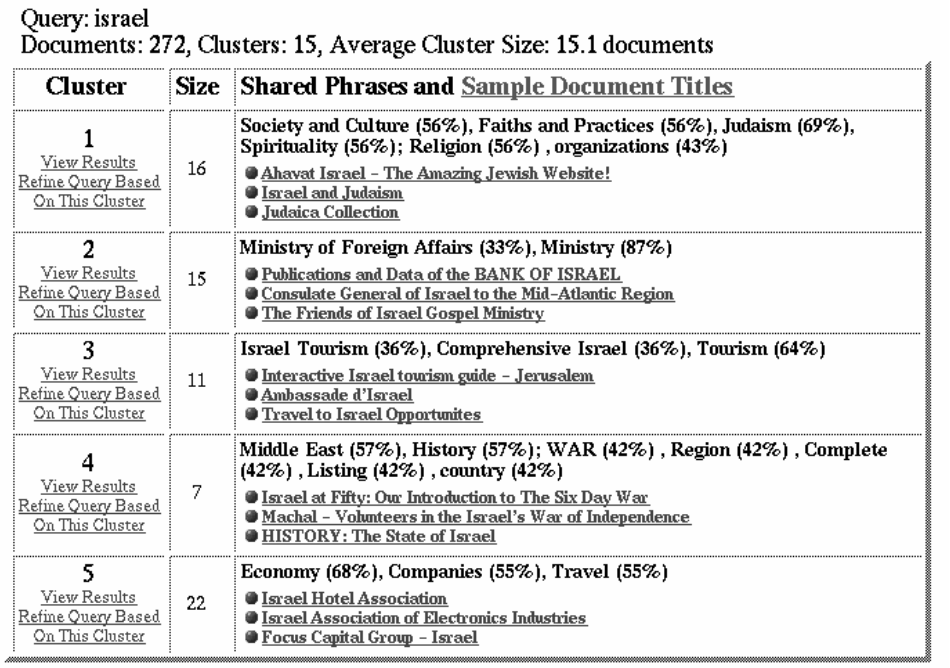
(a) (b)

Figure 2.2: Quintura search engine user interface

2.2.2 Research Prototypes

- **Grouper** [39]. Grouper is an interface for the results of the HuskySearch meta-search engine. Users can specify the number of documents to be retrieved (10-200) from each of the participating search engines. The system queries 10 search engines, so it retrieves 70-1000 documents. Clustering is applied on snippets that are returned by the search engines. Grouper uses the *Suffix Tree Clustering* (STC) algorithm (described in more detail in Section 2.4) to cluster together documents that have common large subphrases. Grouper, in its initial form, is not publicly available but there is *Carrot* which is an open source implementation of it. Figure 2.3 shows a prototype user interface of Grouper and the clusters derived when submitting the query q= `israel`, while Figures 2.5 (a) and (b) show the interface of Carrot and the clusters derived when submitting the query q= `java` and q= `result clustering algorithms` respectively.

- **Scatter/Gather** [11, 18]: Scatter/Gather is a document browsing method that is based on document clustering. At first, Sactter/Gather was applied to large document collections and later it was used for clustering the result set retrieved by any given


Query: israel
Documents: 272, Clusters: 15, Average Cluster Size: 15.1 documents

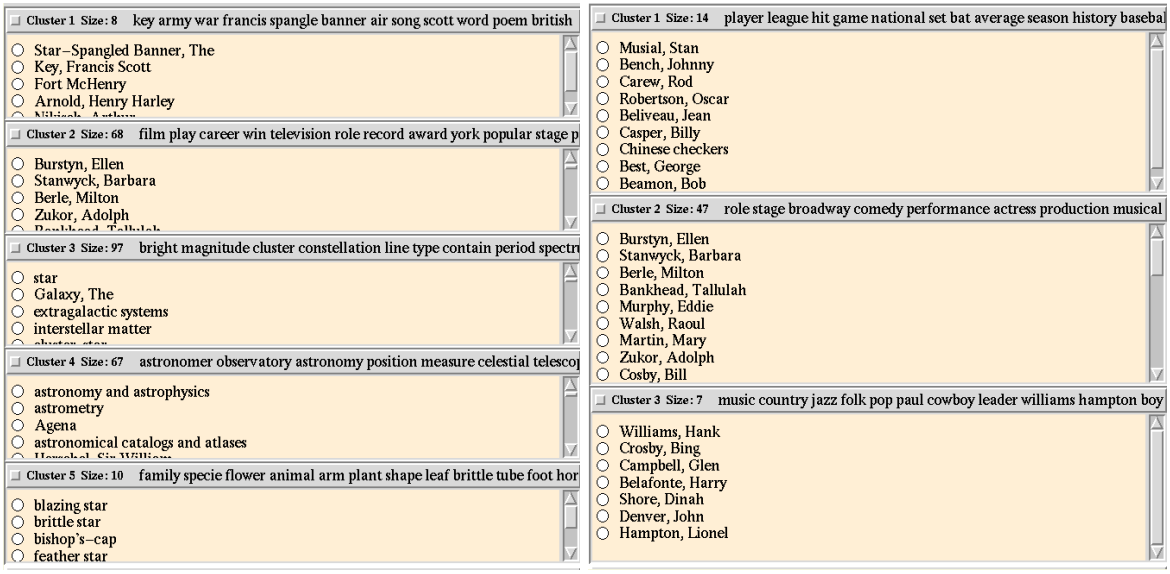
Cluster	Size	Shared Phrases and Sample Document Titles
1 View Results Refine Query Based On This Cluster	16	Society and Culture (56%), Faiths and Practices (56%), Judaism (69%), Spirituality (56%); Religion (56%) , organizations (43%) Ahavat Israel - The Amazing Jewish Website! Israel and Judaism Judaica Collection
2 View Results Refine Query Based On This Cluster	15	Ministry of Foreign Affairs (33%), Ministry (87%) Publications and Data of the BANK OF ISRAEL Consulate General of Israel to the Mid-Atlantic Region The Friends of Israel Gospel Ministry
3 View Results Refine Query Based On This Cluster	11	Israel Tourism (36%), Comprehensive Israel (36%), Tourism (64%) Interactive Israel tourism guide - Jerusalem Ambassade d'Israel Travel to Israel Opportunites
4 View Results Refine Query Based On This Cluster	7	Middle East (57%), History (57%); WAR (42%) , Region (42%) , Complete (42%) , Listing (42%) , country (42%) Israel at Fifty: Our Introduction to The Six Day War Machal - Volunteers in the Israel's War of Independence HISTORY: The State of Israel
5 View Results Refine Query Based On This Cluster	22	Economy (68%), Companies (55%), Travel (55%) Israel Hotel Association Israel Association of Electronics Industries Focus Capital Group - Israel

Figure 2.3: Grouper prototype user interface

search query. In both cases Scatter/Gather's interface remained the same.

Scatter/Gather's interface is interactive. The user can find the information he needs by performing iterative steps. The initial document set is divided into clusters (scatter). Each cluster is presented to the user followed by a number of words that describe its contents and a number of sample documents. The user can select the clusters of his interest. Documents of the selected clusters become the new document set (gather) that is clustered again and is presented to the user. In each iteration, clusters get smaller until individual documents are presented.

Scatter/Gather uses partitional clustering algorithms in order to generate a set of k disjoint documents groups. Partitional clustering constitutes of three steps: finding k centers, assigning each document in the collection to a center and refining of the partition that constructed. *Fractionation* and *Buckshot* were used for the first step. Fractionation, which is used for his accuracy, creates off-line an initial partitioning of the entire set, whereas Buckshot, which is faster, clusters on-the-fly the selected document groups.



(a) (b)

Figure 2.4: Scatter/Gather results for the top 250 documents that contain the word star.

Fractionation clusters n documents into k groups in O(kn) time. It splits document collection into n/m buckets ($m > k$) and clusters each bucket, applying GAC algorithm to each one. These clusters are treated as if they were individuals and the entire process is repeated until only k clusters remain.

Buckshot algorithm combines the determinism and higher reliability of HAC with the efficiency of K-means. First, a small sample of documents of size $\sqrt{kn}$, is randomly selected. Group-average HAC is applied on this sample and the results are used as initial seeds for K-means. Overall algorithm complexity is O(kn) and avoids problems of bad seed selection by employing an HAC algorithm to compute seeds of high quality.

Figures 2.4 (a) and (b) are examples of the Scatter/Gather interface for the top 250 documents that contain the word `star`. Figure 2.4 (a) shows the initial results. Terms of Cluster 1 indicate that this cluster contains documents that involve `stars` as symbols, as in military rank and patriotic songs, while terms of Cluster 2 indicate that it discusses about movies and tv stars. Figure 2.4 (b) shows the results after re-clustering the 68 documents that appear in Cluster 2.

2.2.3 Open-source Systems

- $Carrot^2$ [36, 30, 1]. $Carrot^2$ engine acquires search results from various sources (YahooAPI, GoogleAPI, MSN Search API, eTools Meta Search, Alexa Web Search, PubMed, OpenSearch, Lucene index, SOLR). It supports five different clustering algorithms (*STC, FussyAnts, Lingo, HAOG-STC, Rough k-means*). One of them, Lingo, is the default clustering algorithm used in the $Carrot^2$ live demos. The output is a flat folder structure and the overlapping folders are revealed when the user puts the mouse over a document title. Specifically, all the folders of which the selected document is a member, are marked with a different color, except the selected/current folder.

  Carrot investigated the behavior of STC algorithm for the Polish language. Polish language in comparison with the English is characterized by rich inflection(words have different suffixes depending on their role in a sentence) and the fact that the order of words in a sentence is less strict. Also, they examined the impact on the results from STC merge threshold parameter and a new one, the minimum base cluster score.

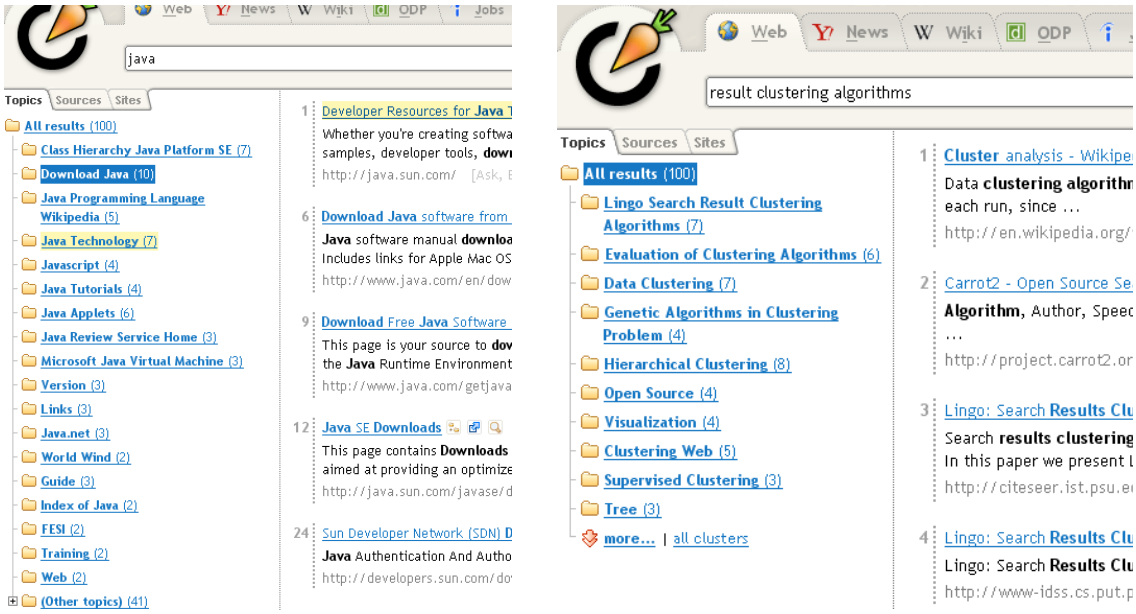
  Base cluster score is calculated with a variation of the original STC formula.

$$s(m) = |m| \times f(|m_p|) \times \sum (tfidf(w_i))$$

  where $|m|$ is the number of terms in phrase m, $f(|m_p|)$ is a function penalizing short-phrases and $tfidf(w_i)$ is a standard Salton's term frequency-inverse document frequency term ranking measure.

  The experiments with the Polish showed that the results have not a significant influence by the merge threshold when the input has been pre-processed. On the contrary, minimum base cluster score threshold seems to influence the number of the derived clusters.

  First in Carrot[36] it is said that the claim of the original paper that the algorithm is not influenced by the preprocessing phase(stemming, stop-words removal) does not stand. In $Carrot^2$[30] it is said that they were expecting a bigger difference in the quality of the results for the Polish. But the combination of stemming and stop words removal did not have the best performance for English and Polish languages. Nevertheless, it was always better compared to results without preprocessing.



(a) (b)

Figure 2.5: Carrot clustering engine

- **SNAKET** (**SN**ippet **A**ggregation for **K**nowledge **E**xtrac**T**ion) [15, 5]. SNAKET engine draws about 200 snippets from 16 search engines about Web, Blog, News and Books domain. This engine offers both hierarchical clustering and folder labeling with variable-length sentences drawn on-the-fly from snippets. SNAKET uses *gapped sentences* as labels, namely sequences of terms occurring not-contiguously into the snippets. Experiments set limit of maximum number of gaps to four. Also, it uses two knowledge bases (the **dmoz**[2] hierarchy and "anchor texts") to improve hierarchy. SNAKET's interface offers the new feature of *personalization* that occurs at the client side. Users can select a set of labels and ask SNAKET to filter out from the ranked list, returned by the queried search engines, the snippets that do not belong to the folders labeled by the selected labels. This approach does not require an explicit login, a pre-compilation of a user profile, or tracking the user's past searches.

  SNAKET's algorithm is composed by the following steps:

[2]http://www.dmoz.org/

– The 200 snippets are processed. They are enriched by querying the anchor-text KB, they are filtered against a stop-list, stemmed and segmented into phrases.

– All pairs of words within a fixed proximity window are extracted and are assigned a score by a function that is based on a TF×IDF measure over dmoz.com categories. Low scored pairs are rejected. The remaining pairs are incrementally merged to form longer gapped sentences. A gapped sentence g is merged with a word pair (w_h, w_k) if they appear in the same snippet and within a proximity window. The score of the longer sentences is calculated again and the process is repeated until no merge is possible or phrases consist of eight words.

– These phrases are the primary labels L_i for the leaves of the folder hierarchy. Each folder C_i contains the snippets that contain L_i. Also, secondary labels S_i are generated for each folder C_i. Secondary labels are gapped sentences that appear on folders over a minimum 80%. The primary label and the secondary labels of a folder constitute the signature of the folder C_i. Based on these signatures, a parent folder is selected for each group of folders that share a gapped sentence. The new parent folders are ranked and low ranked folders are rejected. Afterwards, if two parent folders have the same children folders or the same label then the low ranked folder is discarded. The process is repeated for the remaining parent folders in order to achieve a three level hierarchy.

– The tree hierarchy that has three levels is presented to the user.

Figure 2.12 shows the interface of Snaket and the clusters derived when submitting the query q= `java`.

2.3 Index-based approaches

Traditional clustering algorithms either flat (like K-means and its variants) or hierarchical (agglomerative or divisive) do not require to create snippets. They are based on the vectors of the documents and on a similarity measure. These approaches were further described in Section 1.1.1 and 1.1.2.

Figure 2.6: Snaket clustering engine user interface

2.3.1 Lexical Affinities

A variant of the HAC (Hierarchical Agglomerative Clustering) algorithm for ephemeral clustering is described in [23] which uses "lexical affinities" (pairs of words that appear within a proximity window and are not necessarily contiguous) as indexing units instead of single words.

This approach achieved an O(n^2) complexity for a complete-link HAC using bucket sorting which requires only O(m) steps, rather than O($m$log$m$) steps, to sort m elements, where m is the n(n-1)/2 pairwise similarities between n documents and a linear number of additional steps (where each of them requires only linear time).

In terms of the complete-link method, *similarity value* of a cluster c is defined as the minimum of the pairwise similarities of documents of the cluster c. Also, similarity value of a pair (c_1, c_2) of clusters is the minimum of the pairwise similarities of documents of the union of c_1 and c_2.

For the implementation of the algorithm three basic data structures were used:
1) A "current similarity matrix" which keeps the similarity values for each pair of clusters that can be merged into one (unmarked clusters).

2) Ten buckets, one for each value from 0 to 0.9. Bucket ϑ is a doubly-linked list of distinct unmarked pairs of clusters whose similarity value is in $[\vartheta, \vartheta{+}0.1)$.
3) A "current pointer matrix" where for each pair of distinct unmarked clusters there is a pointer to its position in the doubly-linked list of the appropriate bucket.

The steps of the algorithm are the following:

- Initialization of the data structures. The current similarity matrix is an n $\times$ n matrix where each document consists a cluster. The other structures are initialized accordingly.

- The nonempty bucket with the biggest index ϑ is found, along with the pair (c_1, c_2) at the "top" of the list for this bucket. A new unmarked cluster c_1c_2 is created which contains the members of the union of c_1 and c_2.

- Update of the data structures. In the current similarity matrix the new entry (c_1c_2, c_1c_2) is inserted and (c, c_1c_2) similarities are calculated. Also, the columns and the rows that correspond to c_1 and c_2 are removed. During the computation of similarity values of (c_1c_2, c), the appropriate buckets are added and the current pointer matrix is updated after the insertion of the new entry that corresponds to c_1c_2. Like in the current similarity matrix, the columns and the rows that correspond to c_1 and c_2 are removed from the current pointer matrix.

- Repeat the process until there is only one unmarked cluster left.

2.3.2 Frequent Itemset Hierarchical Clustering (FIHC)

FIHC [16] is a document clustering technique that exploits the notion of frequent itemsets used in data mining. A frequent itemset is a set of words that occur together in some minimum fraction of documents in a cluster. FIHC increases scalability because it reduces dimensions by storing only the frequencies of frequent items that occur in some minimum fraction of documents in document vectors. Also, it has a mechanism that makes clusters disjoint.

The FIHC algorithm can be summarized in three phases:

- *Construct initial clusters:* A cluster is constructed for each global frequent itemset. These clusters are overlapping because a document can contain many global frequent itemsets. In order to make clusters disjoint, each document is assigned only to "the best initial cluster", which is the initial cluster with the highest score. The score function is based on the frequencies of global frequent items of the documents in each cluster.

- *Build a cluster (topic) tree:* In the cluster tree, each cluster (except the root) has exactly one parent. The topic of a parent cluster is more general than the topic of a child cluster and they are "similar" to a certain degree. Each cluster uses a global frequent k-itemset as its cluster label. A tree is build bottom-up by choosing a parent at level k-1 for each cluster k (start from the cluster with the largest number of items in its cluster label).

- *Prune the cluster tree:* Sibling clusters that are similar based on Inter-Cluster Similarity are merged into one cluster. Also, each child cluster that is similar to its parent is replaced by its parent. Documents of child cluster are added on the parent's cluster.

2.3.3 SCuBA - Subspace Clustering

SCuBA [7] is part of an article recommendation system for researchers. It is a Collaborative filtering (CF) system that has the advantage of using information about users' habits in order to recommend potentially interesting items. This system exploits information from researchers' previous searches in order to recommend research papers that users with similar preferences had chosen.

Subspace clustering is a branch of clustering algorithm that is able to find low dimensional clusters in very high-dimensional datasets. Research/article data space is represented by a binary m × n matrix, where rows represent m researchers R=$\{r_1, r_2, ..., r_m\}$ and columns represents n articles A=$\{a_1, a_2, ..., a_n\}$. The algorithm is trying to find subspace clusters of researchers defined in subspaces of articles.

2.4 Snippet-based approaches

2.4.1 Suffix Tree Clustering (STC)

STC [39, 38] is a post-retrieval document browsing technique (that is used in Grouper). It treats a document as an ordered sequence of words. STC is an incremental and linear time clustering algorithm that is based on identifying the phrases that are common to groups of documents, building a suffix tree structure. This method naturally allows overlapping clusters. Moreover, it is applied to short snippets returned by Web Search engines.

STC is described in the following steps:

- *Document "cleaning":* Stemming is applied to snippets, sentence boundaries are marked and non-word tokens (e.g. numbers, HTML tags, most punctuation) are stripped out. The original document strings are kept, as well as pointers from the beginning of each word in the transformed string to its position in the original string.

- *Identifying base clusters:* An inverted index is constructed with the structure of a suffix tree. Suffix tree contains all the suffixes of all strings. Snippets are treated as strings of words, not characters. Based on this structure, base clusters are identified. *Base cluster* is defined as a set of documents that share a common phrase (ordered sequence of one or more words). Each base cluster is assigned a score that is based on the number of documents it contains and the number of words in phrase that characterizes this cluster.

- *Combining base clusters:* Base clusters with a high overlap in their document sets are merged. Overlap is identified with a binary similarity measure. Given two base clusters B_m and B_n, similarity of B_m and B_n is 1 iff:

$$\frac{|B_m \bigcap B_n|}{|B_m|} > 0.5 \text{ and } \frac{|B_m \bigcap B_n|}{|B_n|} > 0.5$$

  Otherwise, similarity is 0.

- Final clusters are scored and sorted based on the scores of their base clusters and their overlap. Only the top few clusters are reported. Each cluster is described by the number of documents it contains, the shared phrases of its base clusters and some sample document titles.

As mentioned above, each base cluster is assigned a score. This score is utilized by [9] in order to present a high-quality label for each cluster instead of the Grouper's presentation that includes all the labels of the base clusters. As a result, the highest ranked label, from the base clusters' labels, is selected as the final cluster label.

Figure 2.7 shows an example of the suffix tree of the strings "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too", numbered from 1 to 3. The nodes of the tree are drawn as circles. Each node has one or more boxes attached to it and each box includes two numbers. The first is the number of string that the suffix is originated from and the second is the index of this suffix inside the string.

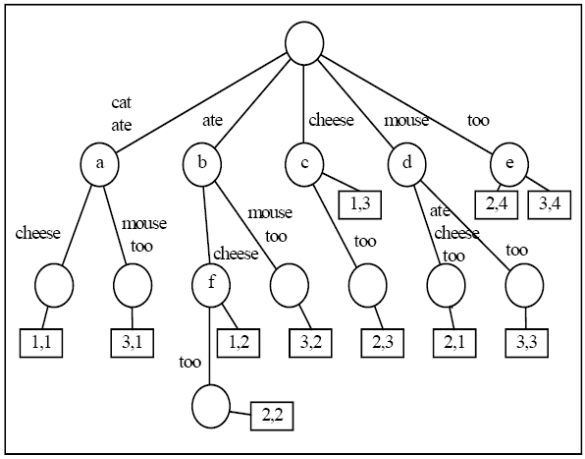


Figure 2.7: The suffix tree of the strings "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too".

Figure 2.8 shows the base clusters, nodes that contain two or more documents, that are derived from the suffix tree of Figure 2.7.

Node	*Phrase*	*Documents*
a	cat ate	1,3
b	ate	1,2,3
c	cheese	1,2
d	mouse	2,3
e	too	2,3
f	ate cheese	1,2

Figure 2.8: Base clusters derived from the suffix tree of Figure 2.7.

Figure 2.9 shows the *base cluster graph* of Figure 2.8 base clusters. Two nodes are

connected iff the two base clusters have a similarity of 1. A cluster is defined as being a connected component in the base cluster graph. Each cluster contains the union of the documents of all its base clusters. In Figure 2.9, there is one connected component, therefore one cluster.

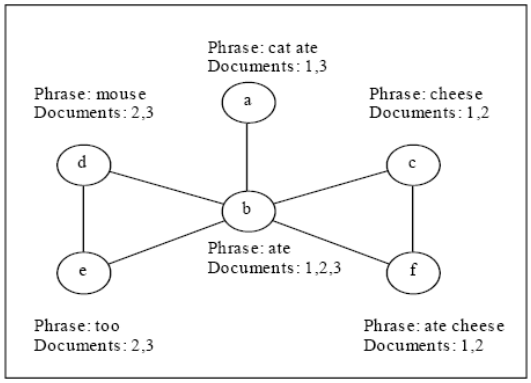


Figure 2.9: The base cluster graph of the example given in Figures 2.7 and 2.8.

2.4.2 TermRank algorithm

TermRank [17] is a variation of the PageRank algorithm that counts term frequency not only by classic metrics of TF and TF×IDF but also it considers term-to-term associations. It is based on a relational graph representation of the content of web document collections. From each Web page the blocks in which the search keyword appears are retrieved. TermRank is trying to separate terms into three categories:

- in *discriminative terms* that belong to a specific context and are strongly related with a distinct sense of the keyword search term
- in *ambiguous terms* that have many senses, and
- *common terms* that appear in many distinct contexts of a keyword search term.

It ranks discriminative terms higher than ambiguous terms, and ambiguous terms higher than common terms.

Document collection is transformed into a weighted undirected graph where nodes are the terms and edge weights is the co-occurence of two terms in the collection. TermRank is

calculated by the following formula:

$$TR(i) = \sum_{j \in N(i)} \frac{TR(j) \cdot w_{ij}}{\sum_{k \in N(j)} w_{jk}} \tag{2.1}$$

where $N(x)$ represents the set of neighbors of the node x and w_{ij} is the number of times edge (i,j) appears in the entire data. Term ranks are estimated after a number of iterations of equation (2.1). Number of iterations is not predetermined but TermRank runs until the difference between two iterations is less than a small threshold d.

Iteration 0: $TR^{(0)}(i) = TF(i)$

Iteration t+1:

$$TR^{(t+1)}(i) = \sum_{j \in N(i)} \frac{TR^{(t)}(j) \cdot w_{ij}}{\sum_{k \in N(j)} w_{jk}}$$

Figure 2.10 shows a fragment of a relational graph of apple data. Sizes of nodes and thickness of edges are proportional to their term frequencies and association strengths respectively. For example, discriminative terms such as "mac" and "recipe" have neighbors with strong associations.

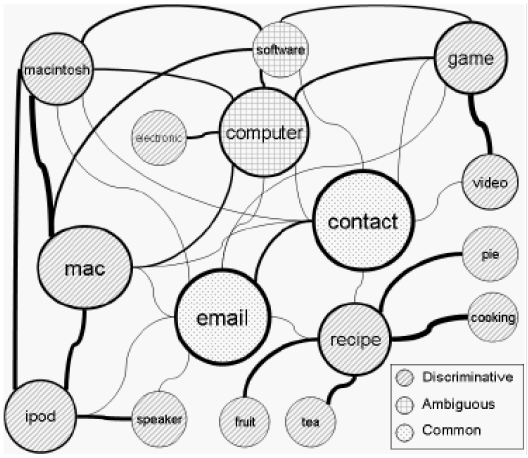


Figure 2.10: A fragment of a relational graph of apple data.

Figure 2.11 shows the ranks of terms based on TermRank and TF×IDF. Initial ranks of the terms are TF values (iteration 0). TermRank converges in 20 iterations and the first five terms are the discriminative terms, the next one is the ambiguous term and finally the common term. This order of terms is better than TF×IDF ranking (computer, mac, contact,

ipod, game, macintosh, video) that gives "computer" the highest rank and put "contact" on the third place.

	TermRank		TF/IDF
	iteration: 0	iteration: 20	
mac	0.1389	0.2600	0.4606
macintosh	0.0663	0.2262	0.2569
game	0.0764	0.1452	0.3666
ipod	0.0928	0.1270	0.3751
video	0.0568	0.1128	0.2549
computer	0.2147	0.1059	0.4679
contact	0.3537	0.0226	0.3864

Figure 2.11: Ranks of terms based on TermRank and TF×IDF.

TermRank can be applied on a set of Web pages that correspond to a specific query. The top-T terms ranked by TermRank can be used as feature vectors in K-means or another clustering algorithm.

Furthermore, there are snippet-based approaches that use *external resources (lexical or training data)*. Some of them are described below.

2.4.3 Deep Classifier

Deep Classifier [37] trims the large hierarchy, returned by an online Web directory, into a narrow one and combines it with the results of a search engine using a classifier based on Bayesian Classifier.

Roughly the process can be described in five steps:

- The query is submitted to an online Web directory to get the categories hierarchy and to a search engine to get the search results.
- The categories hierarchy is pruned. The leaf nodes in the pruned hierarchy are the target category candidates.
- A training data selection strategy is applied to the pruned hierarchy.

  There are three different strategies for training data selection:

  – Flat Strategy

Target category candidates are placed directly to the root of the hierarchy and classification is performed by a flat classifier. As a result, hierarchical structure of the web directory is not considered.

  - Hierarchical Strategy

    Each level of the hierarchy is classified (estimation of probabilities). The estimated probability of each non-candidate category is propagated to their candidate offspring.

  - Ancestor-Assistant Strategy

    It is a combination of the above strategies. Training data of the candidate category are combined with those of its ancestors and siblings.

- Based on the training data, classification model learning is performed.

  Two classification models were used, naive Bayesian and discriminative naive Bayesian classifier. Each document is regarded as a sequence of random variables that corresponds to the sequence of the words. The classifiers estimate the probability that a document (sequence of words) belongs to a category. Discriminative naive Bayesian classifier also takes account the appearance of a word in only one category (discriminative word).

- Classification of the search results and presentation of the hierarchy.

2.4.4 Salient phrases extraction

The purpose of this technique [41] is to produce clusters with highly readable names. It extracts and ranks *salient phrases* as candidate cluster names. Salient phrases are ranked by a score, defined by a regression model, on five different properties, learned from human labeled training data.

Concisely, algorithm's steps are the following:

- *Search result fetching:* the web page of search results returned by a certain Web search engine is analyzed by an HTML parser and snippets are extracted.

- *Document parsing and phrase property calculation:* Porter's stemming algorithm is applied to each word. Salient phrases, all possible n-grams ($n \leq 3$), are produced from

the snippets and the phrases with frequency no greater than 3 times, are considered as noise and are filtered out. For each phrase, five different properties are calculated. These properties are Frequency/Inverse Document Frequency, Phrase Length, Intra-Cluster Similarity, Cluster Entropy and Phrase Independence.

- *Salient phrase ranking:* Salient phrases are ranked by a regression model that combines the five properties. Moreover, the document list, which corresponds to salient phrases, constitutes the candidate clusters.

- *Post-processing:* The phrases that contain only stop-word or words of the query are discarded. Clusters are generated by merging the candidate clusters that their common documents exceed a certain threshold.

Figure 2.12 shows an example of the salient phrases extraction algorithm when submitting the query q= `jaguar`.

Figure 2.12: An example of Salient phrases extraction algorithm

2.4.5 Automatic construction from text information bases

Automatic construction's goal is to identify useful facets and to create hierarchies from them. It is achieved with the selection of a set of facets and assigning each item of a collection to a subset of these facets. A collection can contain either textual or text-annotated items.

Automatic construction does not use predefined facets and manually constructed hierarchies of them as it happens in some commercial systems (e.g. Amazon, eBay).

Unsupervised Facet Extraction for Collections of Text Documents

This is an unsupervised technique used for the extraction of facets from free-text items [12]. Free-text items are neither associated with descriptive keywords nor organized across facets, so external resources are used to identify facet terms.

Automatic facet discovery has three steps:

1. identifying important terms,

2. deriving context using external resources, and

3. comparative term frequency analysis.

In the first step the words that characterize the content of each document (important terms) are located. For this purpose *Named Entities (LPNE*[3]*)*, *Yahoo Terms* and *Wikipedia Terms* are used. Wikipedia Terms is a tool that creates a relational base with the titles of all Wikipedia pages. A term that matches a title of this base is characterized as important. Also, redirected pages are used to capture variations of the same term and anchor text in order to find different descriptions of the same concept.

In the second step important terms are used so as to query one or more external resources and enrich important terms with the retrieved terms. External resources used are *Google*, *WordNet Hypernyms*, *Wikipedia Graph* and *Wikipedia Synonymous*. Wikipedia Graph uses the links that appear in the page of each Wikipedia entry in order to measure the level of association between two connected Wikipedia entries. The top-k terms with the highest scores that are connected with a specific term t are returned. Moreover, Wikipedia Synonymous identify variations of the same term. It uses the titles of entries that redirect to a particular Wikipedia entry and the anchor texts that link to a particular term.

At the end of step 2, two collections exist. The original collection and a contextualized one that was constructed from both the terms of the original collection (step 1) and the terms derived from the second step.

[3] *http://www.alias-i.com/lingpipe/*

In the final step, the difference in term frequencies, between the original collection and the expanded collection are exploited so as to identify the candidate facet terms which are expected to be infrequent in the original collection but frequent in the expanded one.

Facet extraction efficiency is influenced more by Yahoo!Term Extractor during the term extraction step and by Google external resource in the second step. This happens because web-based resources are slower than Wikipedia and WordNet that are faster since they are stored locally.

2.5 STC-based approaches

2.5.1 STC based on True Common Phrase Label Discovery

This approach [19] uses a suffix tree with N-gram. It is trying to alleviate the problems generated from the use of N-gram like the big number of generated base clusters and the extraction of partial common phases when the length of N-gram is smaller than the length of true common phrases.

The algorithm consists of four steps.

- Pre-processing

  A stemming algorithm is applied to snippets and non-word tokens are eliminated.

- Base cluster Identification

  A suffix tree is built with N-grams and then the internal nodes that do not contain snippets and have one child are compacted into one node. The internal nodes which contain at least one snippet at leaf nodes are selected as base clusters. Base clusters that their phrases contain only query words are eliminated.

- Combining base clusters with a partial join operation

  In order to reduce the number of generated base clusters and to find a true common

phrase a join operation between two clusters is performed based on Eq. (2.2)

$$A \oplus B = \left\{ \begin{array}{c} a_0 \bigoplus \\ a_1 = b_0 \\ a_2 = b_1 \\ \vdots \\ a_n = b_{n-1} \\ \bigoplus b_n \end{array} \right\} if(A_{(d)} \subseteq B_{(d)} \ or \ B_{(d)} \subseteq A_{(d)}) \tag{2.2}$$

where A and B are base clusters, $A_{(d)}$ is snippets of cluster A, $B_{(d)}$ is snippets of cluster B, $\{a_0, a_1, ..., a_n\}$ is a set of terms that appear in cluster's A label and $\{b_0, b_1, ..., b_n\}$ is set of terms that appear in cluster's B label.

- Ranking Cluster

  Clusters are reordered according to their base clusters scores. Base cluster's score is calculated according to Eq. (2.3):

$$S(m) = |d| * f|m_p| * f(query) * \sum tfidf(p_i, d) \tag{2.3}$$

$$f|m_p| = \left\{ \begin{array}{c} 0, \ if \ |p| = 1 \\ |p|, \ if \ 2 \leq |p| \leq 8 \\ a, \ if \ |p| > 8 \end{array} \right\}$$

$$f(query) = \left\{ \begin{array}{c} 100, \ if \ query \ word \ appear \ in \ phrase \\ 1, \ if \ query \ word \ not \ appear \ in \ phrase \end{array} \right\}$$

  where $|d|$ is the number of snippets in cluster m and $|m_p|$ is the number of words in phrase p.

2.5.2 STC with X-gram

STC with X-gram [34] is a variant of STC with N-gram. STC with N-gram has the advantage that fewer words are inserted to the suffix tree as suffixes are no longer than N words. Therefore, suffix tree has lower memory requirements than original STC and its building time is reduced. In STC with X-gram, X is an adaptive variable which denotes the

maximum length of a suffix that can be inserted into the tree. True common phrases can be inserted into the tree as a whole and noise words sequences are limited.

The algorithm can be divided into three steps.

- Non-informative text and stop-words are removed. Also, stemming is applied.
- A suffix tree with X-gram is created using the words sequences S[1...m]. The first word S[1] is inserted into the tree and after iteratively each word from 2 to m is checked and if it doesn't match with a node of the tree a new node is inserted, otherwise the longest match is inserted into the tree. For example, snippets of Table 2.1 construct the suffix tree of Figure 2.13.

D1	suffix,tree,clustering,x1,x2
D2	y1,suffix,tree,clustering,y2
D3	z1,z2,suffix,tree,clustering

Table 2.1: Snippets set

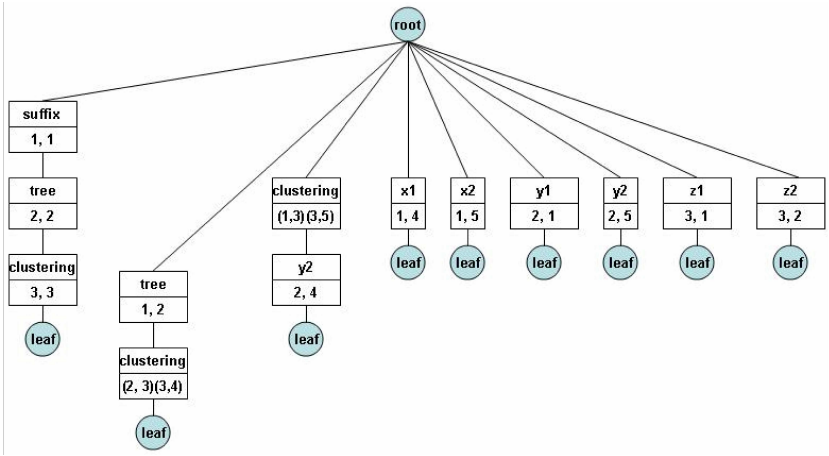


Figure 2.13: Suffix tree with X-gram

As a result of this process a phrase of length L will be inserted into the tree if this phrase appears L times at most. Since 90% of the true common phrases is no longer than 6, the max depth of X-gram was set to 6. Also, for the true phrases that are not fully inserted into the tree the partial phrases join operation is applied. Moreover, it is not possible to discover all snippets that a phrase appears, although this phrase is

wholly in the tree, so some branches are complemented using suffix links of Ukkonen's algorithm. Figure 2.15 shows the suffix tree of Figure's 2.14 after branch A was complemented.

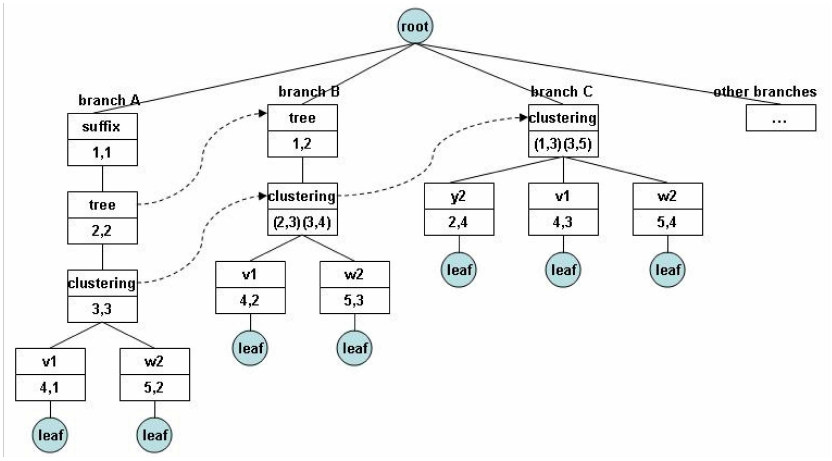


Figure 2.14: Suffix tree with X-gram

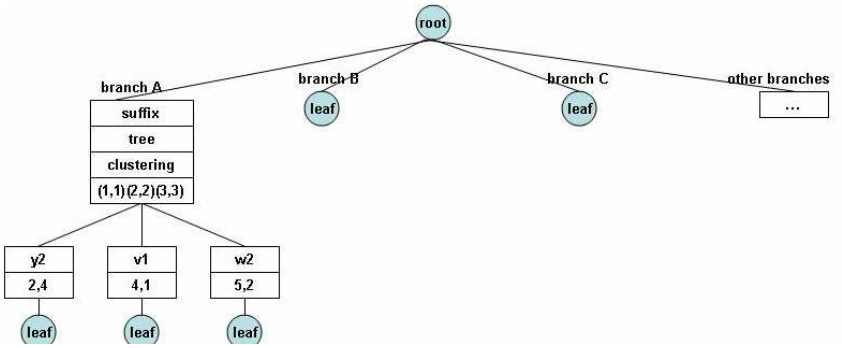


Figure 2.15: Suffix tree with X-gram after complement

- Candidate clusters are merged by checking only the k highest scored candidate clusters. Finally, they are scored and sorted.

2.5.3 Extended STC (ESTC)

Extended STC [10] is a variant of the STC algorithm, which proposes a new score formula to deal with overlapping documents in clusters since neither the score function in [40] nor the simplified score function used in [38] works properly when the whole documents

are clustered. Also, ESTC introduces a cluster selection algorithm to filter the clusters maximizing topic coverage and reducing overlap.

Original STC after the merging step of base clusters that consist a connected component, scores each merged cluster calculating the sum of scores of its underlying base clusters. But this score method favors clusters with a big number of base clusters as it over counts the overlapping documents in base clusters. ESTC uses a new scoring method where each document of a base cluster with score s and $|D|$ documents is assigned score $\frac{s}{|D|}$. For a merged cluster, the score of an overlapping document is the average of its scores from the base clusters of the merged cluster that it belongs. Finally, the score of a merged cluster is the sum of its document scores.

Cluster Selection Algorithm

The heuristic function $H = D - \beta(C - D)$ is used in order to select the cluster that adds more to H. In the above function C is the sum of the sizes of the clusters, and D is the number of distinct documents (coverage). $C - D$ represents the number of overlapping documents in the solution and β is a constant used to balance the trade off between overlap and coverage.

The algorithm approximately maximizes the heuristic by starting with an empty set of clusters and extending that solution incrementally (adding one cluster in each step). In each step, a k-step look-ahead is used to select the cluster to be added to the current solution. A k-step look-ahead considers all possible 1, 2, ..., k+1 cluster extensions to the current solution, and the best cluster of the best extension is chosen to be added to the current solution.

Furthermore, pruning is applied for any branch (possible extension) that can not possibly influence the final solution.

2.5.4 Findex

Findex [20] introduces a statistical algorithm which extracts candidate phrases by moving a window with a length of 1..$|P|$ words across the sentences (P) and fKWIC (frequent keyword-in-context) which extracts the candidate phrases like the statistical algorithm but with the requirement that they must contain at least one of the query words. For this reason, the candidate phrases for fKWIC are more fewer than the statistical algorithm.

The algorithms are applied on the snippets returned from the search engines. Initially, snippets are separated into sentences, stop-words and other irrelevant strings (words composed mostly of non-alphabetical characters) are removed. In fKWIC stop-words are removed in a latter step, during the phrases extraction. After this preprocessing phase, follows the candidate phrases extraction and finally the filtering of the category candidates.

In the filtering of the category candidates for the statical algorithm the extracted phrases that are composed of the same words or are subphrases are removed. For fKWIC, a candidate phrase is removed if the size of its association set gets too low compared to a similar phrase.

2.5.5 Link-based Clustering

Link-based clustering [35] is based on common links shared by pages in correspondence to document clustering algorithms that are based on common words/phrases among documents.

The idea is to cluster together pages that share common links as it is possible these pages to be tightly related. Common links for two web pages p and q mean common *out-links* (point form p and q) as well as common *in-links* (point to p and q). Moreover, **co-citation** measures the number of citations (out-links) in common between two documents and **coupling** measures the number of document (in-links) that cites both of two documents under consideration.

Each web page P in R (set of specified number of search results) is represented as two vectors: P_{Out} (N-dimension) and P_{In} (M-dimension). M and N denote the total number of distinct in-links and out-links extracted for all pages in R respectively. The ith item of vector P_{Out} indicates whether P has the correspondent out-link as the ith one in N out-links. If yes, the ith item is 1, else 0. Identically, the jth item of P_{In} indicates whether P has the correspondent in-link as the jth one in M in-links. If yes, jth item is 1, else 0.

Common links (in-link and out-link) shared by two pages P and Q are estimated using the cosine similarity measure:

$$Cosine(P,Q) = \frac{P \bullet Q}{||P||||Q||} = \frac{((P_{Out} \bullet Q_{Out}) + (P_{In} \bullet Q_{In}))}{||P||||Q||}$$

The steps of the algorithm are the following:

- *Filter irrelevant pages:* Only pages whose sum of in-links and out-links are at least 2 join the clustering procedure.

- *Define similarity threshold:* Similarity threshold is pre-defined to control the process of assigning one page to a cluster (usually 0.1).

- *Assign each page to clusters:* Each page is assigned to existing clusters when the similarity between the page and the correspondent cluster is above the similarity threshold. If none of the current existing clusters meet the demand, the page under consideration becomes a new cluster itself. Centroid vector is used when calculating the similarity and it is incrementally recalculated when new members are introduced to the cluster. While one page could belong to more than one cluster, it is limited to top 10 clusters based on similarity values. All pages that join clustering procedure are processed sequentially and the whole process is iteratively executed until it converges (centroids of all clusters are no longer changed).

- *Generating final clusters by merging base clusters:* When the whole iteration process converges, *base clusters* are formed. Final clusters are generated by recursively merging two base clusters if they share majority members using a specific merging threshold (usually 0.75).

2.5.6 Semantic, Hierarchical, Online Clustering (SHOC)

SHOC [42] is an extension of STC for oriental languages like Chinese.
SHOC is described in the following steps:

- *Data collection and cleaning:* The search results from several search engines are collected. The retrieved snippets are splitted into sentences according to punctuations and HTML tags. The non-word tokens are stripped and redundant spaces are compressed. The English words are stemmed using a stemming algorithm.

- *Feature extraction:* In STC common phrases are identified using a suffix tree structure but this is not efficient for key phrase discovery since oriental languages have much larger alphabet than English. Also, oriental languages like Chinese do not have explicit word separators (such as blanks in English) so partial phrases can be recognized if using

suffix trees. For these reasons, suffix array is used instead. The suffix array s of a document T, is an array of all N suffixes of T, sorted alphabetically.

- *Identifying and organizing clusters:* Taking the identified (from the previous steps) key phrases as terms, the search results can be described as a $m \times n$ term-document matrix A , whose row vectors represent the terms and column vectors represent the documents. The element A(i, j) = 1 if the i-th term T_i occurs in the j-th document D_j, or A(i, j) = 0. SHOC applies orthogonal clustering to the term-document matrix of Web search results using the SVD (Singular Value Decomposition) of the matrix A. Finally, SHOC organizes the clusters into a tree hierarchy by checking each pair of clusters if they can be merged into one cluster or to be treated as a parent-child relationship.

2.6 Synopsis and Comparison

Table 2.2 presents a number of basic features of clustering which are used for providing an overview of the functionality offered by each of the previously described clustering search engines and algorithms.

Generally, the table is filled with a text that explains how these functionalities are applied to the clustering engines/algorithms, or with an equation number, or with one of the symbols ✓, x, -:
✓means that the corresponding engine/algorithm supports the specific functionality,
x is the opposite of ✓, and
- means that we do not have enough information so as to know.

Presentation of clusters feature is the way which the results of clustering are presented. It can be a Tree, a List or a 2D-Map. *Tree* is referred to a tree structure that consists of the clusters' names/labels. *2D-Map* is referred to the positioning of clusters' names in a two-dimensional space. When clusters' names are not provided then a *List* of the clusters is presented. This list consists of a set of phrases that characterize each cluster and some sample documents or all of them.

Cluster structure feature corresponds to the structure of folder hierarchy and can be either flat (F) or hierarchical (H).

Size of cluster names corresponds to the number of words that consist a folder label.

Extra info for each cluster is any additional information that is given apart from the names and the documents of the clusters.

Use of external sources is the case that an algorithm exploits data from external sources such as knowledge bases or classified directories (e.g. dmoz, Wikipedia).

Cluster naming feature are the different ways the cluster labels are estimated. *Description Comes First* (DCF) [31] is an approach in which cluster construction and potential cluster label discovery are split into concurrent phases and merged in the end.

Ordering of clusters is determined by their score (Score(C_i)) that is calculated differently for each approach. For each cluster $C_i \subseteq ans(q) \subseteq Obj$, the scoring function takes one of the following forms:

$$Score(C_i) = |C_i| \quad \text{(i.e. the cardinality of the set } C_i) \tag{2.4}$$

$$= sim(q, \vec{C_i}) \quad (\vec{C_i} \text{ is the centroid of the vectors in } C_i) \tag{2.5}$$

$$= \sum_{d_j \in C_i} sim(q, d_j) \tag{2.6}$$

$$= \sum Score(C_{ij}) = \sum |C_{ij}| \cdot f(|L_{ij}|) \tag{2.7}$$

$$= \sum_{L_{ij} \in L_i} regression\ model\ of(TFIDF, LEN, ICS, CE, IND)(L_{ij}) \tag{2.8}$$

$$= \sum_{d_j \in C_i} Score(d_j) = \sum_{d_j \in C_i} avg(\frac{Score(C_{ij})}{|C_{ij}|}) \tag{2.9}$$

where L_i is the label of C_i and C_{ij} denotes the base cluster j of final cluster C_i. Moreover, in Eq. (2.7) $|L_{ij}|$ is the number of words in phrase L_{ij} that do not appear in a stoplist, or in too few (3 or less) or too many (more than 40%) documents of the collection. The function f penalizes single word phrases, is linear for phrases that are two to six words long, and becomes constant for longer phrases, i.e. it has the following form:

$$f(|L|) = \begin{cases} -x & \text{if } |L| = 1 \\ a|L| & \text{if } 2 \leq |L| \leq 6 \\ c & \text{if } |L| > 6 \end{cases}$$

Ordering of docs within clusters can follow the original order of documents (with respect to their similarity with the user query) or documents can be re-ranked by the score of the

associated cluster label. Original order of documents is the order in the ranked list that is returned by the queried search engines. For each document $d_j \in C_i$, score function takes the following formulas:

$$Score(d_j) = sim(q, d_j) \tag{2.10}$$

$$= sim(L_i, d_j) \tag{2.11}$$

where L_i is the label of cluster C_i.

Used in real/online system is checked for a clustering algorithm when there is a publicly available system that uses an implementation of it and is checked for an engine when it is on-line.

Overlapping clusters feature is checked when generated clusters are possible to contain common documents.

Open Source feature is referred to software that its source code is free and freely available to anyone interested in using or working with it.

Features	Engines-Algorithms													
	Grouper/ STC	Carrot	SNAKET	Vivisimo/ Clusty	Salient phrases extraction	Scatter/ Gather	FIHC	Quintura	STC with N-gram	STC with X-gram	ESTC	SHOC	Findex	Link-based
Presentation of clusters	List	Tree	Tree	Tree	Tree	List	Tree	2D-Map	-	-	Tree	Tree	Tree	-
Cluster structure	F	F	H	H	F	H	H	H	F	F	F	H	F	F
Size of cluster names	variable	variable	[1, 8]	variable	variable	variable	$\leq$ num of global frequent items	Fixed [1]	variable	[1,6]	variable	variable	variable	variable
Extra info for each cluster	cluster size + 3 sample docs	cluster size	cluster size	cluster size	cluster size	cluster size + sample docs	x	font size +2D area	-	-	-	x	cluster size	-
Use of external sources	x	x	KBs (dmoz, anchor texts)	-	training data and learning	x	x	x	x	x	x	x	x	✓
Cluster naming	DCF	DCF	DCF	-	DCF	x	DCF	-	DCF	DCF	DCF	DCF	DCF	-
Ordering of clusters (descending order)	sum of the scores of the phrase clusters (2.7)	cluster size (2.4)	-	-	sum of labels' salient phrases score (2.8)	percentage of relevant documents (2.6)	-	distance (2.5)	sum of the scores of the phrase clusters (2.3)	-	(2.9)	-	cluster size (2.4)	-
Ordering of docs within clusters	-	(2.10)	-	-	(2.10) or (2.11)	-	-	-	-	-	-	-	(2.10)	-
Used in real/online system	x	✓	✓	✓	x	x	✓	✓	x	x	-	✓	✓	x
Overlapping clusters	✓	✓	✓	✓	✓	x	x	-	✓	✓	✓	✓	✓	✓
Open Source	x	✓	✓	x	x	x	✓	x	x	x	x	x	x	x

Table 2.2: Features comparison of clustering search engines and algorithms

2.6.1 Discussion

Below we describe the main advantages and shortcomings of each approach.

STC algorithm creates coherent clusters by allowing documents to be in the same cluster even if they do not share a common phrase but rather share phrases with other documents of the cluster. Also, it reduces fragmentation of the produced clusters. STC is fast, incremental and creates overlapping clusters. It is robust in noisy situations (large number of loosely related documents, snippets without correlation to the query). Finally, original STC algorithm is applied on snippets but Carrot's STC implementation uses titles besides snippets, both with the same priority.

In contrast to STC, Findex does not merge clusters based on the documents they contain, but based on the similarity of the extracted phrases. However, no comparative results regarding cluster label quality have been reported in the literature.

Suffix tree structure (used by STC) can be constructed with N-grams instead of the suffixes. This structure maintains fewer words since suffixes are no longer than N words. Therefore, suffix tree with N-gram has lower memory requirements and its building time is reduced (however less common phrases are discovered and this may hurt the quality of the final clusters). However, STC with N-gram can identify only partial common phrases when N is smaller than the length of true common phrases so cluster labels can be unreadable. In [19] a join operation was proposed to overcome this shortcoming. A variant of STC with N-gram is STC with X-gram [34] where X is an adaptive variable. It has lower memory requirements and is faster than both STC with N-gram and the original STC since it maintains fewer words. It is claimed that it generates more readable labels than STC with N-gram as it inserts in the suffix tree more true common phrases and joins partial phrases to construct true common phrases. The performance improvements reported are small and from our experiments the most time consuming task is the generation of the snippets (not the construction of the suffix tree). No user study results have been reported in the literature.

Another extension of STC, Extended STC (ESTC) [10] is appropriate for application over the full texts (not snippets). To reduce the (roughly two orders of magnitude) increased number of clusters, a different scoring function and cluster selection algorithm is adopted. The cluster selection algorithm is based on a greedy search algorithm aiming at reducing the

overlap and at increasing the coverage of the final clusters. We do not share the objective of reducing overlap as in practice documents concern more than one topic. The comparison of ESTC with the original STC was done using a very small cluster set (consisting of only two queries) and no user study has been performed. Moreover, the major part of the evaluation was done assuming the entire textual contents of the pages (not snippets), or on snippets without title information. Summarizing, clustering over full text is not appropriate for a (Meta) WSE since full text may not be available or too expensive to process.

TermRank algorithm use only the blocks in which the search keyword appear in each Web page. This leads to a reduction of association strengths of words like 'search', 'back', 'copyright' because they rarely co-occur in the same block with the important terms(discriminative, ambiguous).

The Salient Phrases Extraction algorithm has the disadvantage that it needs a training phase, in order to choose the regression model that will be used, which is hard to adapt on the whole heterogeneous web. Another disadvantage is that the performance depends heavily on the search results returned by the queried Web search engines. For some queries (like 'apple', 'jokes') the vocabularies are relatively limited and the salient phrases can be extracted precisely. But for other queries (like 'Clinton', 'yellow pages') the search engine results contain various vocabularies and the performance for them is relatively low. Also, it is observed that the clusters of the top 10 salient phrases contain about half of the search results. A possibly solution would be to design a more sophisticate cluster merge algorithm. Despite the above disadvantages, this algorithm is linear, fast, has O(n) complexity, produces good cluster names and can be further examined in order to improve its drawbacks.

The algorithm used by SNAKET engine has the advantage that labels consist of non-contiguous words within a certain proximity window, in contrast with STC that is limited by the suffixes generation as they consists of contiguous words. Furthermore, SNAKET creates a weight balanced hierarchy, which means that the number of documents in the same level clusters is uniformly distributed.

Carrot^2 search engine has two disadvantages. First, the usability of topics presented is often reduced because the number of folders generated is big. Second, it fails to cluster together similar labels such as "knowledge, knowledge discovery", "mining and knowledge".

FIHC is more efficient and scalable because it reduces dimensionality by keeping only

the frequent items in document vectors. Also, it provides high clustering accuracy compared to common algorithms and is robust even when applied to large and complicated document sets. FIHC like the other Index-based approaches can be applied on a stand alone engine since they require accessing the entire vectors of the documents and they are computationally expensive.

Algorithms/Engines	Pros	Cons
Grouper/STC	incremental creates coherent clusters allows clusters to overlap treat document as an ordered sequence of words	does not provide cluster labels large memory requirements too many candidate clusters construct a long path of suffix tree
STC with N-Gram	lower memory requirements than STC needs less time to build the tree	labels a cluster with a partial phrase probably unreadable too many candidate clusters
STC with X-Gram	lower memory requirements discovers true common phrases maintains fewer words than STC with N-gram	
SNAKET	labels are not limited by the order of words uniform distribution of documents in clusters allows clusters to overlap	personalization interface's functionalities occur at client side
Salient Phrases Extraction	provides highly readable labels allows clusters to overlap	needs learning from training data performance depends on the web search results half of the results are distributed on the top-10 clusters
FIHC	scalability meaningful cluster labels	

Table 2.3: Pros and cons summary of algorithms and search engines

Flat Clustering	Complexity	Hierarchical Clustering	Complexity
Grouper/ STC	O(n)	SNAKET	O(nlogn+mlogmp)
Carrot	O(n)	HAC	$O(n^2)$
Salient Phrases Extraction	O(n)	FIHC	$O(n + g^2 + \sum_{f \in F} global\_support(f))$
K-means	O(nkT)		

Table 2.4: Complexity comparison of clustering algorithms

Table 2.4 reports the time complexity of various clustering algorithms and Table 2.5 explains the parameters of Table 2.4.

Parameter	Explanation
n	number of processed snippets
k	number of desired clusters
T	number of iterations
m	number of extracted sentences/words
p	number of labels extracted by SNAKET
g	number of remaining clusters at level 1
F	the set of global frequent itemsets
global_support(f)	number of documents that contain the f itemset

Table 2.5: Explanation for each parameter of Table 2.4

Chapter 3

Our Approach

This chapter describes our methods for results clustering and their evaluation. It is organized as follows. Section 3.1 formulates the problem and introduces notations. Section 3.2 describes the original STC and our extensions which we call STC+ and NM-STC. Section 3.3 reports comparative experimental results concerning the effectiveness and the efficiency of STC, STC+ and NM-STC. Finally, Section 3.4 summarizes the results.

3.1 Problem Statement and Notations

We consider important the requirements of *relevance, browsable summaries, overlap, snippet-tolerance, speed* and *incrementality* as described in [38]. Regarding the problem of *cluster labeling* we have observed that:

(a) *long labels are not very good*
E.g. not convenient for the left frame of a WSE, or for accessing the WSE through a mobile phone.

(b) *very short labels (e.g. single words) are not necessarily good*
E.g. longer labels could be acceptable, or even desired, in a system that shows the cluster labels in a horizontal frame.

(c) *an hierarchical organization of labels can alleviate the problem of long labels*

(d) *the words/phrases appearing in titles are usually better (for cluster labeling) than those appearing only in snippets*

Observations (a) and (b) motivate the need for configuration parameters. Observations (c) and (d) motivate the algorithms STC+ and NM-STC that we will introduce.

3.1.1 Configuration Parameters

We have realized that several configuration parameters are needed for facing the needs of a modern WSE. We decided to adopt the following:

$$\begin{aligned} K &: \text{number of top elements of the answer to cluster} \\ LL_{max} &: \text{max cluster Label Length} \\ LL_{min} &: \text{min cluster Label Length} \\ NC_{max} &: \text{max Number of Clusters} \end{aligned}$$

Obviously it should be $NC_{max} < K$. However the size of the current answer should also be taken into account. Specifically if $ans(q)$ is the answer of the submitted query, then we shall use A to denote the first K elements of $ans(q)$. However,
(a) if $|A| < K$ then we assume that $K = |A|$,
(b) if $|A| < NC_{max}$ then we assume that $NC_{max} = |A|/2$.
The latter can be justified by an example. Assume that $NC_{max} = 20$ and $|A| = 10$. Instead of giving the user 10 clusters, we believe that giving less (say 5) is better in the sense that clustering should give a synoptical overview of the results.

3.1.2 Notations

We use Obj to denote the set of all documents, hereafter objects, indexed by a WSE, and A to denote the top-K elements of the current answer as defined earlier (i.e. $A \subseteq Obj$ and $|A| = K$).

We use W to denote the set of words of the entire collection, and $W(A)$ to denote the set of the words that appear in a set of documents A (this means that W is a shortcut for $W(Obj)$).

Let $A = \{a_1, \ldots, a_K\}$. For each element a_i of A we shall use $a_i.t$ to denote the title of a_i, and $a_i.s$ to denote the snippet of a_i. Note that the elements of $W(A)$ are based on both titles and snippets of the elements of A.

If a is a document, then we shall use $P(a)$ to denote all phrases of a that are *sentence suffixes*, i.e. start from a word beginning and stop at the end of a sentence of a. For example, P("this is a test") = {"this is a test", "is a test", "a test", "test"},
while P("this is. A test") = {"this is", "is", "A test", "test"}.

We shall use $P(A)$ to denote all phrases of the elements of A, i.e. $P(A) = \bigcup_{a \in A}(P(a.t) \cup P(a.s))$.

If p is a phrase we shall use $Ext(p)$ to denote the objects (of A) to which p appears, i.e. $Ext(p) = \{a \in A \mid p \in a\}$. Also, we shall use $w(p)$ to denote the set of words that phrase p contains.

3.2 STC and Extensions

Our goal is to improve the Suffix Tree Clustering (STC) algorithm proposed by [38]. Specifically we attempt
(a) to improve the quality of cluster labels by exploiting more the titles (document titles can give more concise labels),
(b) to define a more parametric algorithm for facing the requirements of modern WSEs, and
(c) to derive hierarchically organized labels.

Specifically below we describe,
(a) the original STC (Section 3.2.1),
(b) a variation that we have implemented, called STC+ (Section 3.2.2), and
(c) a new algorithm that we have devised called NM-STC (Section 3.2.3).

3.2.1 The Original STC

This method is based on Suffix Tree Clustering (STC) algorithm, which was described in Section 2.4. The algorithm consists of the following steps:

1. Fetch snippets and titles of the top-K documents

2. Preprocess snippets and titles (mark sentence boundaries, remove stop-words)

3. Construct a suffix tree based on the preprocessed data

4. Find candidate clusters (base clusters)

5. Create final clusters by merging candidate clusters

Let's now describe the algorithm in more detail. The algorithm starts with the suffix tree construction. For each sentence of the input data all suffixes are generated and are inserted into the suffix tree. Each node of the tree that contains two or more documents is a *base cluster*. Each base cluster that corresponds to a phrase p is assigned a score which is calculated with the following formula:

$$score(p) = |\{a \in A \mid p \in a.t \text{ or } \ p \in a.s\}| * f(effLen(p))$$

where $effLen(p)$ is the effective length of label p defined as:

$$\begin{aligned} effLen(p) &= |w(p)| - |common(p)| \text{ where} \\ common(p) &= \{w_i \in p \mid df(w_i, A) \leq 3 \text{ or } \frac{df(w_i, A)}{|A|} > 0.4\} \end{aligned}$$

where $df(w_i, A) = |\{d \in A \mid w_i \in d\}|$.

The score of a base cluster is influenced by two factors. The number of documents that it contains and the function f that depends on the effective length of the base cluster's phrase. A phrase that corresponds to a bigger document size base cluster can represent better the cluster as it describes the contents of a big portion of the cluster.

The function f (that takes as input the effective length), penalizes single words, is linear for phrases with effective length from two to six words, and is constant for bigger phrases, specifically:

$$f(effLen(p)) = \begin{cases} 0.5 & \text{if } effLen(p) \leq 1 \\ effLen(p) & \text{if } 2 \leq effLen(p) \leq 6 \\ 7.0 & \text{if } effLen(p) > 6 \end{cases}$$

Afterwards, the overlap is calculated for all pairs of base clusters. Overlap is defined with a binary similarity measure. The similarity between two base clusters C_i and C_j is

defined as:

$$sim(C_i, C_j) = \begin{cases} 1 & \text{if } \frac{|C_i \bigcap C_j|}{|C_i|} > 0.5 \text{ and } \frac{|C_i \bigcap C_j|}{|C_j|} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

The next step is the *merging* of base clusters. In brief, each final cluster contains all base clusters that can be merged (two base clusters can be merged if their similarity equals 1). As a result the document set of a final cluster is the union of its base clusters' document sets and its cluster label is the label of the base cluster with the highest score. Due to cluster merging there can be documents that do not contain the label p. Let $C(p)$ be the document set of a cluster label p. The exact scoring formula for a final cluster is $score(p) = |C(p)| * f(effLen(p))$. Finally, clusters are sorted according to their score and are presented to the user.

3.2.2 STC+: A Variation of STC

Here we describe a variation of STC which differs in the way that clusters are scored and in the way base clusters are merged. Specifically, we adopt the following scoring formula:

$$score(p) = (|\{a \in A \mid p \in a.t\}| + |\{a \in A \mid p \in a.t \text{ or } p \in a.s\}|) * f(effLen(p)) \quad (3.1)$$

This formula favors phrases that occur in titles. In addition, we have modified the function f. Our variation penalizes single words and phrases that their effective length is bigger that 4 words, and is linear for phrases with effective length two to four words. In this way we favor small (but not single word) phrases.

Specifically our function f is defined as:

$$f(effLen(p)) = \begin{cases} 0.5 & \text{if } effLen(p) \leq 1 \text{ or } effLen(p) > 4 \\ effLen(p) & \text{if } 2 \leq effLen(p) \leq 4 \end{cases}$$

Regarding the computation of the similarity measure (that determines cluster merging) we consider as threshold the value 0.4 instead of 0.5. According to [9] it was observed that a 10% reduction from 50% to 40% can significantly affect the number of the generated clusters. From our experience, this value creates fewer and bigger clusters and solves some problematic cases of the original STC. For example, a base cluster with 2 documents that

is compared with a base cluster with 4 documents cannot be merged even if they have 2 common documents, because $2/4 = 0.5$. Therefore we used the following:

$$sim(C_i, C_j) = \begin{cases} 1 & \text{if } \frac{|C_i \bigcap C_j|}{|C_i|} > 0.4 \text{ and } \frac{|C_i \bigcap C_j|}{|C_j|} > 0.4 \\ 0 & \text{otherwise} \end{cases}$$

A lower than 0.4 threshold would decrease the *label precision* as it will be explained in Section 3.3.2.3.

Note that the title set of a final cluster is the union of its base clusters' title sets. Let $T(p)$ be the set of titles of a cluster label p. The exact scoring formula for a final cluster is $score(p) = (|T(p)| + |C(p)|) * f(effLen(p))$.

The example below shows how the utilization of titles from STC+ has as result the selection of better cluster labels compared both to STC that treats snippets and titles with the same priority and to STC approach that uses only the snippets:

Title 1: `Crete hotel: Atlantis`

Snippet 1: `Atlantis Hotel, Phone: +30-28970-27400 Fax: +30`

Title 2: `Hotel accommodation`

Snippet 2: `Knossos Royal Village, Crete, Phone: 2810897675 Fax: 2810897676`

Title 3: `Crete hotel: Agapi Beach`

Snippet 3: `Agapi Beach hotel, Phone: 2832089800 Fax: 2832089801`

Title 4: `Crete hotel`

Snippet 4: `Astoria Capsis Hotel(Eleytherias Square), phone: 2810345678 fax: 2810345679`

Title 5: `Accommodation Heraklion, Crete`

Snippet 5: `Hotels in small villages, Heraklion, Phone: 2810899075 Fax: 2810899076`

After sentence boundaries separation and their preprocessing from lexical analyzer they become:

Title 1: `crete hotel atlantis`

Snippet 1: `atlantis hotel phone fax`

Title 2: `hotel accommodation`

Snippet 2: `knossos royal village crete phone fax`

Title 3: `crete hotel agapi beach`

Snippet 3: `agapi beach hotel phone fax`

Title 4: `crete hotel`

Snippet 4: `astoria capsis hotel eleytherias square phone fax`

Title 5: `accommodation heraklion crete`

Snippet 5: `hotels small villages heraklion phone fax`

Table 3.1 shows the base clusters identified from STC using only the snippets 1, 2, 3, 4 and 5.

Index	Base Cluster	Documents	Score
1	fax	1, 2, 3, 4, 5	2.5
2	phone fax	1, 2, 3, 4, 5	10.0
3	hotel	1, 3, 4	1.5
4	hotel phone fax	1, 3	6.0

Table 3.1: Base clusters identified from STC using only snippets

After the end of clustering for Table's 3.1 base clusters only one final cluster is created with label *phone fax* and document set 1, 2, 3, 4, 5.

Table 3.2 shows the base clusters identified from STC using the titles 1, 2, 3, 4, 5 and the snippets 1, 2, 3, 4, 5 with the same priority.

Index	Base Cluster	Documents	Score
1	fax	1, 2, 3, 4, 5	2.5
2	crete	1, 2, 3, 4, 5	2.5
3	crete hotel	1, 3, 4	6.0
4	accommodation	2, 5	1.0
5	phone fax	1, 2, 3, 4, 5	10.0
6	hotel	1, 2, 3, 4	2.0
7	hotel phone fax	1, 3	6.0

Table 3.2: Base clusters identified from STC using titles and snippets

After the end of clustering for Table's 3.2 base clusters two final clusters are created, *phone fax* with document set 1, 2, 3, 4, 5 and *accommodation* with document set 2, 5.

Table 3.3 shows the base clusters identified from STC+ using the titles 1, 2, 3, 4, 5 and the snippets 1, 2, 3, 4, 5.

Index	Base Cluster	Documents	Titles	Score
1	fax	1, 2, 3, 4, 5	-	2.5
2	crete	1, 2, 3, 4, 5	1, 3, 4, 5	4.5
3	crete hotel	1, 3, 4	1, 3, 4	12.0
4	accommodation	2, 5	2, 5	2.0
5	phone fax	1, 2, 3, 4, 5	-	10.0
6	hotel	1, 2, 3, 4	1, 2, 3, 4	4.0
7	hotel phone fax	1, 3	-	6.0

Table 3.3: Base clusters identified from STC+

After the end of clustering for Table's 3.3 base clusters two final clusters are created, *crete hotel* with document set 1, 2, 3, 4, 5 and *accommodation* with document set 2, 5.

As we can observe, both *phone fax* and *crete hotel* label consists of two words. Although, *phone fax* appears in five documents and *crete hotel* label appears only in three documents, *crete hotel* base cluster has bigger score because its label is contained in tree titles. Based on the scoring formula (3.1) we have

score("crete hotel") = (3+3)*2 = 12.0
score("phone fax") = (5+0)*2 = 10.0

so *crete hotel* is the label of the final cluster since *crete hotel* base cluster has the highest score.

3.2.3 A New Clustering Algorithm (NM-STC)

Here we introduce an algorithm called NM-STC (Non Merging - Suffix Tree Clustering). As in STC, we begin by constructing the suffix tree of the titles and snippets. Then we score each node p of that tree. Let p be a phrase (corresponding to a node of the suffix tree). Below we define four scoring functions:

$$
\begin{aligned}
score_t(p) &= |\{a \in A \mid p \in a.t\}| \\
score_s(p) &= |\{a \in A \mid p \in a.s\}| \\
score_{ts}(p) &= score_t(p) * |A| + score_s(p) \\
score_{tsi}(p) &= score_t(p) * |A| * N + score_s(p) * N + PIDF(p)
\end{aligned}
$$

PIDF stands for Phrase IDF and N is the total number of indexed documents ($N = |Obj|$). If p is a single word (w), then $PIDF(p)$ is the IDF of w (i.e. $IDF(w) = \frac{N}{|\{d \in Obj \mid w \in d\}|}$). If p is a phrase consisting of the words $\{w_1, \ldots, w_m\}$, then PIDF is the average IDF of its words, i.e.

$$PIDF(p) = \frac{1}{m} \sum_{i=1}^{m} IDF(w_i)$$

or alternatively $PIDF(p) = \max_{w \in p}(IDF(w))$. In our experiments we used the average IDF. The IDF can be computed based on the entire collection if we are in the context of a single WSE. In our case, the index of `Mitos` stores only the stems of the words so $IDF(w)$ is estimated over the stemmed words. If we are in the context of a MWSE (Meta WSE), then IDF could be based on external sources, or on the current answer[1].

NM-STC uses the $score_{tsi}(\cdot)$ scoring formula. This scoring function actually quantifies a qualitative preference of the form $title \rhd snippet \rhd PIDF$, where $\rhd$ denotes the priority operator [8]. Notice that PIDF has the lowest priority. It is used just for breaking some ties. From our experiments, the number of broken ties is low, so it does not affect significantly the results.

NM-STC at first scores all labels of the suffix tree using the function $score_{tsi}(\cdot)$. Subsequently we select and return the top-NC_{max} scored phrases. Let B be the set of top-NC_{max} scored phrases. Note that it is possible for B to contain phrases that point to the same objects, meaning that the extensions of the labels in B could have big overlaps. In such cases we will have low "coverage" of the resulting clustering (i.e. the set $\cup_{p \in B} Ext(p)$ could be much smaller than A).

Recall that STC merges base clusters having a substantial overlap in order to tackle this problem. However that approach leads to labels whose extension may contain documents that do not contain the cluster label (in this way users get unexpected results). Instead NM-STC follows a different approach that is described in the sequel, after first introducing an auxiliary notation. If $n(p)$ and $n(p')$ denote the nodes in the suffix tree that correspond to phrases p and p' respectively, we shall say that p is narrower than p', and we will write $p < p'$, iff $n(p)$ is a descendent of $n(p')$, which means that p' is a prefix of p. For instance, in our running example of Figure 3.1 we have $n(\text{"}ab\text{"}) < n(\text{"}a\text{"})$.

[1] $IDF(w) = \frac{|A|}{|\{d \in A \mid w \in d\}|}$

Returning to the issue at hand, our approach is the following: We fetch the top-NC_{max} labels and we compute the maximal elements of this set according to $<$. In this way we get the more broad labels (among those that are highly scored). If their number is less than NC_{max} then we fetch more labels until reaching to a set of labels whose maximal set has cardinality NC_{max}. So the algorithm returns the smaller set of top-scored phrases B that satisfies the equation $|maximal_{<}(B)| = NC_{max}$ if this is possible (even if B is the set of all nodes of the suffix tree, it may be $|maximal_{<}(B)| < NC_{max}$).

The extra labels fetched (i.e. those in $B \setminus maximal_{<}(B)$) are exploited by the GUI for providing an hierarchical organization of the labels (where the user can expand the desired nodes to see their immediate children and so on). Consider the example in Figure 3.1.(A1), and assume that $NC_{max} = 2$. The set of top-3 scored labels whose maximal elements are two are marked (as shown in Figure 3.1.(A2)). At the GUI level, the user can expand a and see the label b.

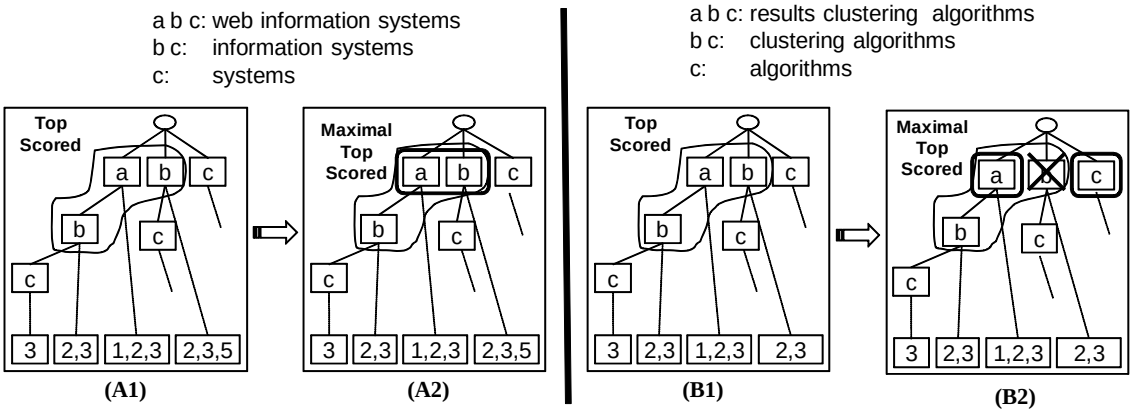


Figure 3.1: Two examples of NM-STC

The algorithm is sketched bellow. It takes as input a tree (the suffix tree) and returns another tree (the cluster label tree). Of course it also takes as input the configuration parameters, as well as the current query q.

If a cluster label p contains only the query words (i.e. $w(p) = w(q)$), then we exclude it from consideration, as such labels would be useless for the users. This is done by zeroing the scores of such labels (step (2)). At step (3) we zero the scores of the labels that do not satisfy the LL_{min} and LL_{max} constraints. The function getTopScored(sf, NC_{max}) returns the NC_{max} most highly scored nodes. At step (8) we remove from the list of maximal labels those that are subphrases of other labels and contain the same documents. Specifically,

Alg. $NM - STC$
Input: sf:SuffixTree, NC_{max}, LL_{min}, LL_{max}, q
Output: cluster label tree
(1) ScoreLabelsOf(sf)
(2) ZeroScoreLabelsEqualTo(sf,q)
(3) ZeroScoreLabelsLabelSize(sf,LL_{min}, LL_{max})
(4) topLabs = getTopScored(sf, NC_{max})
(5) Done=False
(6) while Done=False
(7) maxTopLabs = maximal$_<$(topLabs)
(8) maxTopLabls = ElimSubPhrasesSameExt(maxTopLabs)
(9) missing = NC_{max} - |maxTopLabs|
(10) if (missing>0)
(11) topLabs = topLabs ∪ getNextTopScored(sf,missing)
(12) else Done=True
(13)end while
(14)return topLabs, $<_{|topLabs}$

if $w(p) \subseteq w(p')$ and $Ext(p) = Ext(p')$ then we exclude p. This is shown in the example illustrated in Figure 3.1.(B1 and B2): the node b is discarded because it has the same extension with the node b that is child of a.

The function getNextTopScored(sf, M) returns the next M labels in the ranked list of labels (that are not already consumed).

3.2.4 Notes

STC keep references to the original source of the phrases but we are not doing that. It is not explained which of the original phrases is selected for presentation since a (stemmed) suffix of the tree corresponds to as many phrases as the number of the documents this suffix appears.

Moreover, in our implementation base clusters are those clusters that contain more than 2 documents but in Carrot there are additional criteria. Base clusters must also have score higher than a *minimum base cluster score threshold* and in the merging step only the top-N base clusters participate. In our implementation all base clusters participate in the merging process in order to achieve better coverage.

Another aspect is the documents that do not participate in any base cluster. None of the surveys explains how such cases are treated. From Carrot's interface we observe that these documents are inserted in an additional cluster with the label "Other topics".

3.3 Comparative Evaluation

Two are the main aspects of evaluation: efficiency and quality.

3.3.1 Efficiency

To evaluate efficiency we will measure the time needed for clustering the top-K documents (for various values of K) of each submitted query. Ideally the time of clustering should increase linearly with respect to K.

3.3.1.1 Time Performance

For the evaluation queries we counted the average time to cluster the top-{100, 200, 300} snippets. In NM-STC the IDF of the terms are in main memory from the beginning. Also recall that PIDF could be omitted from the scoring formula as it does not seem to influence the results (except in cases of very small result sets). In that case, the scoring formula used is $score_{ts}(p)$. The measured times (in seconds) are shown next (using a Pentium IV 4 GHz, 2 GB RAM, Linux Debian).

Alg	Top-100	Top-200	Top-300
STC	0.208	0.698	1.450
STC+	0.228	0.761	1.602
NM-STC	0.128	0.269	0.426

Notice that NM-STC is (two to three times) faster than STC and STC+. This is because NM-STC does not have to intersect and merge base clusters.

3.3.2 Effectiveness - Quality

3.3.2.1 UI Examples

The following screen shots presents three parallel frames which correspond to the original STC, STC+ and NM-STC algorithms. Each of the first three frames presents the generated

cluster label tree from the corresponding approach and the fourth frame shows the list of documents of the selected cluster label. All approaches use the same method for the snippets extraction. Also, for all approaches K was set to 100 and for NM-STC we used the parameters LL_{min}=1, LL_{max}=4 and NC_{max}=15.

Figures 3.2 and 3.3 show the clusters derived when submitting the query q= `ηράκλειο` and q= `uml` respectively.

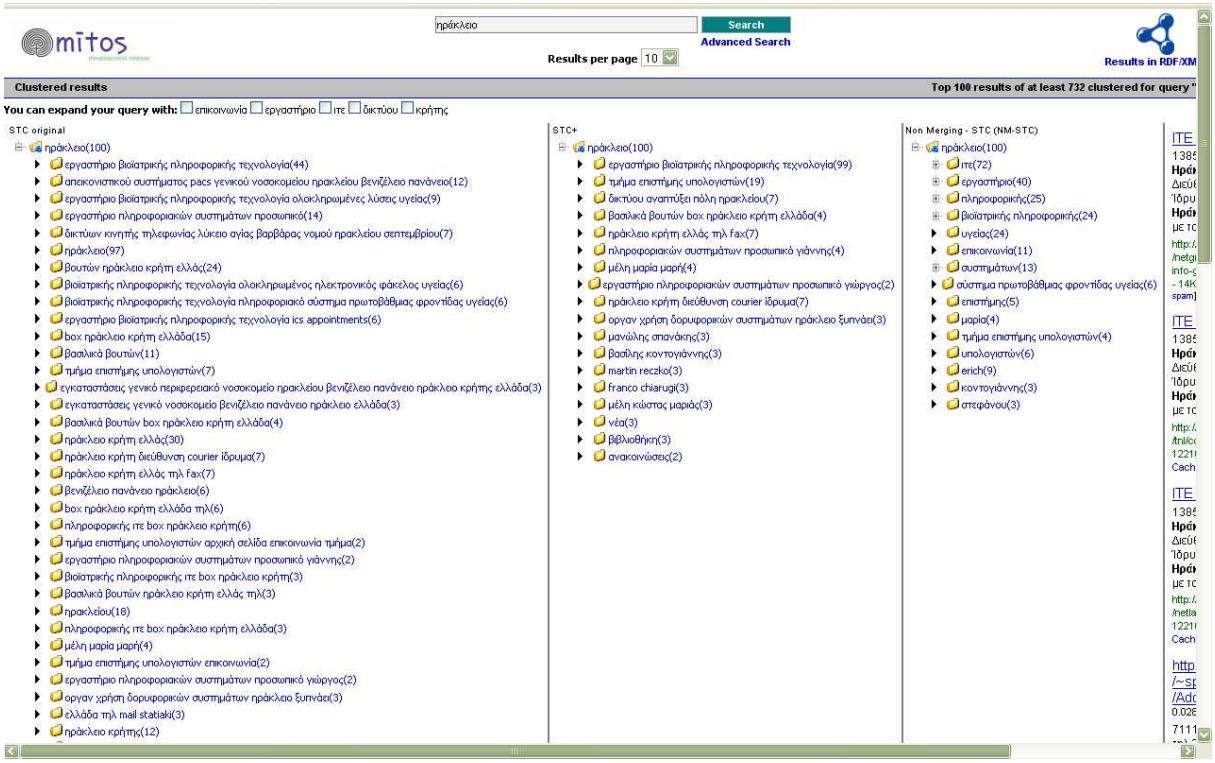


Figure 3.2: Results clustering for the query q=`ηράκλειο`

3.3.2.2 Evaluation by Users

We conducted an empirical evaluation over `Mitos` in order to investigate whether the users of `Mitos` were satisfied by the results clustering feature. Specifically we followed the following process: we defined 16 queries of different sizes consisting of small (single words), medium (2 to 3 words), and big (4 or more words) queries. Figure 3.4 shows the results sizes of these queries (see Appendix A) that range from 14 to 5029 hits.

The queries were given to 11 persons (from 22 to 30 years old, familiar with computers

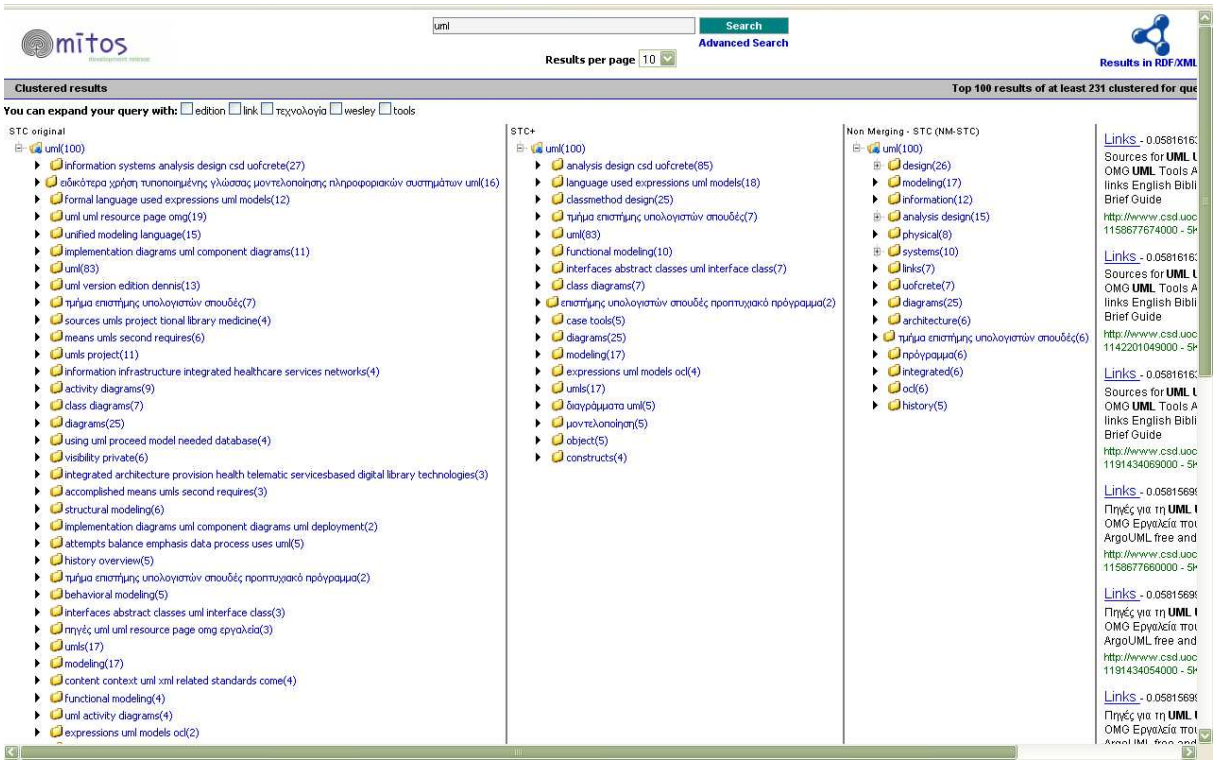


Figure 3.3: Results clustering for the query q=`uml`

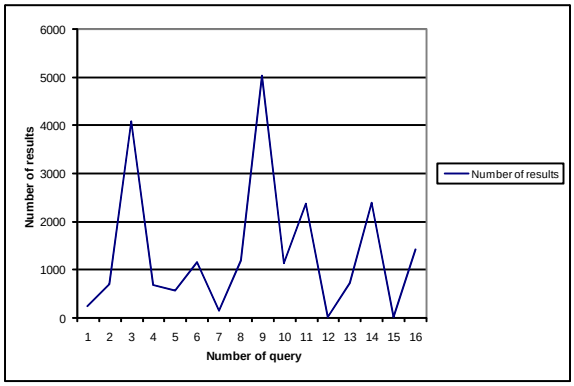


Figure 3.4: Result Sizes

and Web searching). Every participant had to submit each of these queries to a special evaluation system that we developed which visualizes the results of the three clustering algorithms in parallel (we used the parameters $K = 100$, $LL_{min} = 1$, $LL_{max} = 4$, $NC_{max} = 15$). Figure 3.5 shows the evaluation system for the query q=`crete`.

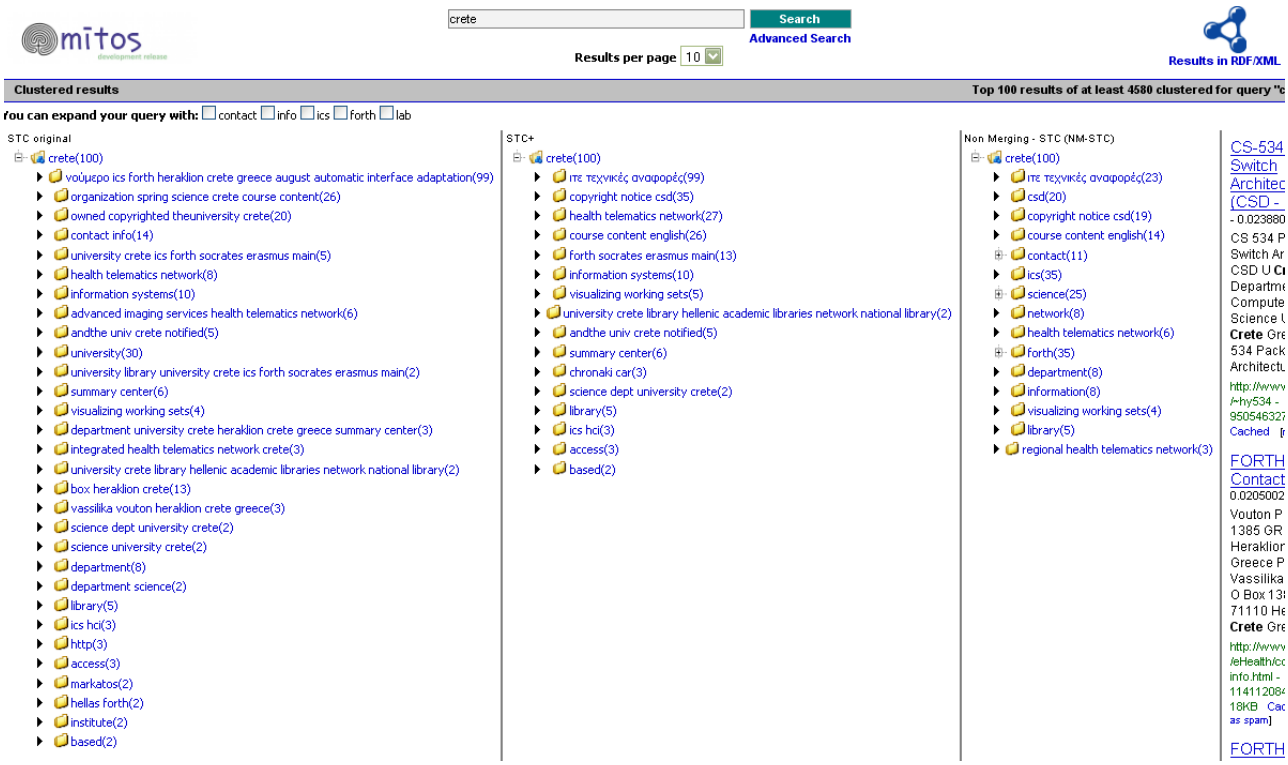


Figure 3.5: Evaluation system user interface

After inspecting the results, each participant had to rank the three methods according to (a) label readability, (b) cluster ordering, (c) number of clusters and (d) overall quality. So 16 * 11 * 4 = 704 user assessments in total. The users expressed their preference by providing numbers from {1, 2, 3}: 1 to the best, and 3 to the worst. Ties were allowed, e.g. STC:1, STC+:1, NM-STC:2 means that the first two are equally good, and NM-STC is the worst. In case all three were indifferent (they liked/disliked them equally), they were giving the value 0. Figure 3.6 illustrates the questionnaire.

We aggregated the rankings using Plurality Ranking (i.e. by considering only the winners, i.e. 1's) and Borda [13] ranking. Table 3.4 reports the average results ("PR" for Plurality and "BR" for Borda Ranking). In a PR column, the higher a value is the better

Evaluation Queries	Αναγνωσιμότητα Ετικετών (Label Readability)			Σειρα ετικετών (Cluster Ordering)			Πλήθος Ετικετών (Number of clusters)			Καλύτερη Προσέγγιση (Συνολικά) (Best method (overall))		
	STC	STC+	NM-STC	STC	STC+	NM-STC	STC	STC+	NM-STC	STC	STC+	NM-STC
Crete	3	2	1	3	2	1	3	2	1	3	2	1
Ηράκλειο	3	1	2	2	1	1	2	1	1	3	1	2
UML	2	1	1	3	1	2	3	2	1	3	2	1
SWKM	3	1	2	2	1	1	0	0	0	3	1	2
Βικελαία	1	3	2	1	3	2	1	3	2	1	3	2
Ανάκτηση Πληροφοριών	3	2	1	2	1	1	3	2	1	3	2	1
Διαχείριση Οντολογιών	3	2	1	3	2	1	3	2	1	3	2	1
Οπτικοποίηση Γράφων	3	2	1	2	1	1	2	1	1	2	1	1
Τηλεοπτικό Πρόγραμμα	2	2	1	2	2	1	2	2	1	2	2	1
Ο Μίτος της Αριάδνης	3	2	1	3	2	1	3	2	1	3	2	1
Διακοπές στη Νότια Κρήτη	3	2	1	3	2	1	3	2	1	3	2	1
Βιβλιοθήκη Ρεθύμνου	3	2	1	2	1	1	3	2	1	3	2	1
Φαρμακεία Ηρακλείου	3	2	1	3	2	1	2	1	1	3	2	1
How to install mitos	3	2	1	3	1	2	3	2	1	3	2	1
How to add jar files in Eclipse	1	1	2	1	2	3	3	2	1	1	2	3
ιπτάμενοι δίσκοι στη Νότια Κρήτη	3	2	1	3	2	1	3	2	1	3	2	1

Figure 3.6: Questionnaire

(i.e. the more first positions it got), while in BR column the less a value is the better. Specifically, to compute the PR value we summed all ones (i.e. first positions) and then we divided by 11*16 (i.e. $|users| \times |queries|$).

Criterion	STC		STC+		NM-STC	
	PR	BR	PR	BR	PR	BR
(a) Label Readability	2.41	33.5	6.25	23.16	9.41	20.83
(b) Cluster Ordering	4.75	28.33	7.33	21.75	6.41	24.9
(c) Number of clusters	2.33	33.5	5.83	23.33	10.41	19.91
(d) Best method (overall)	3.41	31.08	7.08	21.75	6.91	23.5

Table 3.4: Comparative Evaluation by Users

Criterion	STC		STC+		NM-STC	
	PR	BR	PR	BR	PR	BR
(a) Label Readability	3	3	2	2	**1**	**1**
(b) Cluster Ordering	3	3	**1**	**1**	2	2
(c) Number of Clusters	3	3	2	2	**1**	**1**
(d) Best method (overall)	3	3	**1**	**1**	2	2

Table 3.5: Relative Ranking by Users

Table 3.5 shows only the relative ranking of the algorithms: 1 for the best, 2 for the second, and 3 for the third in preference algorithm. Notice that the relative ordering is the same for both PR and BR. The results show STC+ and NM-STC are clearly the most preferred algorithms according to each of the three criteria, and according to the overall assessment. In particular, NM-STC yields the more readable labels, STC+ yields the best

cluster label ordering and NM-STC yields the best number of clusters. Regarding criterion (d) (overall quality), STC+ obtained the best result (PR: 7.08), NM-STC a slightly lower (PR: 6.91), while STC a much lower value (PR: 3.41).

In addition, we asked the participants to answer a small questionnaire. Table 3.6 shows the questions and the answers received. The results show that the majority prefers

(a) hierarchically organized labels,

(b) labels comprising one to three words, and

(c) 10-15 clusters.

Question	Results
Do you prefer Flat or Hierarchical cluster labels?	Flat (24%), Hierarchical (58%), Both are fine (18%)
Preferred cluster label length	$1-3$(75%) $3-6$(25%)
Preferred number of clusters	<10 (25%) $10-15$ (62.5%) $15-20$ (12.5%)

Table 3.6: Questionnaire

3.3.2.3 Clustering Evaluation Metrics

We decided to conduct an additional comparative evaluation between original STC, STC+, and NM-STC. Recall that B is the set of the labels returned by a clustering algorithm. For a $p \in B$ we shall use $C(p)$ to denote the set of objects that are assigned to cluster label p by the clustering algorithm. We used the metrics defined in Table 3.7.

Name	Definition
coverage	$coverage = \frac{\lvert\cup_{p\in B} C(p)\rvert}{\lvert A\rvert}$
average label length	$LL_{avg} = avg_{p\in B}\lvert w(p)\rvert$
overlap	$AvO = \frac{2}{\lvert B\rvert(\lvert B\rvert-1)} \sum_{i=1}^{\lvert B\rvert} \sum_{j=i+1}^{\lvert B\rvert} JO(p_i, p_j)$ where $JO(p_i, p_j) = \frac{\lvert C(p_i)\cap C(p_j)\rvert}{\lvert C(p_i)\cup C(p_j)\rvert}$
label precision	$AvLP = \frac{1}{\lvert B\rvert} \sum_{p\in B} LabelPrec(p)$ where $LabelPrec(p) = \frac{\lvert\{o\in C(p) \mid w(p)\subseteq w(o)\}\rvert}{\lvert C(p)\rvert}$

Table 3.7: Evaluation Metrics

Coverage measures the degree that clusters' extensions cover the answer A (the closer to 1, the better the clusters "cover" the answer A). Its value is low if the clusters cover a small portion of A and this implies that the clusters do not summarize the entire contents of A. The *label precision* of a label p is the percentage of objects in the extension of p that contain all words of p. It is clear that the label precision of NM-STC is (by construction) always 1, but this is not true for the other STC-based algorithms (due to the base cluster merging).

Table 3.8 reports the results of the evaluation. We report the average values for the queries used in the empirical evaluation. The overlap for NM-STC is computed over the maximal elements of B (i.e. those in $maximal_{<}(B)$). The results show that STC and STC+ have exactly the same coverage while NM-STC has slightly lower[2]. STC+ and NM-STC give smaller names than STC. STC+ and NM-STC have higher overlap (which is not bad). The label precision of STC+ is smaller than that of STC due to the threshold 0.4 vs 0.5 in base cluster merging. For threshold=0.3 the average precision of STC+ drops to 0.60 while for threshold=0.2 it further drops to 0.47. These results motivate the reason for not further decreasing this threshold.

Criterion	STC	STC+	NM-STC
coverage	0.994	0.994	0.869
average label length	3.185	2.906	2.249
overlap	0.038	0.048	0.099
label precision	0.893	0.756	1.0

Table 3.8: Comparative Results

3.4 Synopsis

In this work we focused on suffix tree clustering algorithms because they are fast, they do not rely on external resources or training data, and thus they have broad applicability (e.g. different natural languages). We presented a variation of the STC, called STC+, with a scoring formula that favors phrases that occur in document titles, and a novel suffix tree based algorithm called NM-STC that results in hierarchically organized clusters.

[2]In general all coverage values are acceptably high, e.g. higher than those in [20], and recall, that we could achieve 100% coverage by adding an artificial "rest" cluster label.

The advantages of NM-STC are that:

(a) the user never gets unexpected results, as opposed to the existing STC-based algorithms which adopt overlap-based cluster merging,

(b) it is more configurable w.r.t. desired cluster label sizes (STC favors specific lengths),

(c) it derives hierarchically organized labels, and

(d) it favors occurrences in titles (as STC+) and takes into account IDFs, if available.

The user evaluation showed that both STC+ and NM-STC are significantly more preferred than STC (STC+ is slightly more preferred than NM-STC). Figure 3.7 shows the aggregated rank (w.r.t. Borda) of each algorithm for each query of the evaluation collection (the ideal average BR value is 1 the worst is 3). We observe that STC was better than STC+ or NM-STC, only in one query (no=15). In addition NM-STC is about two times faster than STC and STC+. In future we plan to work towards further improving the quality of cluster labels and the interaction with the user.

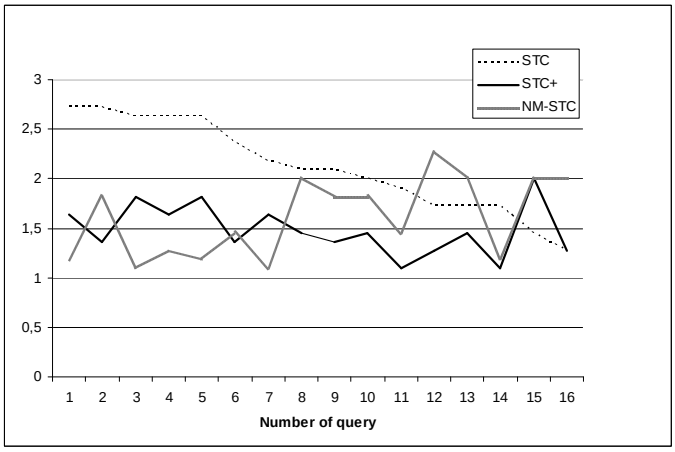


Figure 3.7: Evaluation per query

Chapter 4

Implementation and Applications

This chapter is organized as follows: Section 4.1 describes the `Mitos` Web Search Engine and Section 4.2 describes the application of the results clustering methods (presented in Chapter 3) over `Mitos`. Section 4.3 contains a detailed software description of the STC algorithm. Section 4.4 analyzes the preprocessing of the input data for the clustering methods. Section 4.5 presents `FleXplorer` and the coupling of dynamic taxonomies with results clustering. Section 4.6 describes the administrator parameters of `Mitos` for the snippet-based clustering approaches. Finally, Section 4.7 describes the application of the results clustering methods over Google.

4.1 Application over a Web Search Engine

The results clustering methods have already been applied on `Mitos` [28, 3] search engine. The first version of `Mitos` was developed as a student project in the IR course (CS463) by undergraduate and graduate students of the Computer Science Department of the University of Crete in three semesters (spring: 2006, 2007 and 2008). `Mitos` is not a meta-search engine. It has its own index (currently implemented using a DBMS) [27]. This allows exploiting its index in order to find additional information for the documents, apart from those that can be extracted from the snippets returned by `Mitos`. However, the index of `Mitos` stores only the stems of words, so the readability of indexed words is reduced. For this purpose, for each stemmed word it is also preserved the unstemmed word with the highest frequency in the collection set.

`Mitos` except from the three basic components of a search engine, namely Crawler, Indexer and Query Evaluator, it also consists of the Lexical Analyzer, Stemmer, Link Analysis-based Ranker, Result Clustering, Automatic Taxonomy, User Interface and Administration components, as shown in Figure 4.1.

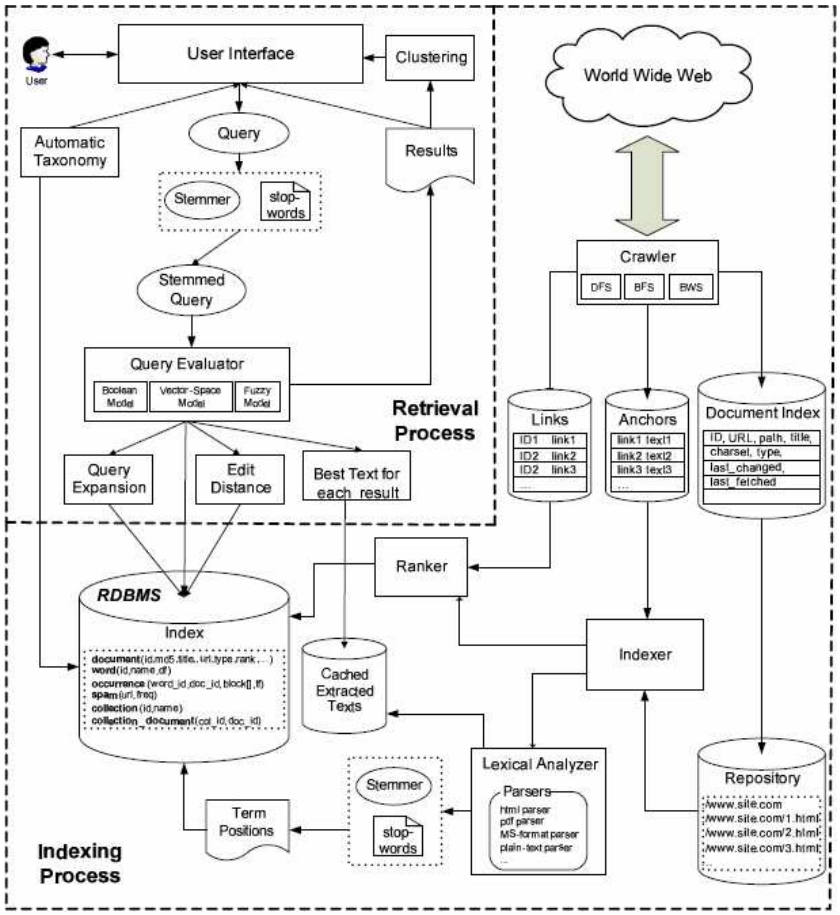


Figure 4.1: The Component Model of `Mitos`

The Crawler fetches Web pages starting from a specific list of URLs and recursively visits all the hyperlinks in each page. The downloaded pages are stored in a local repository and every page is assigned a unique ID number (md5). Also, a Document Index is created that keeps several properties for each page (like md5, path, title, last changed/fetched, etc) and a file that stores the hyperlinks and their anchor texts.

The Indexer uses the Document Index in order to analyze all downloaded documents and build the index. For each document it calls the Lexical Analyzer that returns the set

of words that contains, their frequency and their positions inside the document.

The Lexical Analyzer identifies tokens, removes stop-words and applies stemming algorithms. `Mitos` except from stemming for the English language, it also supports a Greek language stemmer.

The Query Evaluator is responsible for the retrieval process when a query is submitted. It supports several retrieval models, specifically the Vector Space, the Boolean, the Extended Boolean and the Fuzzy Model. If a query term does not exist in the index, then the system applies the Edit Distance algorithm in order to suggest terms that exist and their distance from the submitted term is less than a constant. In addition, the system suggests a list of terms which could be used to expand (refine) the submitted query.

4.1.1 Software Design Diagrams

The components of `Mitos` and their articulation are illustrated in the UML Component Diagram of Figure 4.2. The clustering algorithms are realized by the **Clustering Snippet-based** component.

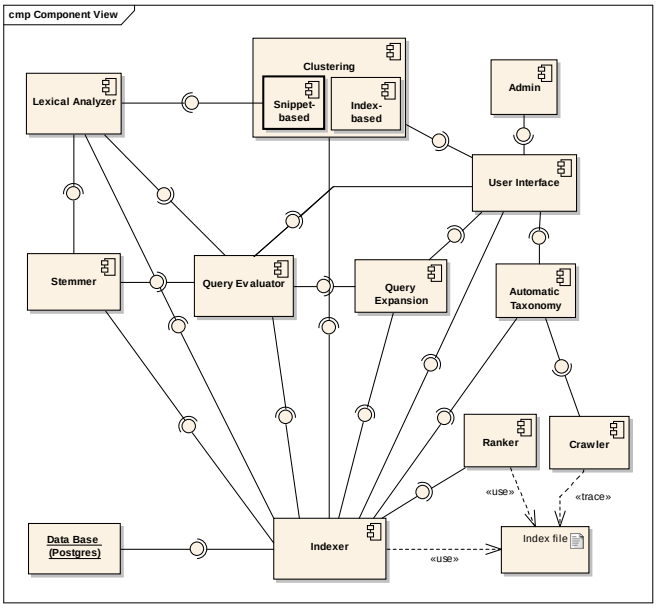


Figure 4.2: Component diagram of `Mitos` search engine.

The Clustering component is separated into the Index-based [1] and the Snippet-based

[1] Index-based approaches were implemented by Manolis Tavlas in the concept of an undergraduate study.

Clustering component. Both communicate with the User Interface, the Indexer and the Admin components. The Snippet-based component also communicates with the Lexical Analyzer. The User Interface gives the top-K documents from the list of the rank files returned by the Query Evaluator and receives the final cluster label tree. The Index-based component communicates with the Indexer in order to get the vector representations of documents. The Snippet-based component gets the documents' titles and words' document frequencies. The Admin component sends to Clustering the configuration parameters for clustering. Finally, the Snippet-based component communicates with the Lexical Analyzer for preprocessing the snippets and the titles.

4.2 Snippet-based Clustering Component

The component supports the original STC, the STC+ and the NM-STC algorithms as presented in Section 3.2.

4.2.1 Sequence Diagrams

The (UML) Sequence Diagram of Figure 4.3 shows the interactions between the components of `Mitos` during results clustering.

Figures 4.4, 4.5 and 4.6 show the sequence diagrams that represent the way that clusters are generated by the original STC and STC+ algorithm. Note that the similarity threshold (simThreshold) for Figure 4.6 (a) is used as a parameter and is set to 0.5 for the original STC and to 0.4 for STC+.

Figures 4.7, 4.8 and 4.9 shows the sequence diagrams for NM-STC algorithm. The first two steps of NM-STC ("construct suffix tree" and "prune suffix tree") are the same as presented above in Figures 4.4 (b) and 4.5 (a) for the original STC and STC+.

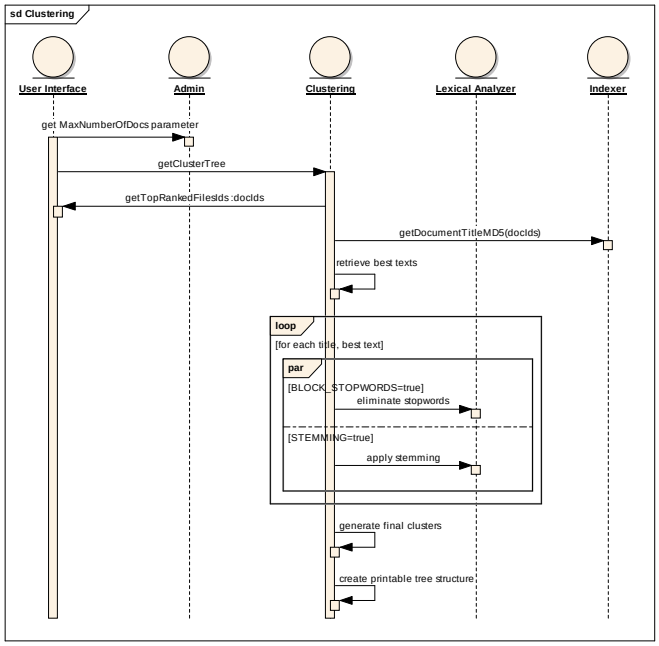
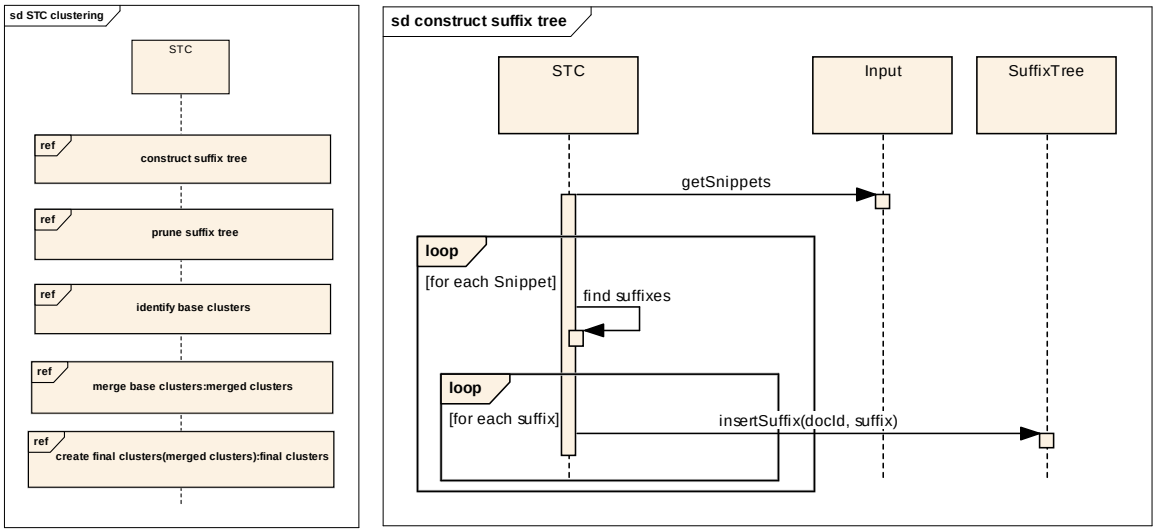
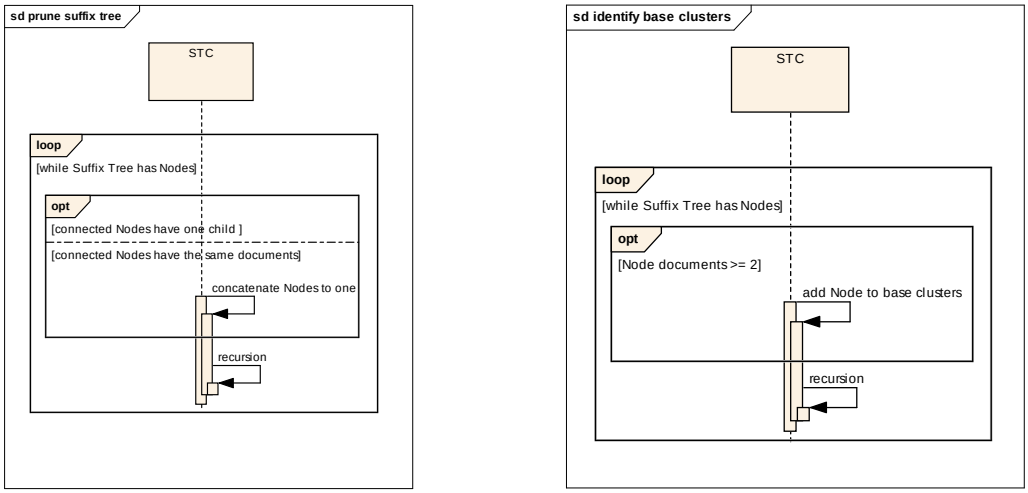


Figure 4.3: Sequence diagram of the results clustering process.



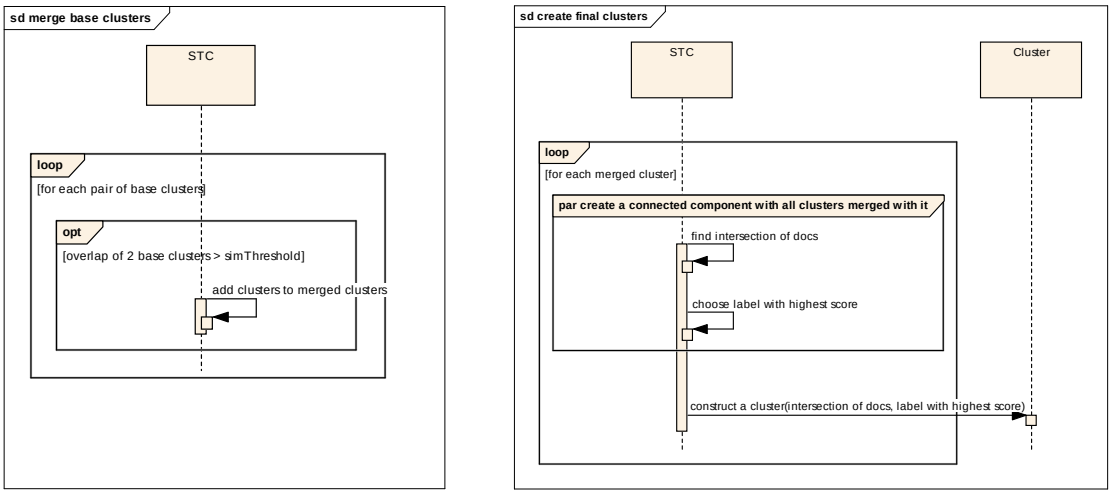
(a) (b)

Figure 4.4: Sequence diagrams for the generation of clusters by the original STC and STC+.



(a) (b)

Figure 4.5: Sequence diagrams for the generation of clusters by the original STC and STC+.



(a) (b)

Figure 4.6: Sequence diagrams for the generation of clusters by the original STC and STC+.

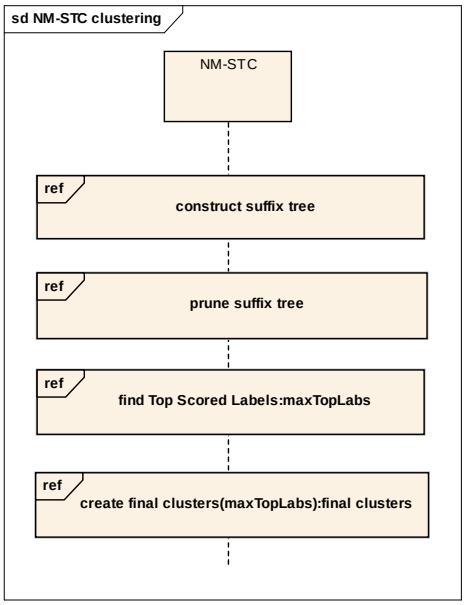


Figure 4.7: Sequence diagrams for the generation of clusters by NM-STC.

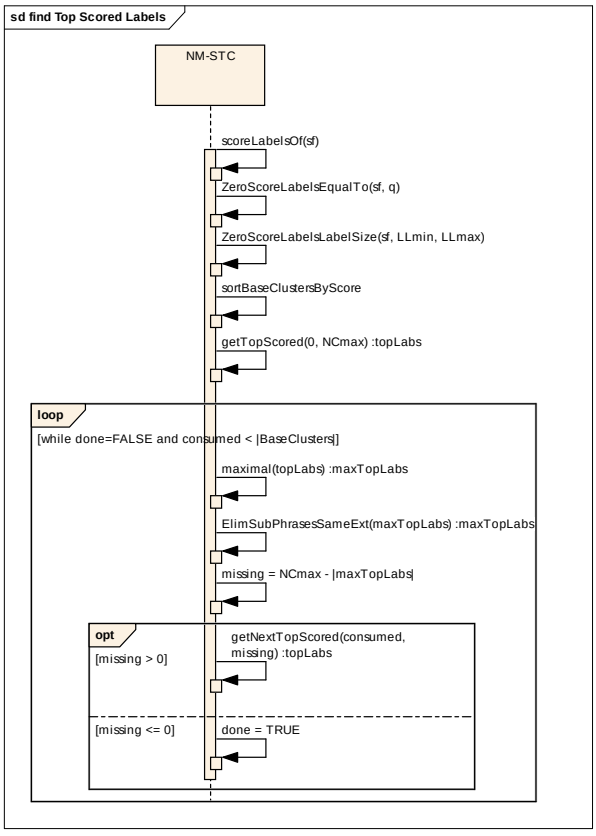


Figure 4.8: Sequence diagrams for the generation of clusters by NM-STC.

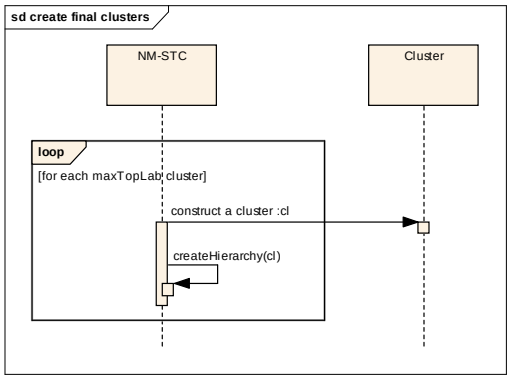


Figure 4.9: Sequence diagrams for the generation of clusters by NM-STC.

4.3 Implementation of STC

The algorithm **clustering(Snippets S[])** takes as input a set of snippets and returns the final clusters.

Alg. $clustering(SnippetsS[])$
(1) constructSuffixTree(Snippets S[])
(2) pruneSuffixTree(Snippets S[])
(3) identifyBaseClusters(SuffixTree ST, "", 0, $\emptyset$, 0)
(4) MC := mergeBaseClusters(BC)
(5) $Clusters$:= createFinalCLusters(MC,BC)
(6) return $Clusters$

The algorithm **constructSuffixTree(Snippets S[])** takes as input a set of snippets where each snippet is a triple of (document id, title, best text). It generates the suffixes of all strings and inserts each one in a suffix tree. Each node of the constructed tree corresponds to a single word. In the following algorithm $|Title|$ denotes the number of sentences of the string $Title$ and $|BestText|$ denotes the number of sentences of the string $BestText$. The function $insertSuffix$ takes as third parameter a boolean value which is set True if the inserted word appears in the title of the specific document id given as first argument. This means that for each suffix we keep separately the document ids that appear in their titles. This is especially useful for STC+ since this information is subsequently used by the function $identifyBaseClusters$ for computing the score of a base cluster.

Alg. $constructSuffixTree$(Snippets S[])
(1) for $i = 0$ to $|S|$ do
(2) $DocId$:= S[i].DocId
(3) $Title$:= S[i].Title
(4) for $j = 0$ to $|Title|$ do
(5) $suffixes$:= generateSuffixes($Title[j]$)
(6) for $z = 0$ to $|suffixes|$ do
(7) insertSuffix($DocId, suffixes[z]$, True)
(8) $BestText$:= S[i].BestText
(9) for $j = 0$ to $|BestText|$ do
(10) $suffixes$:= generateSuffixes($BestText[j]$)
(11) for $z = 0$ to $|suffixes|$ do
(12) insertSuffix($DocId, suffixes[z]$, False)

The algorithm **generateSuffixes(String str)** takes as input a string and generates all suffixes of that string that start after a white space or a punctuation symbol. For example, generateSuffixes("`The mitos search engine`") will return the following suffixes:

```
The mitos search engine

mitos search engine

search engine

engine
```

Alg. $generateSuffixes$(String str)
(1) $Suffixes := \emptyset$
(2) $words := |words(str)|$
(3) for $i = 0$ to $|words|$ do
(4) $suffix := \emptyset$
(5) for $j = i$ to $|words|$ do
(6) append $words[j]$ to $suffix$
(7) append white space to $suffix$
(8) trim $suffix$
(9) add $suffix$ to $Suffixes$
(10) return $Suffixes$

Let now see an example of the suffix tree. Suppose that we have the following snippets from three documents:

Snippet 1: `mitos web search engine`

Snippet 2: `indexer of mitos mitos adopts the tf - idf weighting scheme`

Snippet 3: `mitos : design and evaluation of a dbms - based web search engine`

After the separation of snippets into sentences and their processing from the lexical analyzer (stop-words removal) we get the following strings:

Snippet 1: `mitos web search engine`

Snippet 2: `indexer mitos mitos adopts idf weighting scheme`

Snippet 3: `mitos design evaluation dbms based web search engine`

Figure 4.10 shows the suffix tree constructed after calling function constructSuffixTree(Snippets S[]) for the above Snippets.

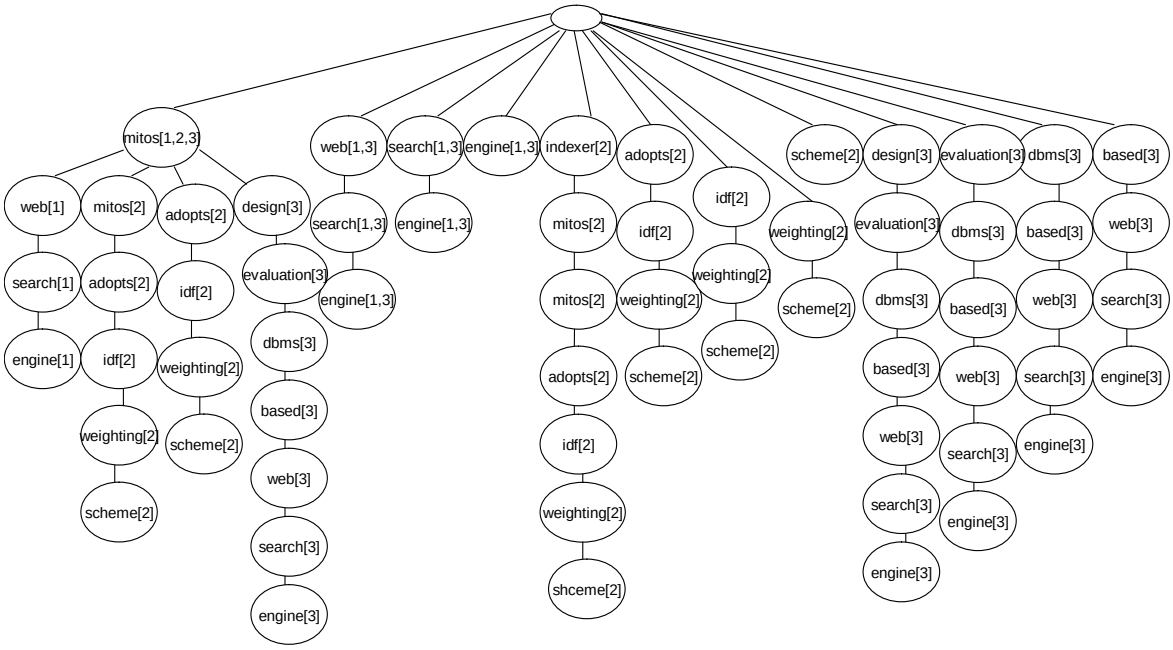


Figure 4.10: The suffix tree of the Snippets 1, 2 and 3.

The algorithm **pruneSuffixTree(SuffixTree ST)** takes as input a suffix tree and compacts it. Specifically if there is a node a that has only one children b and both nodes are associated with the same documents, then these nodes are "concatenated", i.e. only one node remains having as name the concatenation of the names at a and b. For example, as

shown in Figure 4.11, if node a contains documents 1 and 2, node b also contains documents 1 and 2, node a is the parent of node b and node b has two children, namely a node c which contains document 1 and a node d which contains document 2, then node a and node b are concatenated to a node labelled ab which contains documents 1,2 and is the parent of nodes c and d. Figure 4.12 shows the pruned suffix tree constructed after calling the function pruneSuffixTree(SuffixTree ST) for the tree of Figure 4.10.

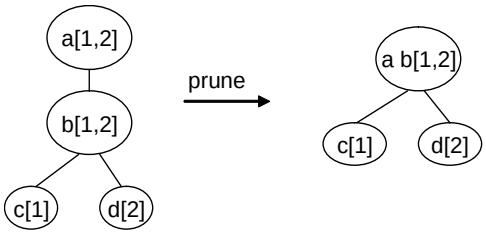


Figure 4.11: Suffix tree pruning example for two nodes a,b.

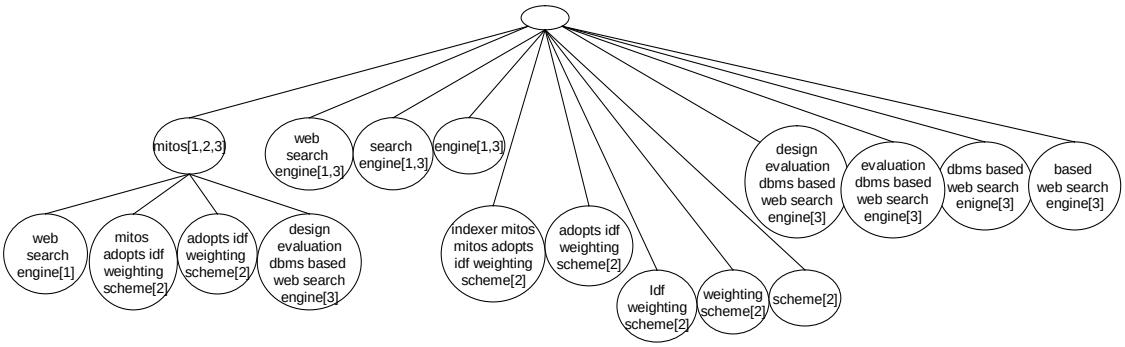


Figure 4.12: The pruned suffix tree of Figure 4.10

The implementation of STC+ relies on the implementation of STC with some minor changes regarding the new scoring formula, the new implementation of the function f and the new similarity threshold. In addition, during pruning, and whenever a node contains only one child node and both nodes contain the same documents but their phrases appear in different document titles, they are concatenated into one node that has as titles ids the titles ids of the broader node.

Let us now introduce the notion of *effective length* of a phrase. The algorithm **effectiveLength(String phrase, Snippets S[], Vocabulary Frequencies VF[], SuffixTree ST)** takes as input a string which is the phrase of a SuffixTreeNode, a set of snippets, a

Alg. $pruneSuffixTree$(SuffixTree ST)
(1) $Docs := \emptyset$
(2) while(ST has Next) do
(3) $W :=$ next key of ST
(4) $TN :=$ tree node of $ST[W]$
(5) if TN is not pruned
(6) $docIds :=$ document list of TN
(+)(STC+) $titleIds :=$ document title list of TN //for STC+
(7) $subTree :=$ children of TN
(8) $newName := W$
(9) $count := 0$
(10) while($|subTree| = 1$)
(11) $W2 :=$ next key of ST
(12) $TN2 :=$ tree node of $subTree[W2]$
(13) $docIds2 :=$ document list of $TN2$
(14) if $|docIds| = |docIds2|$ then
(15) $intersection := docIds \bigcap docIds2$
(16) if $|intersection| = |docIds|$ then
(17) append $W2$ to $newName$
(18) append *space character* to $newName$
(19) $subTree :=$ children of $TN2$
(20) $count := count + 1$
(21) else break;
(22) else break;
(23) if $count > 0$ then
(24) $newNode := newSuffixTreeNode()$
(25) set $subTree$ to $newNode$ children
(26) set $docIds$ to $newNode$ documents
(+)(STC+) set $titleIds$ to $newNode$ document titles //for STC+
(27) remove $ST[W]$
(28) add $(newName, newNode)$ to ST
(29) reset iterator of ST
(30) if $subTree \neq \emptyset$ then
(31) pruneSuffixTree($subTree$)

set of vocabulary frequencies where each frequency is a pair of (string, integer) and a suffix tree. It calculates the effective length of the given phrase, which is defined as the number of words in the phrase that do not appear in too few (3 or less) or too many (more than 40% of the collection) documents. At first the effective length of a phrase is the number of words it contains. Stop-words are removed during generation of snippets and are not considered in the calculation of the effective length. Subsequently and for each of the words whose frequency is more than three documents or it appears in more than 40% of the result set, the effective length is reduced by one. Some examples follow:

- effectiveLength("`web search engine`") = 3
- effectiveLength("`mitos`") = 1
- If the phrase is `mitos`, the number of the result set is 100 documents and the query is also `mitos`, thus `mitos` appears in 100 documents, then effectiveLength("`mitos`") = 0
- If the phrase is `genetic information`, the number of the result set is 100 documents and `genetic` appears in 2 documents, then effectiveLength("`genetic information`")= 1

Alg. $effectiveLength$(String phrase, Snippets S[], Vocabulary Frequencies VF[], SuffixTree ST)
(1) $Words := phrase$ splited with *space character*
(2) $EffLen := |Words|$
(3) $P := 0.4 * |S|$
(4) for $i = 0$ to $|Words|$ do
(5) if $Words[i] \in VF$ then $num := VF[Words[i]]$
(6) else
(7) $num := findDocsThatContain(Words[i], ST, \emptyset)$
(8) add $(Words[i], num)$ to VF
(9) if $num =< 3$ or $num > P$ then $EffLen := EffLen - 1$
(10) return $EffLen$

In order to calculate the effective length of a phrase we must find the number of documents that each word of the phrase appears. To avoid traversing the tree every time we see a word, we use a structure (Vocabulary Frequencies) to store the words we have already met and the corresponding number of documents in which they are presented.

The algorithm **identifyBaseClusters(SuffixTree ST, String subphrase, int effLen, int index, Base Clusters BC[])** takes as input a suffix tree and identifies the base clusters. A base cluster is a node that has at least 2 documents. For each base cluster, a score is calculated based on its label effective length. Before explaining how this is done, let's see the result of this step. Table 4.1 shows the base clusters identified after calling the function identifyBaseClusters(SuffixTree ST, "", 0, 0, ∅) over Figure's 4.12 pruned suffix tree.

Index	Base Cluster	Documents	Score
1	mitos	1, 2, 3	1.5
2	web search engine	1, 3	6
3	search engine	1, 3	4
4	engine	1, 3	1

Table 4.1: Base clusters identified from Figure's 4.12 suffix tree.

Figure 4.13 shows the base clusters graphically. That figure also shows the final result which in our case is a single cluster whose label is that of the cluster with the highest score.

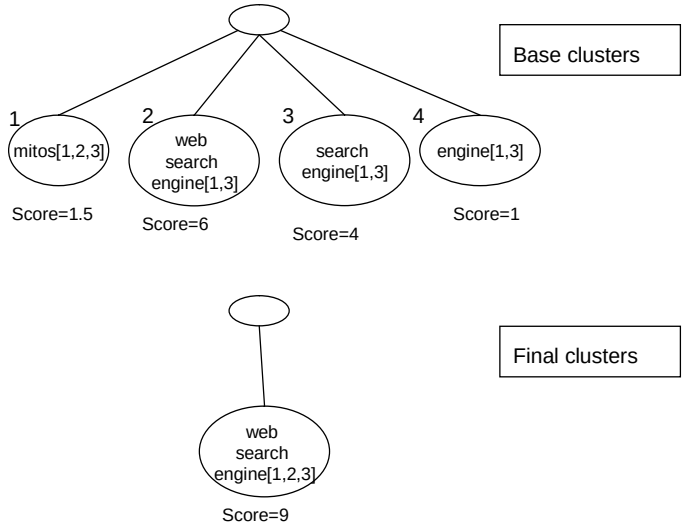


Figure 4.13: Base clusters and final cluster

The algorithm **identifyBaseClusters** is a recursive function. At the initial call it takes as input only the root of the suffix tree and then iteratively it takes the children of each node from all levels of the tree. Also, it takes as input a subphrase that is the labels' concatenation

from the root of the suffix tree to the first node of the subtree given as parameter. The effective length of the subphrase is the third argument effLen. The last argument is a list of the base clusters that have been identified and the argument index is related with this list.

Alg. $identifyBaseClusters$(SuffixTree ST, String subphrase, int effLen, int index, Base Clusters BC[])
(1) if $ST = \emptyset$ then return 0
(2) while(ST hasNext) do
(3) $STNode$:= pair of (label, suffixTreeNode)
(4) $W := STNode.label$
(5) $TN := STNode.suffixTreeNode$
(6) $docNum := |TN.docIds|$
(+)(STC+) $titleNum := |TN.titleIds|$ //for STC+
(8) $Phrase := \emptyset$
(9) if $subphrase \neq empty\ string$ then
(10) $Phrase.append(subphrase)$
(11) $Phrase.append(\text{""})$
(12) $Phrase.append(W)$
(13) if $docNum >= 2$ then
(14) $effLen := effLen + effectiveLength(W)$
(15)(STC) $score := (docNum) * funct\_f(effLen)$ //for STC
(15)(STC+) $score := (docNum + titleNum) * funct\_f(effLen)$ //for STC+
(16) set $score$ to TN
(17) set W to TN
(18) set $Phrase$ to TN
(19) add $STNode$ to $BC[index]$
(20) $Ch :=$ children of TN
(21) if $Ch \neq \emptyset$ then
(22) $index := identifyBaseClusters(Ch, W, effLen, index, BC)$
(23) return index

The algorithm **funct_f(int effectiveLength)** takes as input the effective length of a phrase and map this length to a number. For the original STC, it penalizes single words, is linear for phrases with effective length from two to six words, and is constant for bigger phrases.

The algorithm **funct_f(int effectiveLength)** for STC+ takes a different form. It penalizes single words and the phrases that their effective length is bigger that 4 words by returning a positive value that is less than one. Also, it is linear for phrases with effective length two to four words (returns the effective length of the phrase).

The algorithm **findDocsThatContain(String word, SuffixTree ST, List Docs)**

Alg. $funct\_f$(int effectiveLength) //for STC
(1) $score := 0.0$
(2) if $effectiveLength \leq 1$ then $score := 0.5$
(3) else if $effectiveLength \geq 2$ or $effectiveLength \leq 6$ then $score := effectiveLength$
(4) else if $effectiveLength > 6$ then $score := 7.0$
(6) return score

Alg. $funct\_f$(int effectiveLength) //for STC+
(1) $score := 0.0$
(2) if $effectiveLength \leq 1$ or $effectiveLength > 4$ then $score := 0.5$
(3) else if $effectiveLength \geq 2$ or $effectiveLength \leq 4$ then $score := effectiveLength$
(4) return score

takes as input a string, a suffix tree and a list of document ids. It traverses the suffix tree in order to find the number of documents in which the string appears. Documents which have been found already out of this suffix tree are considered so as to avoid duplicates.

The algorithm **mergeBaseClusters(Base Clusters BC[])** takes as input a list of base clusters and calculates similarity measure for each base cluster with its next base clusters in the list and find which of them can be merged. Returns merged clusters which is a map from each base cluster index to the base clusters' indexes that are going to be merged with it. Similarity threshold is defined to 0.5 for STC and to 0.4 for STC+.

Table 4.2 shows all the possible merges which where generated by calling the function mergeBaseClusters(Base Clusters BC[]) with input Table's 4.1 base clusters.

Index	Merged Base Clusters
1	2, 3, 4
2	1, 3, 4
3	1, 2, 4
4	1, 2, 3

Table 4.2: Merged Clusters

The algorithm **baseClustersOverlap(Document List listA, Document List listB, float simThreshold)** takes as input two sets of documents ids and calculates the overlap of two sets to each one and returns True if these percentages are over the similarity threshold given as the third argument.

Alg. $findDocsThatContain$(String word, SuffixTree ST, List Docs)
(1) if $ST = \emptyset$ then return 0
(2) $docCount := 0$
(3) while(ST hasNext) do
(4) $phrase :=$ next key of ST
(5) $TN :=$ tree node of $ST[phrase]$
(6) $tmpCount := 0$
(7) if $phrase$ contains $word$ then
(8) $sizeBefore := |Docs|$
(9) $Docs :=$unionOfLists($Docs$, Docs of TN)
(10) $sizeAfter := |Docs|$
(11) $tmpCount := sizeAfter - sizeBefore$
(12) else
(13) $Ch :=$ children of TN
(14) if $Ch \neq \emptyset$ then
(15) $tmpCount := findDocsThatContain(word, Ch, Docs)$
(16) $docCount := docCount + tmpCount$
(17) return $docCount$

Alg. $mergeBaseClusters$(Base Clusters BC[])
(1) $MC := \emptyset$
(2) for $i = 0$ to $|BC|$ do
(3) $indexes := \emptyset$
(4) add $(i, indexes)$ to MC
(5) for $i = 0$ to $|BC|$ do
(6) $DocList1 :=$ documents of $BC[i]$
(7) for $j = i + 1$ to $|BC|$ do
(8) $DocList2 :=$ documents of $BC[j]$
(9) if baseClustersOverlap($DocList1, DocList2, simThreshold$) = True then
(10) add j to $indexes$ of $MC[i]$
(11) add i to $indexes$ of $MC[j]$
(12) return MC

Alg. $baseClustersOverlap$(Document List listA, Document List listB, float simThreshold)
(1) $intersection := |listA \bigcap listB|$
(2) $overlap1 := |intersection|/|listA|$
(3) $overlap2 := |intersection|/|listB|$
(4) if $overlap1 > simThreshold$ and $overlap2 > simThreshold$ then return True
(5) else return False

The algorithm **createFinalClusters(Merged Clusters MC[], Base Clusters BC[])** takes as input a set of indexes that are related with the base clusters that can be merged and a set of base clusters. It creates a cluster for each group of base clusters that can be merged. Note that two base clusters that can not be merged directly it is possible to be merged in the same cluster if a third one base cluster can be merged with them. Each final cluster has as name the label of the highest score base cluster that it is constituted and as document set it has the union of its base clusters' document sets. Finally, the score of a final cluster is calculated based on its label and its new document set.

Alg. $createFinalClusters$(Merged Clusters MC[], Base Clusters BC[])
(1) $Clusters := \emptyset$
(2) $VI := \emptyset$ (visited Indexes)
(3) $tpmNode :=$ empty SuffixTreeNode
(4) for $index = 0$ to $|MC|$ do
(5) if VI contains $index$
(6) continue;
(7) add $index$ to VI
(8) $startNode :=$ SuffixTreeNode of $BC[index]$
(9) set $stratNode.Phrase$ to $tmpNode$;
(10) set $stratNode.Score$ to $tmpNode$;
(11) set $stratNode.TitleIds$ to $tmpNode$;
(12) set $stratNode.DocIds$ to $tmpNode$;
(13) $maxScoreDocNum := |tmpNode.docIds|$
(14) recursiveMergeIndexes($MC, index, tmpNode, maxScoreDocNum, VI$)
(15) $maxScorePhrase := tmpNode.Phrase$
(16) $len := effectiveLength(maxScorePhrase)$
(STC) $newScore := |tmpNode.docIds| * funct\_f(len)$//for STC
(STC+) $newScore := (|tmpNode.titleIds| + |tmpNode.docIds|) * funct\_f(len)$//for STC+
(18) add $(maxScorePhrase, newScore, tmpNode.docIds)$ to $Clusters$
(19) return $Clusters$

The algorithm **recursiveMergeIndexes(Merged Clusters MC[], int index, SuffixTreeNode tmpNode, int maxScoreDocNum, Base Clusters BC[])** takes as input the base clusters that can be merged into one cluster, an index to this structure, the current suffix tree node that is examined, the document set's size of the current highest score base cluster, the associated indexes with the base clusters structure that have already been visited and a set of base clusters. This method recursively merges the list of base clusters

that can be merged with the base cluster that is associated with the argument index.

```
Alg. recursiveMergeIndexes(MC[], int index, SuffixTreeNode tmpNode, int maxScoreDocNum, VI[], BC[])
(1)   mergedIndex := MC[index]
(2)   for i = 0 to |mergedIndex| do
(3)       if VI contains mergedIndex[i]
(4)           continue;
(5)       add mergedIndex[i] to VI
(6)       Node := SuffixTreeNode of BC[mergedIndex[i]]
(7)       score := score of Node
(8)       maxScore := score of tmpNode
(9)       if score >= maxScore
(10)          if score = maxScore
(11)              if maxScoreDocNum < Node.docIds
(12)                  maxScoreDocNum := Node.docIds
(13)                  set stratNode.Phrase to tmpNode;
(14)          else
(15)              maxScoreDocNum := Node.docIds
(16)              set stratNode.Phrase to tmpNode;
(17)          set stratNode.Score to tmpNode;
(18)      unionOfLists(tmpNode.docIds, Node.docIds)
(STC+)    unionOfLists(tmpNode.titleIds, Node.titleIds)//for STC+
(19)      recursiveMergeIndexes(MC, mergedIndex[i], tmpNode, maxScoreDocNum, VI)
```

After calling createFinalClusters(Merged Clusters MC[], Base Clusters BC[]) with input base clusters of Table 4.1 and merged base clusters of Table 4.2 only one cluster is created with label *web search engine* and document set 1, 2 and 3.

4.4 Preprocessing

The preprocessing times listed for STC+ in Table 4.3 are the times needed for retrieving titles from the data base, extracting the snippets from the plain text of each document and manipulating the titles and snippets from the lexical analyzer. Specifically, notice that preprocessing is the most expensive task (more than one magnitude, more expensive than the rest tasks). The second and the third column of the table (block stop words, stemming) are the options/configuration parameters for the *Lexical Analyzer* which implies how the titles and the extracted snippets will be manipulated.

Snippets generation is the most time-consuming task as plain texts are stored in txt files in the hard disk. Our original implementation (BT1) was trying to find the two sentences with the bigger number of appearances of the search keywords within them. The second implementation (BT2) stops when a second sentence, that contains the search keywords more times than the first, is found. The impact is a small reduction in execution time as Table 4.4 shows.

Top-K	block stop words	stem-ming	sentence separation	Preprocessing				construct suffix tree	prune tree	identify base clusters	merge base clusters	create final clusters	Total
				best text	lexical	titles	Total						
100	✓	x	✓	73%	26%	1%	0.654	0.025	0.039	0.016	0.044	0.0060	0.788
100	✓	✓	✓	75%	24%	1%	0.654	0.025	0.029	0.015	0.049	0.0070	0.784
200	✓	x	✓	87%	12%	1%	1.619	0.043	0.148	0.119	0.255	0.033	2.223
200	✓	✓	✓	86%	13%	1%	1.612	0.043	0.114	0.107	0.268	0.031	2.181
300	✓	x	✓	90%	9%	1%	2.621	0.053	0.366	0.257	0.658	0.076	4.035
300	✓	✓	✓	90%	9%	1%	2.647	0.051	0.214	0.298	0.661	0.073	3.95

Table 4.3: Execution times (in seconds) for query q= `kernel` with BT1 and similarity threshold 0.4.

Top-K	block stop words	stem-ming	sentence separation	Preprocessing				construct suffix tree	prune tree	identify base clusters	merge base clusters	create final clusters	Total
				best text	lexical	titles	Total						
100	✓	x	✓	66%	33%	1%	0.519	0.025	0.041	0.016	0.044	0.0070	0.656
100	✓	✓	✓	66%	33%	1%	0.526	0.024	0.032	0.015	0.048	0.0060	0.655
200	✓	x	✓	87%	13%	1%	1.472	0.039	0.113	0.068	0.099	0.025	1.822
200	✓	✓	✓	85%	14%	1%	1.451	0.077	0.063	0.059	0.102	0.02	1.78
300	✓	x	✓	92%	7%	1%	2.858	0.141	0.112	0.154	0.225	0.045	3.542
300	✓	✓	✓	90%	9%	1%	2.843	0.047	0.141	0.124	0.235	0.038	3.435

Table 4.4: Execution times (in seconds) for query q= `kernel` with BT2 and similarity threshold 0.4.

Table 4.5 is a comparison of clustering results for seven queries using two different approaches for snippet generation, BT1 and BT2. It shows the number of clusters produced using BT1 and BT2 and the number of common cluster labels between them.

The execution times for the best text extraction include the time needed for stemming each word of the cached copy of a document. Without stemming but following a matching approach that identifies substrings, best text extraction becomes four times faster.

	BT1	BT2	
Query	clusters	clusters	common labels
kernel	18	15	12
information retrieval	20	18	10
php examples	16	19	15
forthnet	20	22	19
yannis tzitzikas	33	14	11
crete	18	14	11
java tutorial	7	9	5

Table 4.5: Differences in number of clusters between the best text approaches and number of common labels.

4.4.1 Problems in Detecting the Right Sentence Boundaries

During the preprocessing of the snippets there are some cases that make the identification of sentence boundaries difficult and are listed below.

- File names (e.g. proc.c, proc.h)
- Abbreviations (e.g. FORTHnet S.A)
- Numbers (e.g. 1.5)
- Times (e.g. 12:27:52)
- Paths (e.g. /src/kernel/proc.c)
- E-mail (e.g. stella.kop@gmail.com)
- URL (e.g. www.w3.org)
- Human names (e.g. Y . Marketakis , N . Armenatzoglou and Y . Tzitzikas)

Sentence boundaries selection was implemented by using Java API functions [2].

This method does not work properly for all these cases. For example, the string `", Y. Marketakis , N. Armenatzoglou and Y. Tzitzikas Mitos : Design and Evaluation"` is separated into:

```
, Y.
Marketakis , N.
```

[2] http://java.sun.com/docs/books/tutorial/i18n/text/sentence.html

```
Armenatzoglou and Y.
Tzitzikas Mitos :  Design and Evaluation
```

Moreover, after the processing of the lexical analyzer the output is the following sentences:

```
marketakis
armenatzoglou
tzitzikas mitos design evaluation
```

It works right for paths like `/src/kernel/proc.c` but not for path `../adonomics.ps` as it is separated into `..` and `/adonomics.ps`. Also, it works right for file names, abbreviations, numbers, times, e-mail and urls.

The following examples are cases that there is no change:

```
Information Systems Laboratory:  People, Yannis Tzitzikas
FORTH - ICS: Announcements
Java 2 Platform SE v1.4.0:  Uses of Interface javax.xml.transform.sax.TransformerHandler
Java Object Serialization Specification:  - Example of Serializable Fields
Creating a GUI with JFC/Swing:  Indexes of Examples
Course Content in English (U.Crete, CS-225)
User-Level Atomic Operations
```

Except from human names and abbreviations the other cases does not offer valuable information. This is an issue for further research. One could apply techniques like those proposed for Named Entity Recognition(NER) in [29, 25].

4.5 Combining Results Clustering with Metadata Exploratory through Dynamic Taxonomies

`FleXplorer` [33] is a main memory API (Application Programmatic Interface) that allows managing (creating, deleting, modifying) terms, taxonomies, facets and object descriptions. It supports both finite and infinite terminologies (e.g. numerically-valued attributes). In addition it supports explicitly and intentionally defined taxonomies. Examples of the

former include classification schemes and thesauri, while examples of the latter include hierarchically organized intervals (based on the *inclusion* relation). Regarding, interaction, the framework provides methods for setting (resp. computing) the focus (resp. zoom-in points). In addition, the framework allows materializing on demand the relationships of a taxonomy, even if the domain is infinite and intentionally defined (e.g. between numbers, intervals, etc).

`FleXplorer` is used by `Mitos` providing the *Faceted Taxonomies* interactive scheme, for offering general purpose browsing and exploration services. Currently, only some general and content-independent facets are supported. Specifically, the facets/taxonomies, that are created and presented to the users, are:

- web domain, a hierarchy is defined (e.g. csd.uoc.gr < uoc.gr < gr),
- format type (e.g. pdf, html, doc, etc), no hierarchy is created in this case,
- language of a document based on the encoding of a web page (e.g. Greek, English, Latin-1) and
- (modification) date hierarchy

The Clustering component is called by `FleXplorer` and the derived clusters are considered as an additional facet. Tree-based presentation of the clustering results is suitable for integrating this functionality to `FleXplorer`.

Figure 4.14 shows the Faceted Taxonomies results for the query=`computer science department`.

To the best of our knowledge, there are no other WSEs that offer the same kind of information/interaction. A somehow related interaction paradigm that involves clustering is Scatter/Gather [11, 18]. This paradigm allows the users to select clusters, subsequently the documents of the selected clusters are clustered again, the new clusters are presented, and so on. This process can be repeated until individual documents are reached. However, for very big answer sets, the initial clusters apart from being very expensive to compute on-line, will also be quite ambiguous and thus not very helpful for the user. Our approach alleviates this problem, since the user can restrict his focus through the available metadata, to a size that allows deriving more specific and informative cluster labels.

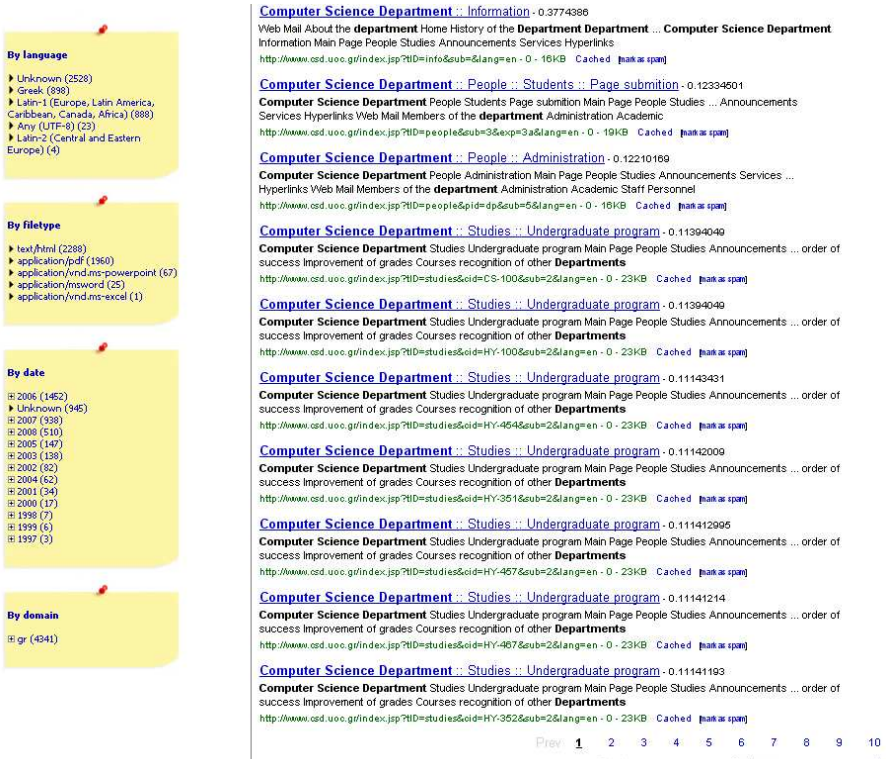


Figure 4.14: Faceted Taxonomies interface on `Mitos`

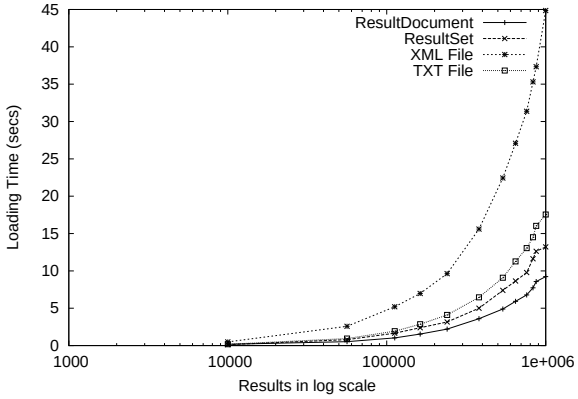


Figure 4.15: Time to load results to `FleXplorer`

4.5.1 On-Demand Integration

Dynamic taxonomies can load and handle thousands of objects very fast as it is proved from Figure's 4.15 example that shows the loading time of the top-K answer for various values of K: from 10^4 to 10^6. As the loading time depends on the format employed (and the associated parsing costs), the figure reports the loading times for four different formats, namely: (a) JDBC ResultSet, (b) XML, (c) a (proprietary) TXT-based format, and (d) a main memory format, called `ResultDocument` that is provided by the `FleXplorer` API and is used in `Mitos`.

However, the application of results clustering on thousands of snippets would have the following shortcomings:

- *Inefficiency.* Real-time results clustering is feasible with hundreds (not thousands) of snippets.

- *Low cluster label quality.* The resulting labels would probably have low quality, since they would be too general.

To this end we have developed a *dynamic (on-demand) integration* approach. The idea is to *apply the result clustering algorithm only on the top-C (for C usually 100) snippets of the current focus.* This approach not only can be performed fast, but it is expected to return more specific (informative/predictive) cluster labels.

Let q be the user query and let $Ans(q)$ be the answer of this query. We shall use A_f to denote top-K (usually $K < 10000$) objects of $Ans(q)$ and A_c to denote top-C (usually C

equals 100) objects of $Ans(q)$. Clearly, $A_c \subseteq A_f \subseteq Ans(q)$. In particular, the steps of the process are the following:

(1) The snippets of the elements of A_c are generated.

(2) The results clustering is applied on the elements of A_c. A cluster label tree *clt* is generated.

(3) The set of A_f (with their metadata), as well as *clt*, are loaded to `FleXplorer`, a module for creating and managing the faceted dynamic taxonomy. As the facet that corresponds to automatic clustering includes only the elements of A_c, we create an additional artificial cluster label, named "REST" where we place all objects in $A_f \setminus A_c$ (i.e. it will contain $K - C$ objects).

(4) `FleXplorer` computes and delivers to the GUI the (immediate) zoom points.

The user can start browsing by selecting the desired zoom point(s). When the user selects a zoom point or submits a new query, the steps (1)-(4) are performed again.

4.5.2 Application over `Mitos`

In our implementation, we have chosen not to apply the re-clustering process (i.e. steps (1) and (2) of the on-demand algorithm), when the user interacts with the clustering facet. This behavior is more intuitive, since it preserves the clustering hierarchy while the user interacts with the clustering facet and does not frustrate the user with unexpected results. Furthermore, if the user is not satisfied by the available cluster labels for the top-C objects of the answer, he can enforce the execution of the clustering algorithm for the next top-C objects. This feature is available by pressing the *REST* zoom-in point, which as already mentioned, keeps pointers to $K-C$ objects. These objects are not included in the extensions of the original cluster labels. Figure 4.16 shows an indicative screendump of the Web-based GUI. Notice the *REST* zoom-in point in the *By clustering* facet.

4.5.3 Incremental Evaluation Algorithm

Here we present an incremental approach for exploiting past computations and results. Let A_f be the objects of the current focus. If the user selects a zoom point he moves to a

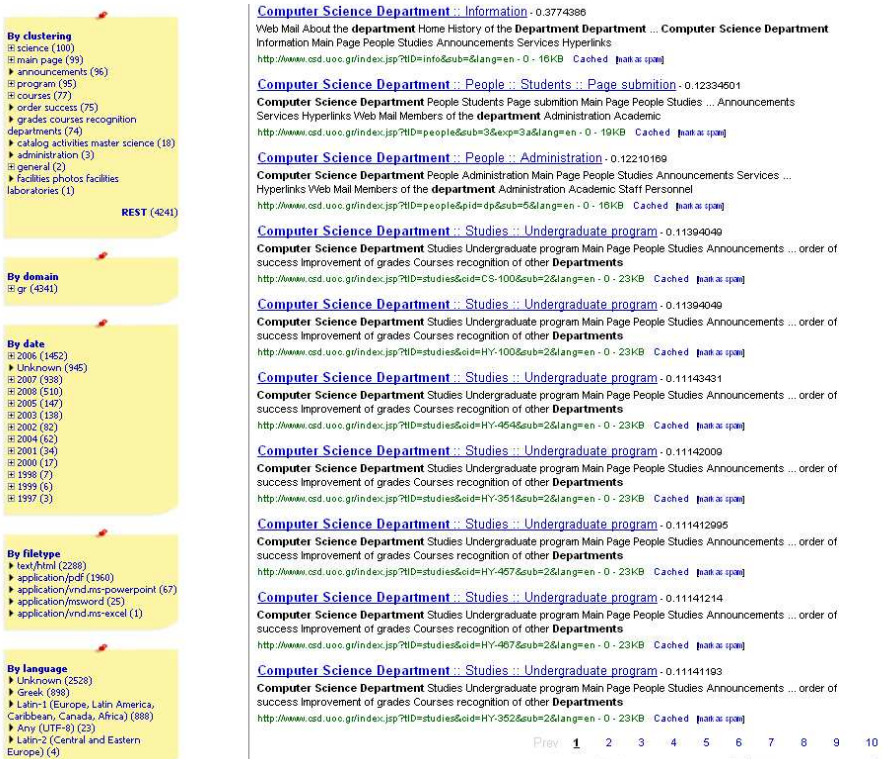


Figure 4.16: Faceted Taxonomies based on Clustering interface on `Mitos`

different focus. Let A'_f denote the top-K elements of the new focus, and A'_c the top-C of the new focus. The steps of the algorithm follow.

(1) We set $A_{c,new} = A'_c \setminus A_c$ and $A_{c,old} = A_c \setminus A'_c$, i.e. $A_{c,new}$ is the set of the new objects that have to be clustered, and $A_{c,old}$ is the set of objects that should no longer affect clustering.

(2) The snippets of the objects in $A_{c,new}$ are generated (those of $A_{c,old}$ are available from the previous step). Recall that snippet generation is expensive.

(3) NM-STC is applied *incrementally* to $A_{c,new}$.

(4) The new cluster label tree clt' is loaded to `FleXplorer`.

(5) `FleXplorer` computes and delivers to the GUI the (immediate) zoom points for the focus with contents A'_f.

Let's now focus on Step (3), i.e. on the incremental application of NM-STC. Incremental means that the previous suffix tree sf is preserved. Specifically, we extend sf with the suffixes of the elements in the titles/snippets of the elements in $A_{c,new}$, exploiting the incremental nature of STC. Let sf' denote the extended suffix tree. To derive the top scored labels, we have to score again all nodes of the suffix tree. However we should not take into account objects that belong to $A_{c,old}$. Specifically, scoring should be based on the extension of the labels that contain elements of A'_c only.

The preserved suffix tree can be either the initial suffix tree or the pruned suffix tree. Each node of the initial tree corresponds to a single word, while the pruned tree is more compact in the sense that if a node contains only one child node and both nodes contain the same objects, they are collapsed to one single node that has as label the concatenation of the labels of the constituent nodes. Scoring is done over the pruned suffix tree. However to add and delete objects to/from a pruned suffix tree sometimes requires "splitting" nodes (due to the additions) and pruning extra nodes (due to the deletions). On the other hand, if the unpruned suffix tree is preserved, then additions and deletions are performed right away and pruning takes place at the end. Independently of the kind of the preserved suffix tree, below we discuss two possible approaches for updating the suffix tree:

- *Scan*-approach

  We scan the nodes of the suffix tree sf' and delete from their extensions all elements that belong to $A_{c,old}$.

- *Object-to-ClusterLabel Index*-approach

  An alternative approach is to have an additional data structure that for each object o in A_c it keeps pointers to the nodes of the suffix tree to whose extension o belongs. In that case we don't have to scan the entire suffix tree since we can directly go to the nodes whose extension has to be reduced. The extra memory space for this policy is roughly equal to the size of the suffix tree. However the suffix tree construction process will be slower as we have to maintain the additional data structure too.

We have to note that sf can be considered as a cache of snippets and recall that snippet generation is more expensive than clustering. The gained speedup is beneficial both for a stand-alone WSE as well for a Meta WSE, since fetching and parsing of snippets are reused. The suffix tree sf has to be constructed from scratch whenever the user submits a new query and is incrementally updated while the user browses the information space by selecting zoom points. If the suffix tree size exceeds a threshold, we delete it and we reconstruct it based on the snippets of the top-C elements of the new focus.

4.5.3.1 Using the initial suffix tree

For this method the initial (unpruned) suffix tree is stored in the main memory. The clustering algorithm inserts into this unpruned suffix tree the suffixes that corresponds to documents of $A_{c,new}$. In the non-incremental algorithm the next step would be the pruning of the tree but now we must first delete the old documents from the nodes of the tree in order pruning to be consistent. Reduction of old documents is done using *Scan*-approach as described above. After the elimination of the old documents a copy of this unpruned suffix tree is stored. The other steps remain the same.

Figure 4.17 (A1) shows the constructed suffix tree after the insertion of the following documents (A_c):

Title 1: `a b`

Snippet 1: `c a b`

Title 2: `a e`

Snippet 2: `c a e`

We suppose that $A_{c'}$ consists of the following documents:

Title 1: `a b`

Snippet 1: `c a b`

Title 3: `a b`

Snippet 3: `c g`

Figure 4.17 (A2) shows the constructed suffix tree after the insertion of $A_{c,new}$={3} into the suffix tree of Figure 4.17 (A1). Figure 4.17 (A3) shows the generated suffix tree after the elimination of $A_{c,old}$={2} (note that after this stage a copy of this suffix tree is stored in the main memory) and finally Figure 4.17 (A4) shows the pruned suffix tree.

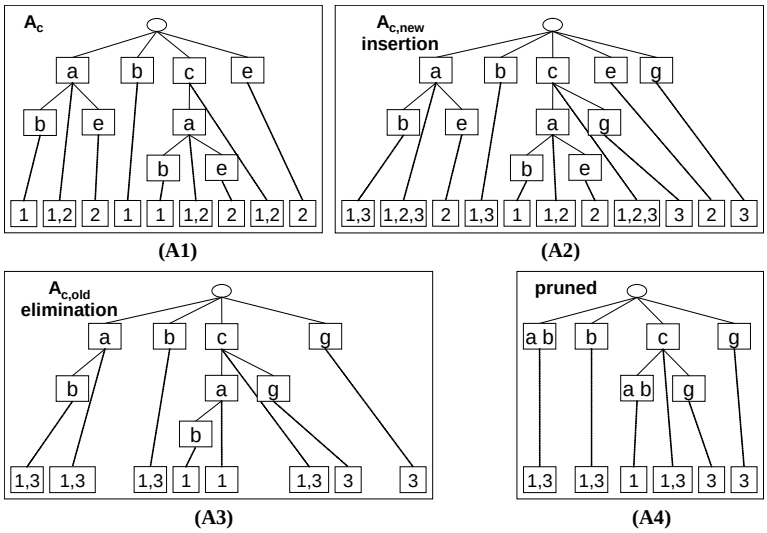


Figure 4.17: Example of using the initial suffix tree

4.5.3.2 Using the pruned suffix tree

For this method the pruned suffix tree is stored in the main memory as it is in a compact form so it has less memory requirements. The clustering algorithm inserts into this pruned suffix tree the suffixes that corresponds to documents of $A_{c,new}$. This means that some nodes

are going to be divided. During this process of inserting the new suffixes a data structure is maintained that maps each document to the nodes it belongs. Now the old document ids are reduced using the additional data structure. Next the tree is pruned, a copy of this pruned suffix tree is stored and then the algorithm continues as the non-incremental.

Figure 4.18 (A1) shows the preserved pruned suffix tree and the additional data structure (Object Map) for documents(titles/snippets) 1 and 2 while Figure 4.18 (A2) shows the constructed suffix tree after the insertion of $A_{c,new}$={3} into the suffix tree of Figure 4.18 (A1) and the updated Object Map. Figure 4.19 (A3) shows the elimination of $A_{c,old}$={2} based on the Object Map (red color denotes the deletion of the specific element) and Figure 4.19 (A4) shows the generated suffix tree and the Object Map after the deletion of the entries that corresponds to the $A_{c,old}$ documents. Finally, Figure 4.20 (A5) shows the pruned suffix tree and the Object Map, both of them will be stored in the main memory.

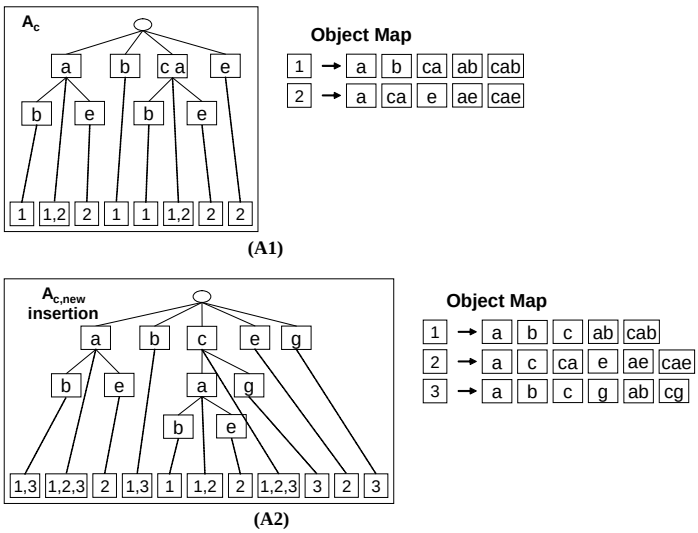


Figure 4.18: Example of using the pruned suffix tree

4.5.4 Experimental Results

4.5.4.1 Clustering Performance

It is worth noticing that the most time consuming subtask is not the clustering itself but the extraction of the "best text" (snippet) from the cached copies of textual contents of the

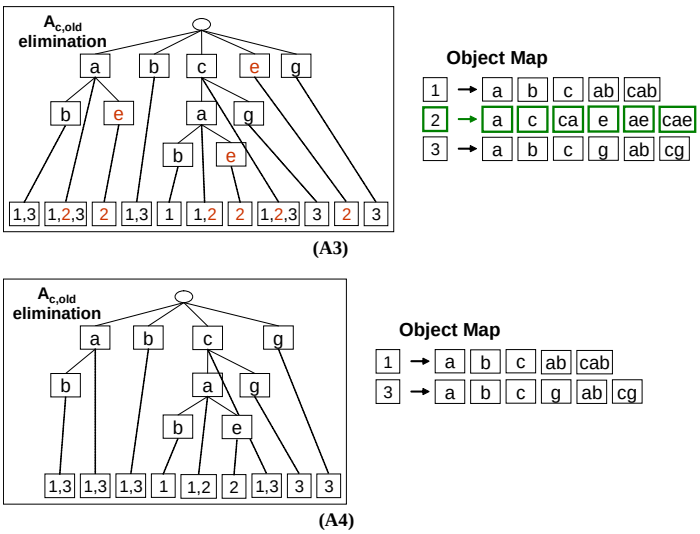


Figure 4.19: Elimination of $A_{c,old}$

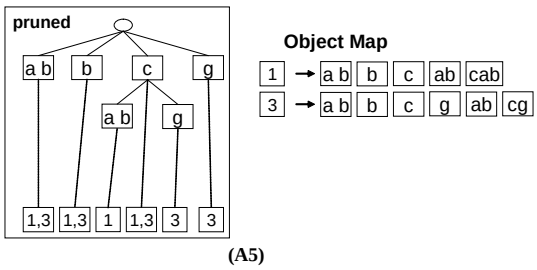


Figure 4.20: Example of using the pruned suffix tree

pages [3]. To measure the performance of the clustering algorithm and the snippet generation, we selected 16 queries and we counted the average times to generate and cluster the top-{100, 200, 300, 400, 500} snippets. All measurements were performed using a Pentium IV 4 GHz, 2 GB RAM, Linux Debian.

Measured Task	**100**	**200**	**300**	**400**	**500**
Time to generate snippets	0.793	1.375	1.849	2.268	2.852
Time to apply STC	0.138	0.375	0.833	1.494	2.303
Time to apply NM-STC	0.117	0.189	0.311	0.449	0.648

Table 4.6: Top-C Snippet Generation and Clustering Times (in seconds)

Table 4.6 shows snippet generation times and the clustering algorithms performance. Both times are in seconds. Notice that snippet generation is a very slow operation and is the bottleneck in order to provide fast on-demand clustering, for a big top-C number (C bigger than 100). We should mention though, that our testbed includes a rather big number of large sized files (i.e. pdf, doc, ppt), which hurt snippet generation times. Furthermore, notice that NM-STC is at least two times faster than STC. This is because NM-STC does not have to intersect and merge base clusters.

4.5.4.2 Overall Performance

In this experiment we measured the cost of coupling the cluster generation times (i.e. snippet generation and clustering algorithm execution) with the dynamic taxonomies times (i.e. the times to compute the zoom points and the times to load the new clustering labels to the corresponding facet). Moreover we compare the non-incremental with one incremental algorithm, which preserves the initial suffix tree and the elimination of old objects is done using the Scan-approach. The scenario we used includes: (a) the execution of the query *crete* which returns 4067 results, (b) the expansion of the *.gr* zoom point of the *By domain* facet and the selection of the *uoc.gr* (1277) zoom-in point from the hierarchy revealed from the expansion, and (c) the selection of the *text/html* (807) zoom-in point of the *By filetype* facet. Let c_a, c_b and c_c be snippets of the $top - C$ elements in the steps (a), (b) and (c) respectively. Figure 4.21 shows the facet terms after steps (a), (b) and (c), as they are

[3]The snippets in our experiments contain up to two sentences where the query terms appear most times and each one consists of 11 words maximum.

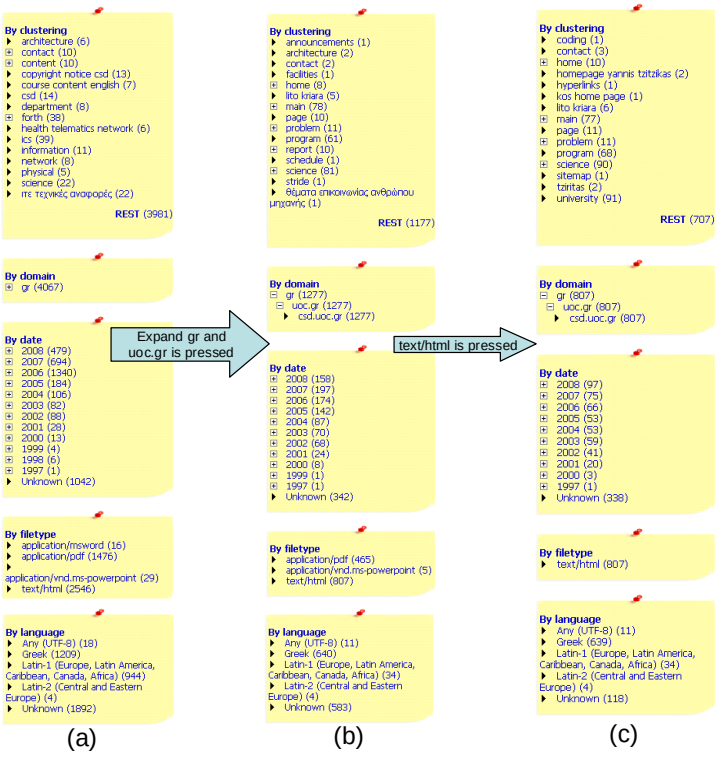


Figure 4.21: Steps (a)-(c) of running scenario

displayed in the left bar of the WSE GUI. We set $K = 10000$ (i.e. the whole answer set is loaded) and repeated the above steps for the following values of C:100, 200 ... 500. We do not measure the cost of the query evaluation time. In all experiments `FleXplorer` computes count information.

Table 4.7 shows the intersection of A_c and A'_c for steps (a), (b) and (c) and the execution times that correspond to the integration of `FleXplorer` and results clustering when the non-incremental NM-STC and an incremental approach of NM-STC is used, for the $top-C$ elements. It is evident that for top-100 and top-200 values, the results are presented to the user almost instantly (around 1 second), making the proposed on demand clustering method suitable as an online task. Moreover we can see that there is a linear correlation between time cost and the top-C value. Finally calculating and loading clusters for the top-500 documents, costs around 3 seconds making even big top-C configurations a feasible configuration.

Comparing the incremental and the non-incremental algorithm, we observe a significant speedup whenever the overlap is more than 50%, for our scenario. At step (a) the suffix tree construction is the same for both algorithms as the suffix tree sf has to be constructed from

	Step (a)	Step (b)	Step (c)
top-100	$\|c_a\| = 100$	$\|c_a \cap c_b\| = 43$, overlap=43%	$\|c_b \cap c_c\| = 85$, overlap=85%
Non-Incr.	0.914	0.443	0.204
Incr.	0.931	0.431	0.101
top-200	$\|c_a\| = 200$	$\|c_a \cap c_b\| = 71$, overlap=35.5%	$\|c_b \cap c_c\| = 113$, overlap=56.5%
Non-Incr.	1.266	1.245	0.789
Incr.	1.245	0.965	0.68
top-300	$\|c_a\| = 300$	$\|c_a \cap c_b\| = 74$, overlap=24.6%	$\|c_b \cap c_c\| = 201$, overlap=67.7%
Non-Incr.	1.676	2.534	1.383
Incr.	1.65	2.527	0.761
top-400	$\|c_a\| = 400$	$\|c_a \cap c_b\| = 85$, overlap=21.5%	$\|c_b \cap c_c\| = 252$, overlap=63%
Non-Incr.	2.246	3.067	1.944
Incr.	2.118	3.335	0.942
top-500	$\|c_a\| = 500$	$\|c_a \cap c_b\| = 97$, overlap=19.4%	$\|c_b \cap c_c\| = 324$, overlap=64.8%
Non-Incr.	2.483	3.495	2.001
Incr.	2.493	3.652	0.751

Table 4.7: Top-C Comparison of Incremental/Non-Incremental Algorithms (in seconds)

scratch. For step (b) there are small variations due to the small overlap, so the time saved from the snippets generation/parsing is compensated by the time needed for eliminating old objects. Specifically, the incremental algorithm is faster for the top-200 case and slower for the top-{400, 500} cases which have the lowest overlap. For the other cases performance is almost the same. Notice that although the top-100 case has the biggest overlap of all, there are no differences in the execution time of the two algorithms. This is probably due to the fact that the overlapping documents have fast snippet generation times, while the rest are big sized. At step (c) the benefit from the incremental approach is clear, since it is almost twice as fast as the non incremental one. Specifically, the best speedup is in the case of top-500, where overlap reaches 65% and the execution time of the non-incremental is 2.001, while for the incremental is just 0.751.

4.6 Admin Parameters

Administrator's parameters for clustering are stored in the database and can be changed only by authenticated users [4]. Note that the clustering algorithm that is currently used on `Mitos` is also specified by a parameter which is called *Name hierarchy*.

The following parameters are used by Snippet-based approaches:

- K : number of top elements of the answer to cluster

[4] http://groogle.csd.uoc.gr:8080/mitos/admin/

- LL_{max} : minimum number of words that a cluster label can contain
- LL_{min} : maximum number of words that a cluster label can contain
- NC_{max} : maximum number of the generated clusters

4.7 Application over Google

Our approaches were also applied over the search results of Google. A Google parser was implemented that is based on an HTML parser. Google pages are requested with 100 results per page which is the maximum number. The information we gather for each result is its title, snippet, address links and file length. Titles and snippets are the input data for the clustering process, while the address links and file length are used only at the presentation layer. At the GUI layer the user can select the number of results which can vary from 100 to 500 and the clustering algorithm he wants.

Google groups together its News, Books, Videos, Blogs, Images, Shopping results. Usually only titles are provided for these results. We exploit all these results except the Images results. Also, in order to exploit the grouping of these hits we observe that their titles consist of the title of the group concatenated with the title of the hit. For example, if we have the following Book results:

```
Books by Nikos Kazantzakis
  Zorba the Greek
  At the Palaces of Knossos:  A Novel
  The Last Temptation of Christ
```

the titles that will be used are:

```
Books Zorba the Greek
Books At the Palaces of Knossos:  A Novel
Books The Last Temptation of Christ
```

The name of the group (e.g. Books) is usually followed by the query words (e.g. Nikos Kazantzakis), so we do not use them as they will be excluded at step (2) of the algorithm. For example, `News results for Obama` will be reduced to `News`.

Note that each of these special groups are counted as one result for Google, but our

Google parser identifies each constituent title as a separate result which means that the number of the requested results can be 100 and the clustered documents more than 100.

Figures 4.22 and 4.23 show the user interface of Clustering over Google[5] and the clusters derived when submitting the query q= `Eleftherios Venizelos` and q= `Nikos Kazantzakis` respectively.

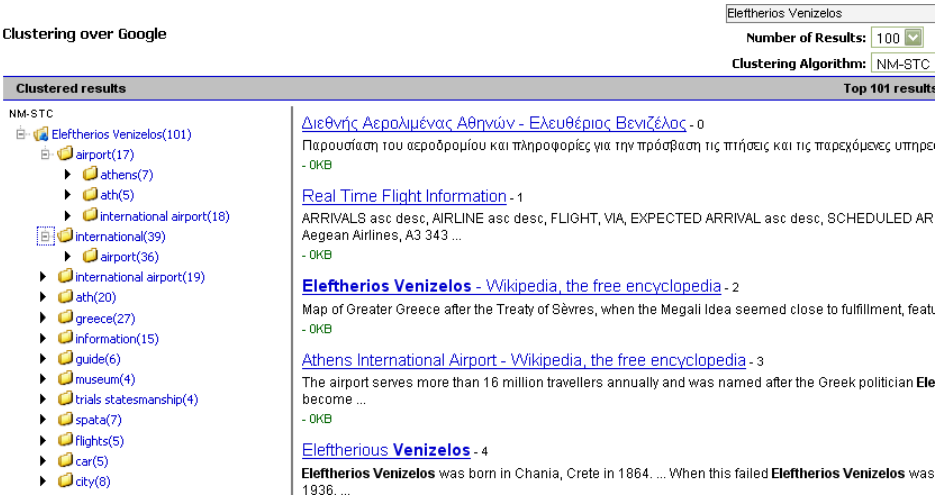


Figure 4.22: Clustering over Google user interface

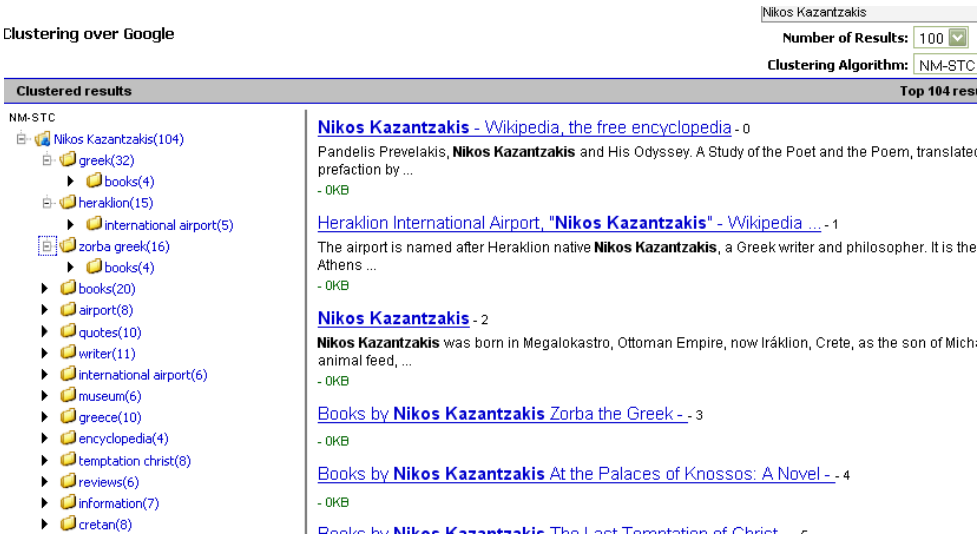


Figure 4.23: Clustering over Google user interface

[5]http://groogle.csd.uoc.gr:8080/google/google.jsp

Chapter 5

Conclusion

5.1 Synopsis

Web Search Engines typically return a ranked list of documents that are relevant to the query submitted by the user. The absence of concise overviews and the inability of the user to determine his information need accurately, make it difficult for the user to satisfy his information needs. Results clustering is a solution which provides a quick overview of the search results.

Results clustering can be applied either to the original documents (like in [11, 18, 23]), or to their (query-dependent) snippets (as in [38, 30, 15, 41, 17, 34]).

Clustering should provide each generated cluster with a cluster label that characterize the contents of its objects in order to allow users to detect what they need quickly. This task of deriving readable and meaningful (single-word or multiple-word) names for clusters is called *cluster labeling* and is very difficult as labels must be predictive, descriptive, concise and syntactically correct. Some clustering algorithms [15, 12, 37] use internal or external sources of knowledge so as to identify significant words/phrases that represent the contents of the retrieved documents or to enrich the extracted words/phrases in order to optimize the clustering and improve the quality of cluster labels.

In this thesis we relied on *Suffix Tree Clustering (STC)* which is a clustering technique where search results (mainly snippets) are clustered fast (in linear time), incrementally, and each cluster is labeled with a common phrase. Other advantages of STC is that it uses

phrases (rather than words) and that it allows clusters to overlap. Based on this algorithm we introduced (a) a variation of the STC, called STC+, with a scoring formula that favors phrases that occur in document titles and differs in the way base clusters are merged, and (b) a novel algorithm, called NM-STC, that adopts a different scoring formula, does not merge clusters and results in hierarchically organized labels.

The comparative evaluation of the three algorithms showed that NM-STC is (two to three times) faster than STC and STC+. Moreover, the empirical evaluation conducted with the participation of 11 people showed that both STC+ and NM-STC are significantly more preferred than STC, and that STC+ is slightly more preferred than NM-STC. The majority of the users prefer (a) hierarchically organized labels, (b) labels comprising one to three words, and (c) 10-15 clusters.

A complementary approach for the presentation of web search results is to exploit the various metadata that are available to WSE (like domain, dates, language, document type, etc) in the context of the interaction paradigm of faceted and dynamic taxonomies. We have proposed an on-demand integration of content-based results clustering with dynamic taxonomies. To this end we have exploited the incremental nature of STC and presented an incremental approach for exploiting past computations and results. The evaluation of the incremental algorithms showed that the benefit from the snippet caching is considerable as the result set is restricted since snippet generation is more expensive than clustering.

5.2 Directions for further work and research

The current work can be extended in order to further improve the quality of cluster labels. One direction is to investigate the applicability of Named Entities Recognition(NER) techniques.

Regarding efficiency, a possible optimization for STC+ and NM-STC could be to prune two nodes only when their labels appear in the same document titles and not in the same documents which means that a label appears either in the document title or in the document snippet.

Another issue for further research is how to improve the performance of the incremental algorithms presented and finally the investigation of what top-C value most users prefer.

Bibliography

[1] *Carrot*^{2} search engine. http://www.carrot2.org.

[2] Clusty search engine. http://clusty.com.

[3] Mitos search engine. http://groogle.csd.uoc.gr:8080/mitos/.

[4] Quintura search engine. http://www.quintura.com.

[5] Snaket search engine. http://snaket.di.unipi.it/.

[6] Vivisimo search engine. http://vivisimo.com.

[7] N. Agarwal, E. Haque, H. Liu, and L. Parsons. A subspace clustering framework for research group collaboration. *International Journal of Information Technology and Web Engineering*, 1(1):35–58, 2006.

[8] H. Andreka, M. Ryan, and P.-Y. Schobbens. Operators and Laws for Combining Preference Relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.

[9] H. O. Borch. On-Line Clustering of Web Search Results. Master thesis, Norwegian University of Science and Technology, July 2006.

[10] D. Crabtree, X. Gao, and P. Andreae. Improving Web Clustering by Cluster Selection. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI '05)*, pages 172–178, Compiegne, France, September 2005.

[11] D.R. Cutting, D. Karger, J.O. Pedersen, and J.W. Tukey. Scatter/Gather: a cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '92)*, pages 318–329, Copenhagen, Denmark, June 1992.

[12] W. Dakka and P.G. Ipeirotis. Automatic Extraction of Useful Facet Hierarchies from Text Databases. In *Proceedings of the 24th International Conference on Data Engineering (ICDE '08)*, pages 466–475, Cancún, México, April 2008.

[13] J. C. de Borda. Memoire sur les Elections au Scrutin, 1781. Histoire de l'Academie Royale des Sciences, Paris.

[14] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[15] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. In *Proceedings of the 14th international conference on World Wide Web (WWW '05) - Special interest tracks and posters*, volume 5, pages 801–810, May 2005.

[16] B.C.M. Fung, K. Wang, and M. Ester. Hierarchical Document Clustering Using Frequent Itemsets. In *Proceedings of the SIAM International Conference on Data Mining*, volume 30, San Francisco, CA, USA, May 2003.

[17] F. Gelgi, H. Davulcu, and S. Vadrevu. Term ranking for clustering web search results. In *10th International Workshop on the Web and Databases (WebDB '07)*, Beijing, China, June 2007.

[18] M.A. Hearst and J.O. Pedersen. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR '96)*, pages 76–84, Zurich, Switzerland (Special Issue of the SIGIR Forum), August 1996.

[19] J. Janruang and W. Kreesuradej. A New Web Search Result Clustering based on True Common Phrase Label Discovery. In *Proceedings of the International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA/IAWTIC '06)*, Washington, DC, USA, November 2006.

[20] M. Käki. Findex: properties of two web search result categorizing algorithms. In *Proc. IADIS Intl. Conference on World Wide Web/Internet*, Lisbon, Portugal, October 2005.

[21] S. Kopidaki, P. Papadakos, and Y. Tzitzikas. STC+ and NM-STC: Two Novel Online Results Clustering Methods for Web Searching. In *Proceedings of the 10th International Conference on Web Information Systems Engineering (WISE '09)*, October 2009.

[22] T.K. Landauer, P.W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25:259–284, 1998.

[23] Y.S. Maarek, R. Fagin, I.Z. Ben-Shaul, and D. Pelleg. Ephemeral document clustering for web applications. *IBM Research Report RJ 10186*, April 2000.

[24] C.D. Manning, P. Raghavan, and H. Schtze. *Introduction to information retrieval*. Cambridge University Press New York, NY, USA, 2008.

[25] I. Michailidis, K. Diamantaras, S. Vasileiadis, and Y. Frère. Greek named entity recognition using Support Vector Machines, Maximum Entropy and Onetime. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pages 45–72, Genova, Italy, 2006.

[26] P. Papadakos, S. Kopidaki, N. Armenatzoglou, and Y. Tzitzikas. Exploratory Web Searching with Dynamic Taxonomies and Results Clustering. In *Proceedings of the 13th European Conference on Digital Libraries (ECDL '09)*, Corfu, Greece, Sept.-Oct. 2009.

[27] P. Papadakos, Y. Theoharis, Y. Marketakis, N. Armenatzoglou, and Y. Tzitzikas. Mitos: Design and Evaluation of a DBMS-based Web Search Engine. In *Procs of the 12th Pan-Hellenic Conference on Informatics (PCI '08)*, Greece, August 2008.

[28] P. Papadakos, G. Vasiliadis, Y. Theoharis, N. Armenatzoglou, S. Kopidaki, Y. Marketakis, M. Daskalakis, K. Karamaroudis, G. Linardakis, G. Makrydakis, V. Papathanasiou, L. Sardis, P. Tsialiamanis, G. Troullinou, K. Vandikas, D. Velegrakis, and Y. Tzitzikas. The Anatomy of Mitos Web Search Engine. *CoRR, Information Retrieval*, abs/0803.2220, 2008. Available at http://arxiv.org/abs/0803.2220.

[29] Y. Ramírez-Cruz and A. Pons-Porrata. Spanish Nested Named Entity Recognition Using a Syntax-Dependent Tree Traversal-Based Strategy. In *Proceedings of the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, pages 144–154, Atizapán de Zaragoza, Mexico, October 2008. Springer-Verlag Berlin, Heidelberg.

[30] J. Stefanowski and D. Weiss. Carrot2 and language properties in web search results clustering. In *Proceedings of the International Atlantic Web Intelligence Conference*, Madrid, Spain, May 2003.

[31] J. Stefanowski and D. Weiss. Comprehensible and Accurate Cluster Labels in Text Clustering. In *8th International Conference on Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications) - RIAO 2007*, Pittsburgh, PA, USA, May-June 2007.

[32] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. *KDD Workshop on Text Mining*, 34:35, 2000.

[33] Y. Tzitzikas, N. Armenatzoglou, and P. Papadakos. FleXplorer: A Framework for Providing Faceted and Dynamic Taxonomy-Based Information Exploration. In *Database and Expert Systems Application (FIND '08 at DEXA '08)*, pages 392–396, Torino, Italy, 2008.

[34] J. Wang, Y. Mo, B. Huang, J. Wen, and L. He. Web Search Results Clustering Based on a Novel Suffix Tree Structure. In *Procs of 5th International Conference on Autonomic and Trusted Computing (ATC '08)*, volume 5060, pages 540–554, Oslo, Norway, June 2008.

[35] Y. Wang and M. Kitsuregawa. Use link-based clustering to improve Web search results. In *Proceedings of the Second International Conference on Web Information System Engineering (WISE '01)*, Kyoto, Japan, December 2001.

[36] D. Weiss and J. Stefanowski. Web search results clustering in Polish: Experimental evaluation of Carrot. In *Intelligent Information Processing and Web Mining: Proceedings of the International IIS: IIPWM '03*, Zakopane, Poland, June 2003.

[37] D. Xing, G.R. Xue, Q. Yang, and Y. Yu. Deep classifier: automatically categorizing search results into large-scale hierarchies. In *Proceedings of the international conference on Web Search and Web Data Mining (WSDM '08)*, pages 139–148, Palo Alto, California, USA, February 2008.

[38] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR '98)*, pages 46–54, Melbourne, Australia, August 1998.

[39] O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31(11-16):1361–1374, 1999.

[40] O.E. Zamir. *Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results*. PhD thesis, University of Washington, 1999.

[41] H.J. Zeng, Q.C. He, Z. Chen, W.Y. Ma, and J. Ma. Learning to cluster web search results. In *Proceedings of the 27th annual international conference on Research and Development in Information Retrieval (SIGIR '04)*, pages 210–217, Sheffield, UK, July 2004.

[42] D. Zhang and Y. Dong. Semantic, Hierarchical, Online Clustering of Web Search Results. In *6th Asia-Pacific Web Conference on Advanced Web Technologies and Applications (APWeb '04)*, pages 69–78, Hangzhou, China, April 2004.

Index

Bisecting K-means, 2
Buckshot, 18

component diagram, 73

data mining, 25
distance functions, 7
dmoz, 22

FleXplorer, 92
Fractionation, 18
Frequent Itemset Hierarchical Clustering (FIHC), 25

human labeled training data, 32

incremental, 27

K-means, 2

Latent Semantic Indexing, 5
linear time, 27

metric spaces, 7
mitos, 71

NM-STC, 58

PageRank, 29

salient phrases, 32
scatter/gather, 18
SCuba, 26
sequence diagram, 74
SHOC, 41
similarity functions, 7
Subspace clustering, 26
Suffix Tree Clustering (STC), 18, 27

TermRank, 29

Appendix A

Here are presented the queries used for the user evaluation.

1. UML
2. Ανάκτηση Πληροφοριών
3. Crete
4. Ηράκλειο
5. Διαχείριση Οντολογιών
6. Οπτικοποίηση Γράφων
7. Ο Μίτος της Αριάδνης
8. Βιβλιοθήκη Ρεθύμνου
9. How to add jar files in Eclipse
10. How to install mitos
11. Διακοπές στη Νότια Κρήτη
12. SWKM
13. Φαρμακεία Ηρακλείου
14. ιπτάμενοι δίσκοι στη Νότια Κρήτη
15. Βικελαία
16. Τηλεοπτικό Πρόγραμμα