ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές Σύστασης Όρων για Αναζήτηση σε Επιστημονικές Βάσεις Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΕΤΣΙΟΥ ΘΕΟΦΙΛΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές Σύστασης Όρων για Αναζήτηση σε Επιστημονικές Βάσεις Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΕΤΣΙΟΥ ΘΕΟΦΙΛΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Ιουλίου 2011.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Αν. Καθηγητής Ε.Μ.Π.	Θοδωρής Δαλαμάγκας Ερευνητής Β ' ΙΠΣΥΠ/Ε.Κ "Αθηνά"

Αθήνα, Ιούλιος 2011

(Υπογραφή)

.....

ΠΕΤΣΙΟΣ ΘΕΟΦΙΛΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Τιμολέοντα Σελλή για την ευκαιρία που μου έδωσε να εκπονήσω την παρούσα διπλωματική στο Ινστιτούτο Πληροφοριακών Συστημάτων και Προσομοίωσης. Επίσης ευχαριστώ ιδιαιτέρως τους Θανάση Βεργούλη και Θοδωρή Δαλαμάγκα, για την πολύτιμη βοήθειά τους και την εξαιρετική συνεργασία που είχαμε κατά τη διεξαγωγή της παρούσης εργασίας.

Περίληψη

Βασικός στόχος της συγκεκριμένης διπλωματικής είναι να κατασκευαστεί ένα καλύτερο σύστημα παροχής προτάσεων για τις εφαρμογές Ιστού DIANA. Αναπτύχθηκαν κατάλληλα εργαλεία για τη διαχείριση και την κατασκευή μηχανών αναζήτησης, με κύρια έμφαση σε τεχνικές συντακτικής απόστασης και ευρετηρίων gram. Τα εργαλεία αυτά περιλαμβάνουν κατά κύριο λόγο προγράμματα γραμμένα σε Perl για την κατασκευή ευρετηρίων και μία σειρά από mysql udfs που επιτελούν εργασίες συναφείς με n-grams. Παράλληλα, έγινε χρήση της php και τεχνολογιών ajax για την τροποποίηση του γραφικού περιβάλλοντος του συστήματος DIANA, μέσω του yii framework.
Συνολικά πετύχαμε σημαντική βελτίωση των χρόνων απόκρισης της μηχανής αναζήτησης του συστήματος DIANA.Βελτιώθηκε η ποιότητα των παρεχόμενων αποτελεσμάτων της εφαρμογής σε επίπεδο εύρους προτάσεων καθώς και το περιβάλλον αναζήτησης της εφαρμογής. Δημιουργήθηκαν εργαλεία κατασκευής ευρετηρίων και διαχείρισης της βάσης δεδομένων για τους διαχειριστές τόσο της εφαρμογής DIANA όσο και οποιουδήποτε άλλου συστήματος. Ο διαχειριστής του συστήματος έχει τη δυνατότητα να επιλέξει την κατασκευή index με οποιοδήποτε αριθμό grams και να καθορίσει το βάρος που θα χρησιμοποιηθεί για τα grams.Τέλος, επεκτάθηκε το πακέτο λογισμικού flamingo ώστε να είναι συμβατό με το λειτουργικό σύστημα Mac OS X.

Λέξεις Κλειδιά: << παροχή προτάσεων, αναζήτηση, συντακτική απόσταση, ευρετήρια, ταίριασμα χαρακτήρων>>

Abstract

The main purpose of this thesis was the development of a better term suggestion mechanism for DIANA web applications. We developed several tools to manage search engines, focusing on edit distance and n-gram techniques. These tools mainly consist of programs written in Perl, in order to construct and maintain inverted indexes for ngram-based search engines and mysql udfs which implement operations concerning n-grams. We modified the graphic interface of the web application with the use of php and ajax, implemented in yii framework.
Overall, we achieved a major improvement in time response of the average query on the web application. The options offered by the search engine where improved in terms of variety and the ease of use of the web application improved as well. We also created a series of administration tools for DIANA administrators. These tools consist of programs to manage databases which include inverted indexes for search operations, and are applicable to any operating system. The system administrator has the ability to choose the construction of indexes of variable gram length and assign an arbitrary weight to the grams used. Finally, we made changes to flamingo software installer in order for it to be applicable to Mac OS X.

Keywords: <<term suggestion, edit distance, n-grams, inverted index, mysql udf, string matching, string searching >>

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. Εισαγωγή.....1
1.1 Εισαγωγικά Σχόλια 1
1.2 Αντικείμενο της Εργασίας.....3
1.2.1 Το σύστημα DIANA micro-T.....3
1.2.2 Στόχοι.....3
1.2.2.1 Βελτίωση του χρόνου απόκρισης του συστήματος.....4
1.2.2.2 Βελτίωση της διεπαφής.....4
1.2.3 Σχεδιασμός της Λύσης.....4
1.2.4 Συνεισφορά της Εργασίας.....5
1.3 Οργάνωση του κειμένου 6

2. Θεωρητικό Υπόβαθρο.....7
2.1 Ταίριασμα Συμβολοσειρών.....7
2.1.1 Ακριβές ταίριασμα συμβολοσειρών.....8
2.1.1.1 Αλγόριθμος Bitap(1).....8
2.1.2 Προσεγγιστικό Ταίριασμα Συμβολοσειρών.....11
2.1.2.1 On-line Αλγόριθμοι.....11
2.1.2.1.1 Αλγόριθμος Needleman-Wunsch.....11
2.1.2.1.2 Αλγόριθμος Bitap (2).....14
2.1.2.1.3 Απόσταση Hamming.....15
2.1.2.2 Off-line Αλγόριθμοι.....15
2.1.2.2.1 N-grams & Δενδρα Προθεμάτων.....15
2.1.2.2.2 Grams Μεταβλητού Μεγέθους.....17
2.1.2.2.3 Δένδρα Επιθέματος.....21
2.2 Ταξινόμηση και Ανεύρεση πληροφοριών.....22
2.2.1 Precision 23
2.2.2 Recall.....23
2.2.3 F-measure.....24
2.2.4 Ταξινόμηση & Ομοιότητα.....25

3. Υλοποίηση.....27
3.1 Περιγραφή της λειτουργίας της εφαρμογής.....27
3.2 Το πακέτο flamingo.....31
3.3 Mysql udfs.....34
3.3.1 Προσθήκη μιας udf μέσω της διεπαφής της mysql.....35
3.3.2 Προσθήκη μιας udf στον mysqld server..... 36
3.3.3 Δημιουργία αποθηκευμένων διαδικασιών και συναρτήσεων.....37
3.3.3.1 Συναρτήσεις.....37
3.3.3.2 Διαδικασίες.....38
3.3.4 Ngram udfs.....39
3.3.4.1 C/C++ udfs.....39
3.3.4.2 Αποθηκευμένες Διαδικασίες/Συναρτήσεις.....40
3.4 Υλοποίηση του μηχανισμού αναζήτησης με ngrams.....43
3.5 Παρουσίαση των αποτελεσμάτων σε πραγματικό χρόνο.....45

3.5.1 Asynchronous JavaScript and XML (AJAX) 45
3.5.2 jQuery....46
3.5.3 Τροποποίηση της εφαρμογής με χρήση των μεθόδων του yii framework....47

4. Αξιολόγηση....49
4.1 Περιγραφή των συνόλων δεδομένων (datasets)....49
4.2 Μέτρηση της αποδοτικότητας των μεθόδων που χρησιμοποιούν grams....53
4.2.1 Πραγματικό σύνολο δεδομένων....53
4.2.2 Συνθετικό σύνολο δεδομένων....55
4.3 Μέτρηση της ακρίβειας των μεθόδων που χρησιμοποιούν grams....58
4.3.1 Πραγματικό σύνολο δεδομένων....58
4.3.2 Συνθετικό σύνολο δεδομένων....59
4.4 Μέτρηση χρονικής απόκρισης για μεταβλητό μέγεθος αποτελεσμάτων....63
4.5 Πλεονεκτήματα και μειονεκτήματα της κάθε μεθόδου....63
4.6 Επιλογή των παραμέτρων αναζήτησης για το σύστημα DIANA....66

5. Επίλογος....67
5.1 Σύνοψη....67
5.2 Μελλοντικές Εργασίες....68

6. Βιβλιογραφία....69

1

Εισαγωγή

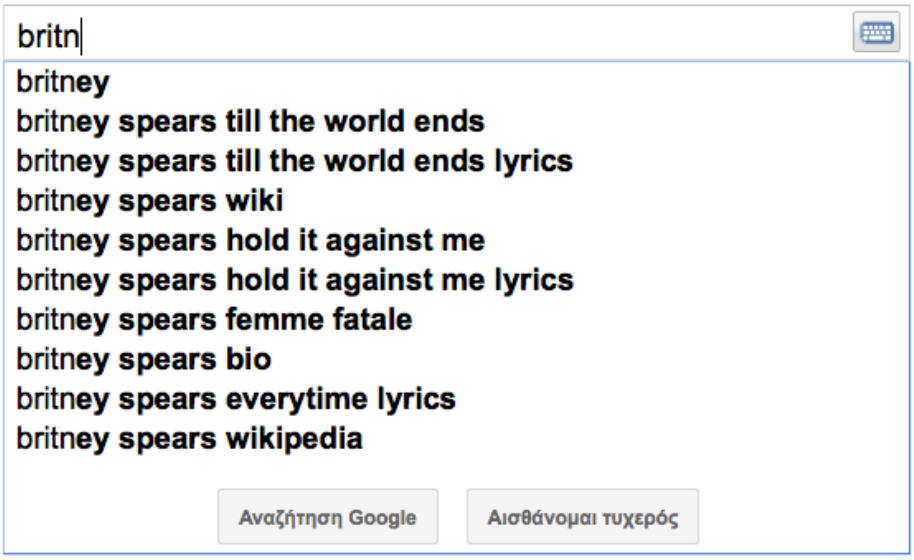
1.1 Εισαγωγικά Σχόλια

Τις τελευταίες δεκαετίες, η εξέλιξη του παγκοσμίου ιστού ήταν τέτοια που κατέστησε αναγκαία την εύρεση αποδοτικών μηχανισμών για τη διαχείριση του τεραστίου όγκου της πληροφορίας που διακινούνταν στο Διαδίκτυο αλλά και εσωτερικά στους μεγάλους οργανισμούς. Αυξήθηκε δραματικά η ανάγκη για περισσότερο εξειδικευμένες αναζητήσεις εικόνων, video, δημοσιεύσεων κ.ά. καθώς και για αποτελεσματική ταξινόμηση και επεξεργασία των δεδομένων, ενώ ταυτόχρονα, έγινε μία προσπάθεια να βελτιωθεί η ποιότητα των παρεχόμενων αποτελεσμάτων και να προσαρμοστεί η αναζήτηση στις ατομικές ανάγκες των χρηστών.

Ο πιο γνωστός και διαδεδομένος τρόπος αναζήτησης υλικού είναι η αναζήτηση που πραγματοποιείται με βάση λέξεις-κλειδιά (*keyword search*). Σε αυτό τον τύπο αναζήτησης, ο χρήστης καθορίζει ένα πλήθος λέξεων-κλειδιών, ο οποίος χαρακτηρίζει το υλικό που τον ενδιαφέρει. Η μηχανή αναζητήσεων βασίζεται σε αυτές τις λέξεις-κλειδιά προκειμένου να επιστρέψει τα αντίστοιχα αποτελέσματα. Ένα γνωστό πρόβλημα σε αυτού του τύπου τις αναζητήσεις είναι ότι ο χρήστης θα πρέπει να γνωρίζει επακριβώς ποιές λέξεις κλειδιά να χρησιμοποιήσει. Συνήθως κάτι τέτοιο δεν είναι δυνατό διότι υπάρχουν διάφορες παραλλαγές που μπορεί κανείς να χρησιμοποιήσει και η μηχανή αναζήτησης δεν μπορεί να λαμβάνει υπ'όψιν τις όλες αυτές τις εναλλακτικές. Επιπρόσθετα, είναι πολύ συνηθισμένο κατά τη διάρκεια της πληκτρολόγησης οι χρήστες να εισάγουν λάθη στο σύνολο των λέξεων-κλειδιών. Αυτού του είδους τα προβλήματα, μπορούν να περιορίσουν τα αποτελέσματα που θα επέστρεφε η μηχανή αναζήτησης προς τους χρήστες και κάποιο σημαντικό υλικό να αγνοηθεί. Προκειμένου να εξαλειφθούν τέτοιου είδους προβλήματα, ξεκίνησαν να εμφανίζονται στο προσκήνιο μηχανισμοί *παροχής προτάσεων*. Αυτοί οι μηχανισμοί, προσπαθούν να προβλέψουν τη λέξη κλειδί που ενδεχομένως επιθυμούσε να πληκτρολογήσει ο χρήστης (αλλά δεν το έκανε) βάσει συγκεκριμένων κριτηρίων συνάφειας με το

αντικείμενο της αναζήτησης. Η χρησιμότητά τους αναδεικνύεται εντονότερα σε εφαρμογές μεγάλου όγκου πληροφοριών με οριζόντια ή κατακόρυφη κατάτμηση, όπου οι συνδέσεις μεταξύ των κατηγοριών και τύπων δεδομένων είναι μη προφανείς και οι χρήστες είναι περισσότερο πιθανό να πραγματοποιήσουν κάποιο σφάλμα.

Τα συστήματα παροχής προτάσεων, έχουν ενσωματωθεί στην πλειοψηφία των διαδικτυακών εφαρμογών και ιστοσελίδων προκειμένου να διευκολύνονται οι χρήστες κατά τις αναζητήσεις τους. Στο παρακάτω στιγμιότυπο, παρουσιάζεται ένα σύνολο προτάσεων για τον όρο 'britn', που πληκτρολογεί ο χρήστης στην μηχανή αναζήτησης της Google:



Τα κριτήρια ταξινόμησης των αποτελεσμάτων, η φύση των προτάσεων και η ταχύτητα απόκρισης στα ερωτήματα του χρήστη είναι τα βασικά σημεία που καθορίζουν την ποιότητα και την απόδοση των διαφόρων μηχανών αναζήτησης. Το αντικείμενο της συγκεκριμένης διπλωματικής είναι η επέκταση ενός μηχανισμού παροχής προτάσεων για ένα σύνολο υπαρχουσών διαδικτυακών εφαρμογών σχετικών με τις βιοεπιστήμες. Στις επόμενες παραγράφους, αναλύουμε σε μεγαλύτερη λεπτομέρεια το αντικείμενο της εργασίας.

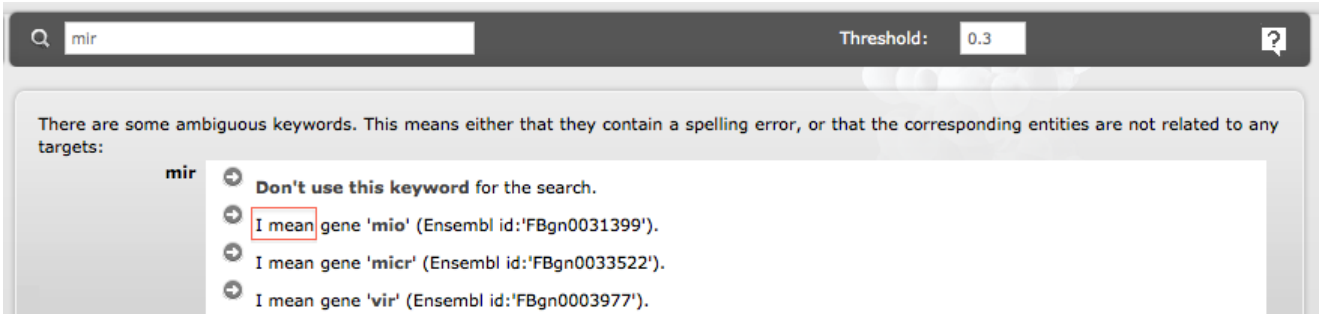
1.2 Αντικείμενο της εργασίας

1.2.1 Το σύστημα DIANA micro-T

Το σύστημα DIANA micro-T είναι μια εφαρμογή Ιστού, που έχει αναπτυχθεί από το ΙΠΣΥΠ για το Ερευνητικό Κέντρο Βιοϊατρικών Επιστημών **"Αλέξανδρος Φλέμινγκ"**, το οποίο διεξάγει έρευνα μεταξύ άλλων και στο πεδίο των microRNAs. Τα microRNAs είναι βιομόρια πολύ μικρού μήκους, τα οποία έχουν "στόχους" (targets) σε γονίδια. Η ύπαρξη στόχου ενός μορίου microRNA σε κάποιο γονίδιο συνεπάγεται στην παρεμπόδιση της παραγωγής της αντίστοιχης πρωτεΐνης. Η πληροφορία αυτή είναι ιδιαίτερα πολύτιμη για την εξαγωγή συμπερασμάτων σχετικά με τα αίτια ασθενειών.
Η εφαρμογή micro-T παρέχει στους βιολόγους ερευνητές τη δυνατότητα να αναζητήσουν microRNA στόχους και σχετικές με αυτούς πληροφορίες. Οι στόχοι έχουν προβλεφθεί με βάση υπολογιστικές τεχνικές της ερευνητικής ομάδας του "Αλέξανδρος Φλέμινγκ" και στη συνέχεια έχουν αποθηκευτεί σε μία σχεσιακή βάση δεδομένων. Η αναζήτηση των microRNA στόχων γίνεται από τους χρήστες μέσω του προσδιορισμού λέξεων-κλειδιών (keywords), οι οποίες χαρακτηρίζουν είτε κάποιο γονίδιο, είτε κάποιο μόριο microRNA, είτε και τα δύο μαζί. Επίσης οι λέξεις-κλειδιά, μπορεί να προσδιορίζουν κάποια σχετική με τους στόχους πληροφορία (π.χ. παράγοντες μεταγραφής, συμμετοχή αντίστοιχου γονιδίου σε βιολογικά μονοπάτια κτλ).

1.2.2 Στόχοι

Στη μηχανή αναζήτησης της εφαρμογής DIANA micro-T[1] υπάρχει μία φόρμα αναζήτησης που συμπληρώνεται από το χρήστη με λέξεις-κλειδιά. Όταν ο χρήστης συμπληρώνει μια λέξη-κλειδί, και η λέξη ταυτίζεται με κάποιο στοιχείο της βάσης δεδομένων της εφαρμογής, τότε το σύστημα την αναγνωρίζει και παρέχει στο χρήστη διάφορες επιλογές. Σε αντίθετη περίπτωση, το σύστημα παρέχει μία λίστα με προτάσεις για το ποιά λέξη-κλειδί ενδεχομένως θα ήθελε χρήστης να πληκτρολογήσει. Για παράδειγμα, στο στιγμιότυπο που ακολουθεί, ο χρήστης πληκτρολογεί προς αναζήτηση τη συμβολοσειρά "mir". Ωστόσο το σύστημα δεν την αναγνωρίζει και παρέχει *εναλλακτικά αποτελέσματα* (τις συμβολοσειρές "mio", "micr" ,"vir" κτλ) στην αναζήτηση. Οι προτάσεις αυτές καθορίζονται από τον ειδικό μηχανισμό αναζήτησης που εφαρμόζεται στην εφαρμογή.



[1] http://www.microrna.gr/microT-CDS

Το συγκεκριμένο σύστημα αναζήτησης, εμφανίζει διάφορα προβλήματα, το κυριότερο των οποίων είναι η μεγάλη καθυστέρηση κατά την παρουσίαση των αποτελεσμάτων. Στα πλαίσια αυτά, δεν παρέχονται αποτελέσματα σε πραγματικό χρόνο, καθώς ο χρήστης πληκτρολογεί και σε ορισμένες περιπτώσεις παρουσιάζονται αστοχίες ως προς την ποιότητα των αποτελεσμάτων. Στόχος μας είναι η βελτίωση αυτού του συστήματος προτάσεων, η οποία μπορεί να προσανατολιστεί στις εξής κατευθύνσεις:

1. Βελτίωση του **χρόνου απόκρισης**
2. Βελτίωση της **διεπαφής (interface)**

Αναλύουμε συνοπτικά το κάθε ένα από αυτά:

1.2.2.1 Βελτίωση του χρόνου απόκρισης του συστήματος

Το υπάρχον σύστημα παροχής προτάσεων χρησιμοποιεί την συνάρτηση levenshtein της PHP προκειμένου να υπολογίσει την συντακτική απόσταση (edit distance) κάποιας λέξης-κλειδιού σε σχέση με τους αποθηκευμένους όρους στη βάση. Η μέθοδος αυτή οδηγεί σε μεγάλες χρονικές καθυστερήσεις κατά την παρουσίαση των αποτελεσμάτων στο χρήστη, κυρίως λόγω των παρακάτω παραγόντων:

- Όλες οι εγγραφές στη βάση πρέπει να διατρέχονται σε κάθε ερώτημα.
- Όλες οι εγγραφές της βάσης πρέπει να μεταφερθούν από τη MySQL στην PHP προκειμένου να υπολογιστεί η συντακτική απόσταση.

Σκοπός μας είναι με την παρούσα εργασία να μελετηθούν εναλλακτικοί τρόποι υπολογισμού ομοιότητας μεταξύ συμβολοσειρών, οι οποίοι θα δίδουν καλύτερα χρονικά αποτελέσματα.

1.2.2.2 Βελτίωση της διεπαφής

Στα πλαίσια της παροχής καλύτερων αποτελεσμάτων προς το χρήστη, εφαρμόζουμε ορισμένες τροποποιήσεις στο μηχανισμό παροχής προτάσεων, προκειμένου να μην έχουμε μία απλή φόρμα αναζήτησης αλλά να παρέχονται προτάσεις στο χρήστη κατά την πληκτρολόγηση (προτάσεις που θα προσαρμόζονται δηλαδή στη συμβολοσειρά που ο χρήστης έχει εισάγει σε μία δεδομένη χρονική στιγμή). Τη δυνατότητα παροχής προτάσεων σε πραγματικό χρόνο μας εξασφαλίζουν οι βελτιώσεις αποδοτικότητας που εφαρμόζονται στην μηχανή αναζήτησης και οδηγούν σε σημαντική μείωση του μέσου χρόνου απάντησης στα ερωτήματα των χρηστών.

1.2.3 Σχεδιασμός της Λύσης

Προκειμένου να υλοποιήσουμε τα παραπάνω, αρχικά επιδιώκουμε να επιλύσουμε τα προβλήματα που εισάγονται στο σύστημα λόγω των χρονικών καθυστερήσεων που εισάγει η επικοινωνία μεταξύ MySQL και PHP. Προς τούτο, χρησιμοποιούμε, σε πρώτο στάδιο, το πακέτο λογισμικού flamingo. Πρόκειται για ένα σύνολο προγραμμάτων γραμμένων σε C++ αλλά και udf συναρτήσεων, που επιτελούν λειτουργίες προσεγγιστικού ταιριάσματος αλφαριθμητικών (approximate string matching). Ενσωματώνοντας τις παρεχόμενες από το flamingo udf συναρτήσεις στη βάση δεδομένων, τροποποιούμε τη μηχανή

αναζήτησης, προκειμένου οι αναγκαίοι υπολογισμοί να πραγματοποιούνται κατά το δυνατόν από τη βάση, και όχι από την εφαρμογή.
Η προηγούμενη λύση βελτιώνει σημαντικά το χρόνο της μέσης αναζήτησης (περίπου 10 φορές σε σχέση με την αρχική έκδοση της εφαρμογής), ωστόσο, όπως αναφέρθηκε και στην προηγούμενη ενότητα, απαιτεί να εξετάζουμε κάθε φορά ολόκληρη τη βάση δεδομένων. Συνεπώς δεν είναι αποδοτική όσο μεγαλώνει ο αριθμός των εγγραφών και ως εκ τούτου, εξετάζουμε διαφορετικούς αλγορίθμους για την υλοποίηση string matching, προκειμένου να επιλέξουμε την καταλληλότερη λύση για την εφαρμογή μας.
Το επίκεντρο της μελέτης μας εστιάζεται σε τεχνικές που χρησιμοποιούν n-grams. Με τον όρο n-gram, εννοούμε μία υπακολουθία μήκους n, μίας δεδομένης συμβολοσειράς s. Στα πλαίσια της παρούσης εργασίας, επιλέξαμε να υλοποιήσουμε την τελική λύση μας, κατασκευάζοντας μία αντεστραμμένη λίστα με grams σταθερού μήκους. Υλοποιήσαμε συναρτήσεις udf προκειμένου να γίνεται η επεξεργασία των ερωτημάτων σχεδόν εξολοκλήρου από τη βάση δεδομένων. Τα κυριότερα σημεία της μεθόδου που ακολουθήσαμε είναι τα εξής:

- Για κάθε δεδομένο της βάσης παράγουμε όλα τα πιθανά grams σταθερού μεγέθους n.
- Τα grams αυτά τα εισάγουμε σε ένα πίνακα, ο οποίος θα αποτελεί και τον inverted index μας. Ο πίνακας αυτός έχει τη μορφή [<gram>,<term>,..] και ως εκ τούτου, για κάθε ζευγάρι gram-term (το gram και ο όρος από τον οποίο προέρχεται), υπάρχει και η αντίστοιχη εγγραφή στη βάση.
- Όταν ο χρήστης εισάγει μία συμβολοσειρά προς αναζήτηση στην εφαρμογή, παράγουμε όλα τα πιθανά grams της συμβολοσειράς. Ακολούθως προτείνουμε στο χρήστη τους όρους εκείνους με τους οποίους η συμβολοσειρά έχει τα περισσότερα κοινά grams, βάσει του inverted index.

Προκειμένου να βελτιώσουμε την ποιότητα των παρεχόμενων αποτελεσμάτων, εξετάζουμε εναλλακτικές τεχνικές από τον κλάδο της ανεύρεσης πληροφοριών (Information Retrieval). Αξιολογούμε διαφορετικές μεθόδους υπολογισμού tfxidf βαρών για τα grams του inverted index και πραγματοποιούμε μετρήσεις για τα μεγέθη recall και precision που πετυχαίνουν οι μέθοδοι αυτοί. Τέλος, για την παροχή προτάσεων σε πραγματικό χρόνο, χρησιμοποιήθηκαν οι διαφορετικές δυνατότητες των τεχνολογιών ajax και JQuery, προκειμένου αυτές να ενσωματωθούν στην εφαρμογή.

1.2.4 Συνεισφορά Της Εργασίας

Τα εργαλεία που αναπτύχθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας, ήταν κατά κύριο λόγο προσανατολισμένα στην εφαρμογή ιστού DIANA, ωστόσο είναι δυνατό να χρησιμοποιηθούν και σε οποιοδήποτε άλλο σύστημα. Η κυριότερη συνεισφορά της παρούσης εργασίας συνίσταται στα εξής βασικά σημεία:

- Σημαντική βελτίωση των χρόνων απόκρισης της μηχανής αναζήτησης του συστήματος DIANA
- Βελτίωση της ποιότητας των αποτελεσμάτων της εφαρμογής.

- Βελτίωση του περιβάλλοντος αναζήτησης για τους χρήστες οι οποίοι πλέον λαμβάνουν αποτελέσματα ενώ πληκτρολογούν.
- Δημιουργία εργαλείων κατασκευής inverted index και διαχείρισης της βάσης δεδομένων για τους διαχειριστές τόσο της εφαρμογής DIANA όσο και οποιουδήποτε άλλου συστήματος. Ο διαχειριστής του συστήματος έχει τη δυνατότητα να επιλέξει την κατασκευή index με οποιοδήποτε αριθμό grams και να την επιλογή του tfxidf βάρους που θα χρησιμοποιηθεί για τα grams.
- Δημιουργία ποικιλίας mysql udf για approximate string matching με τεχνικές n-grams.
- Προσαρμογή του πακέτου λογισμικού flamingo ώστε να είναι συμβατό με το λειτουργικό σύστημα Mac OS X.

1.3 Οργάνωση του κειμένου

Η συγγραφή της παρούσας διπλωματικής εργασίας έχει οργανωθεί με τέτοιο τρόπο ώστε να παρουσιάζονται αναλυτικά οι επιμέρους φάσεις της ανάλυσης, της υλοποίησης και της δοκιμής του συστήματος. Συνοπτικά:

- Στο $2^{ο}$ κεφάλαιο γίνεται σύντομη περιγραφή των δημοφιλέστερων αλγορίθμων και τεχνικών που έχουν αναπτυχθεί γύρω από τα πεδία της αναζήτησης και του ταιριάσματος χαρακτήρων.
- Στο $3^{ο}$ κεφάλαιο παρουσιάζονται τα επιμέρους υποσυστήματα που απαρτίζουν την εφαρμογή, και παρατίθενται τα βήματα των διαφορετικών υλοποιήσεων μηχανισμών αναζήτησης που εφαρμόσαμε επί της εφαρμογής.
- Στο $4^{ο}$ κεφάλαιο αξιολογούνται τα χαρακτηριστικά των υλοποιήσεων που αναπτύχθηκαν στην ενότητα 3 μέσα από τις ανάλογες πειραματικές διαδικασίες.
- Στο $5^{ο}$ κεφάλαιο παρουσιάζονται οι αποφάσεις που ελήφθησαν για την τελική επιλογή των παραμέτρων της εφαρμογής, αναλύονται τα πλεονεκτήματα και τα μειονεκτήματα των διαφόρων τεχνικών, και παρατίθενται επιπλέον πληροφορίες συναφείς με τις εργασίες που επιτελέστηκαν.

2

Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό, παρουσιάζεται το αναγκαίο θεωρητικό υπόβαθρο για την κατανόηση των μεθόδων που υλοποιούν το βελτιωμένο σύστημα παροχής προτάσεων για την εφαρμογή DIANA micro-T. Αναφερόμαστε στο ταίριασμα συμβολοσειρών και στους κυριότερους αλγορίθμους που το υλοποιούν. Αρχίζοντας με τη συντακτική απόσταση, αναλύουμε διαφόρους αλγορίθμους που υλοποιούν ακριβές και μερικό ταίριασμα, όπως ο αλγόριθμος *Needleman*-Wunsch και οι παραλλαγές τους αλγορίθμου bitap. Ακολούθως, αναλύουμε τις κυριότερες τεχνικές ευρετηρίου, όπως να n-grams και τα v-grams.

2.1 Ταίριασμα Συμβολοσειρών

Μία βασική τεχνική που χρησιμοποιείται σε μηχανές αναζήτησης εφαρμογών ιστού είναι αυτή του ταιριάσματος συμβολοσειρών (string matching). Η μέθοδος είναι γνωστή εδώ και αρκετές δεκαετίες και βρίσκει διάφορες εφαρμογές, όπως η αναγνώριση φωνής, η αναπαράσταση ομιλίας, η αναγνώριση μοριακών τύπων και άλλων ακολουθιών, η κρυπτογραφία κ.ά.
Το ταίριασμα συμβολοσειρών μπορεί να είναι απόλυτο (exact string matching) ή προσεγγιστικό (approximate string matching).Στο απόλυτο ταίριασμα αναζητούμε και επιστρέφουμε ως αποτέλεσμα συμβολοσειρές οι οποίες ταυτίζονται ακριβώς με τη συμβολοσειρά που πληκτρολόγησε ο χρήστης. Στο προσεγγιστικό ταίριασμα χαρακτήρων αντίθετα, αναζητούμε και επιστρέφουμε συμβολοσειρές οι οποίες παρουσιάζουν ένα βαθμό «ομοιότητας» με τη συμβολοσειρά που εισήγαγε ο χρήστης. Η ομοιότητα αυτή δεν είναι αναγκαία ταύτιση, όπως συμβαίνει με το απόλυτο ταίριασμα, χωρίς ωστόσο κάτι τέτοιο να αποκλείεται.

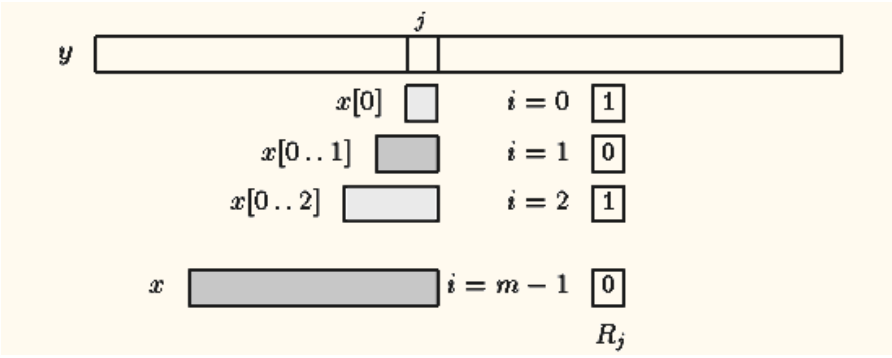
2.1.1 Ακριβές ταίριασμα συμβολοσειρών

Ουσιαστικά πρόκειται για την αναζήτηση μιας συμβολοσειράς (string searching) και όχι τόσο για ταίριασμα, όπως εξηγήθηκε και προηγουμένως. Έχουν αναπτυχθεί διάφοροι αλγόριθμοι για την αναζήτηση συμβολοσειρών, ορισμένοι από τους οποίους αναφέρονται παρακάτω μαζί με τα χαρακτηριστικά τους[2]:

Algorithm	Preprocessing time	Matching time[1]
Naïve string search algorithm	0 (no preprocessing)	Θ((n-m+1) m)
Rabin–Karp string search algorithm	Θ(m)	average Θ(n+m), worst Θ((n-m+1) m)
Finite-state automaton based search	Θ(m |Σ|)	Θ(n)
Knuth–Morris–Pratt algorithm	Θ(m)	Θ(n)
Boyer–Moore string search algorithm	Θ(m + |Σ|)	Ω(n/m), O(n)
Bitap algorithm (*shift-or, shift-and, Baeza–Yates–Gonnet*)	Θ(m + |Σ|)	O(mn)

2.1.1.1 Αλγόριθμος Bitap(1)

Ο **αλγόριθμος bitap**[3], γνωστός επίσης και ως **shift-or** ή **shift-and** ή **Baeza-Yates-Gonnet** μπορεί να χρησιμοποιηθεί για exact string matching κάνοντας πρακτικά approximate string matching. Συγκεκριμένα, αν δύο συμβολοσειρές έχουν λιγότερα από k διαφορετικά στοιχεία, ο αλγόριθμος τις θεωρεί ίδιες. Για τον υπολογισμό, υπολογίζει bitmasks που περιέχουν ένα bit για κάθε στοιχείο του string και ακολούθως χρησιμοποιεί πράξεις σε επίπεδο bit (bitwise operations) που επιταχύνουν ιδιαίτερα τη διαδικασία.
Περισσότερο αναλυτικά, έστω ότι R είναι ένας bit array μεγέθους m (το pattern το οποίο αναζητούμε μέσα σε μία μεγαλύτερη συμβολοσειρά text). Το διάνυσμα R_j είναι η τιμή του πίνακα R αφού έχουμε επεξεργαστεί τον χαρακτήρα y[j] της text,σύμφωνα με την παρακάτω εικόνα:



[2] http://en.wikipedia.org/wiki/String_matching
[3] http://en.wikipedia.org/wiki/Bitap_algorithm

Ουσιαστικά, περιέχει πληροφορίες για όλα τα προθέματα του x που τελειώνουν στη θέση j μέσα στο text (που στην παραπάνω εικόνα αναπαριστούμε με j), για 0<i<=m-1. Αν οι αντίστοιχες συμβολοσειρές ταυτίζονται τότε χρησιμοποιούμε την τιμή 0 αλλιώς χρησιμοποιούμε την τιμή 1, δηλαδή ισχύει ότι

$$R_j[i]=\begin{cases} 0 \; if \; x[0,1] = y[j-i,j] \\ 1 \; otherwise \end{cases}$$

Προκειμένου να υπολογίσουμε το διάνυσμα R_{j+1} από το R_j, στις θέσεις όπου έχουμε ταίριασμα χαρακτήρα (δηλαδή ισχύει $R_j[i]=0$) έχουμε:

$$R_{j+1}[i+1]=\begin{cases} 0 \; if \; x[i+1] = y[i+1], \\ 1 \; otherwise \end{cases}$$

και

$$R_{j+1}[0]=\begin{cases} 0 \; if \; x[0] = y[j+1], \\ 1 \; otherwise \end{cases}$$

Αν $R_{j+1}[m+1]=0$ τότε έχουμε απόλυτο ταίριασμα. Η μετάβαση από το R_j στο R_{j+1} μπορεί να υπολογιστεί αποδοτικά ως εξής:

- Για κάθε χαρακτήρα c που ανήκει σε ένα αλφάβητο Σ, έστω S_c ένας bit array μεγέθους m τέτοιος ώστε για κάθε ∀i∈[0,m-1) να ισχύει $S_c[i]=0$ ανν x[i]=c.
- Ο πίνακας S_c υποδεικνύει τις θέσεις του χαρακτήρα c μέσα στο pattern x. Κάθε S_c μπορεί να υποστεί επεξεργασία πριν την αναζήτηση. Ο υπολογισμός του R_{j+1} ανάγεται σε δύο βασικές λειτουργίες, την shift και την or: $R_{j+1}=SHIFT(R_j)$ OR $S_{y[j+1]}$.

Παραθέτουμε τον κώδικα για την υλοποίηση του αλγορίθμου σε C:

Αλγόριθμος bitap για exact string matching

```c
#include <stdlib.h>
#include <string.h>

typedef char BIT; /* needs only to hold the values 0 and 1 */

const char *bitap_search(const char *text, const char *pattern)
{
    const char *result = NULL;
    int m = strlen(pattern);
    BIT *R;
    int i, k;


    if (pattern[0] == '\0') return text;

    /* Initialize the bit array R */
    R = malloc((m+1) * sizeof *R);
    R[0] = 1;
    for (k=1; k <= m; ++k)
      R[k] = 0;
```

```
    for (i=0; text[i] != '\0'; ++i) {
        /* Update the bit array. */
        for (k=m; k >= 1; --k)
          R[k] = R[k-1] && (text[i] == pattern[k-1]);

        if (R[m]) {
            result = (text+i - m) + 1;
            break;
        }
    }

    free(R);
    return result;
}
```

Ένα παράδειγμα της χρήσης του αλγορίθμου είναι το εξής:

Έστω ότι αναζητούμε τη συμβολοσειρά ‘gca’ μέσα στην ‘tsgcag’. Τα βήματα είναι τα εξής:

Αρχή:

1)
tsgcag
|
transition: 111.t -> 111

2)
tsgcag
|
transition: 111.s -> 111

3)
tsgcag
|
transition: 111.g -> 011

4)
tsGcag (εντοπίστηκε το g)
|
transition: 011.c -> 101

5)
tsGCag (εντοπίστηκε το c)
|
transition: 101.a -> 110

6)
tsGCAg
|
transition: 110.g -> 011

---

tsGCAg (εντοπισμός της ακολουθίας)

Επειδή ο αλγόριθμος κάνει αναγκαστικά χρήση δομών δεδομένων, αποδίδει καλύτερα σε συμβολοσειρές μικρότερες από ένα καθορισμένο όριο (το μήκος του pattern πρέπει να μην είναι μεγαλύτερο από το μέγεθος της μνήμης για κάθε λέξη της μηχανής στην οποία εκτελείται ο αλγόριθμος). Μόλις εφαρμοστεί μία φορά για μία δοσμένη αλφάβητο και ένα μήκος λέξης m, η **χρονική πολυπλοκότητα** είναι O(mn), ανεξαρτήτως της δομής δεδομένων που χρησιμοποιήθηκε. Τέλος, αξίζει να σημειωθεί ότι ο αλγόριθμος αυτός είναι η καρδιά της λειτουργίας agrep του Unix.

2.1.2 Προσεγγιστικό Ταίριασμα Συμβολοσειρών

Το προσεγγιστικό ταίριασμα συμβολοσειρών[4] (approximate string matching), που αναφέρεται συχνά και ως fuzzy string searching, ουσιαστικά εισάγει την ανάγκη ύπαρξης μίας μετρικής για την «ομοιότητα» μεταξύ δύο ή περισσότερων συμβολοσειρών. Οι αλγόριθμοι που αναπτύχθηκαν για τον υπολογισμό αυτό χωρίζονται γενικά σε δύο ευρείες κατηγορίες: τους αλγορίθμους που τρέχουν εκ του μηδενός (on-line) και αυτών που απαιτούν μία προεργασία επί των δεδομένων στα οποία θα κάνουμε την αναζήτηση (off-line). Οι on-line αλγόριθμοι συνηθέστερα χρησιμοποιούν δυναμικό προγραμματισμό αξιοποιώντας μία μετρική, ενώ οι offline χρησιμοποιούν μία σύνθετη δομή αποθήκευσης δεδομένων προκειμένου να καταστήσουν γρηγορότερη την αναζήτηση στα δεδομένα. Ακολούθως παραθέτουμε ορισμένους χαρακτηριστικούς αλγορίθμους για κάθε κατηγορία.

2.1.2.1 On-line Αλγόριθμοι

Αυτή η ομάδα αλγορίθμων βασίζεται συνήθως σε παραλλαγές χρήσης της συντακτικής απόστασης (**edit distance)** μεταξύ δύο συμβολοσειρών, την οποία ορίζουμε ως τον ελάχιστο αριθμό εισαγωγών, διαγραφών και αντικαταστάσεων (οι τρείς αυτές λειτουργίες καλούνται πρωταρχικές –primitive[5]-), που απαιτούνται, προκειμένου οι δύο συμβολοσειρές να ταυτίζονται.

- Με τον όρο **εισαγωγή (insertion)**, εννοούμε την εισαγωγή **ενός** χαρακτήρα στην υπάρχουσα συμβολοσειρά, π.χ. hi -> hi**t**.
- Με τον όρο **διαγραφή (deletion)**, εννοούμε την διαγραφή **ενός** χαρακτήρα από την υπάρχουσα συμβολοσειρά, π.χ. hi->h.
- Με τον όρο **αντικατάσταση (substitution)**, εννοούμε την αντικατάσταση **ενός** χαρακτήρα από την υπάρχουσα συμβολοσειρά, π.χ. pitcher->picher.

Ανάλογα με τις πράξεις που χρησιμοποιούνται κάθε φορά ως βασικές, έχουμε διαφοροποιήσεις στον παραπάνω ορισμό και στον τρόπο υπολογισμού. Στο σημείο αυτό, οφείλουμε να σημειώσουμε ότι πολλοί ερευνητές αναφέρονται στην συντακτική απόσταση με τον όρο **Levenshtein distance**, εννοώντας την edit distance.Ορισμένοι χαρακτηριστικοί αλγόριθμοι που κάνουν χρήση της edit distance αναφέρονται ακολούθως:

2.1.2.1.1 Αλγόριθμος Needleman-Wunsch

Ο αλγόριθμος αυτός είναι και ο πιο δημοφιλής, με αποτέλεσμα να αναφερόμαστε σε αυτή την τεχνική προκειμένου να υπολογίσουμε την απόσταση Levenshtein (edit distance). Πρόκειται για υπολογισμό βάσει του ορισμού που δόθηκε στην προηγούμενη ενότητα (2.1.2.1). Προκειμένου να κατανοήσουμε τη λειτουργία του αλγορίθμου σχηματίζουμε ένα πίνακα που έχει στην πρώτη γραμμή και στήλη τις λέξεις των οποίων θέλουμε να υπολογίσουμε την edit distance, και γεμίζουμε τον πίνακα με τον αριθμό των

[4] http://en.wikipedia.org/wiki/Approximate_string_matching

[5] Πολλοί ερευνητές ορίζουν ως primitive operation και την μετάβαση (transposition) όπου δύο γράμματα μίας λέξης εναλλάσσουν τη θέση τους.

πρωταρχικών λειτουργιών (primitive operations) που απαιτούνται προκειμένου να καταλήξουμε από τη μία συμβολοσειρά στην άλλη. Για παράδειγμα, στον παρακάτω πίνακα, τα strings “S” και ”Sa” έχουν απόσταση 1, καθώς απαιτείται μία μόλις πρωταρχική λειτουργία για να μεταβούμε από το ένα στο άλλο (στοιχείο [1,2] του πίνακα), ενώ τα “S” και “S” απόσταση 0 (στοιχείο [1,1]).

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Ακολούθως παραθέτουμε τον αλγόριθμο αναλυτικότερα (straightforward implementation). Η λύση βασίζεται σε δυναμικό προγραμματισμό[6]. Θεωρούμε ότι για κάθε i,j του πίνακα d που περιέχει τις αποστάσεις Levenshtein, το d[i,j] θα περιέχει την Levenshtein Distance ανάμεσα στους πρώτους i χαρακτήρες του string s και τους πρώτους j χαρακτήρες του string t. Η απόσταση Levenshtein των δύο συμβολοσειρών θα δίνεται από το στοιχείο $d[n_1, n_2]$, όπου n_1 το μήκος της συμβολοσειράς 1 και n_2 το μήκος της συμβολοσειράς 2, δηλαδή από το τελευταίο στοιχείο της διαγωνίου του πίνακα. Στο παραπάνω παράδειγμα, η απόσταση Levenshtein των συμβολοσειρών Saturday και Sunday θα είναι 3. Πράγματι, η λέξη Saturday προκύπτει έπειτα από τις παρακάτω πράξεις:

- n->r (αντικατάσταση): Su**r**day
- Εισαγωγή του a: S**a**urday
- Εισαγωγή του t: Sa**t**urday

Αντίστροφα προκύπτει και η λέξη Sunday από την Saturday. Για να υπολογίσουμε το τελευταίο στοιχείο της κύριας διαγωνίου εργαζόμαστε ως εξής:

- Αρχικά εισάγουμε στην πρώτη γραμμή και στήλη των αριθμό των βημάτων που απαιτούνται προκειμένου να φτάσουμε στο αντίστοιχο σημείο της συμβολοσειράς, που ταυτίζεται με τον αριθμό του στοιχείου στο οποίο βρισκόμαστε. Στο παραπάνω παράδειγμα, το πρώτο στοιχείο του πίνακα περιέχει το 0 καθώς δεν έχουμε εισάγει κάποιο χαρακτήρα, το δεύτερο στοιχείο της πρώτης γραμμής και στήλης τον αριθμό 1 καθώς εισάγαμε ένα χαρακτήρα στην γραμμή και τη στήλη αντίστοιχα κοκ.
- Ακολούθως, διασχίζουμε όλο τον πίνακα ως εξής: Αν στη θέση [i,j] οι χαρακτήρες των αντιστοίχων συμβολοσειρών ταυτίζονται, τότε η edit distance δεν μεταβάλλεται σε σχέση με την προηγούμενη θέση στην οποία βρισκόμαστε και συνεπώς ισχύει d[i, j] := d[i-1, j-1] . Σε αντίθετη περίπτωση προκειμένου να υπολογίσουμε την edit distance d[i, j], εξετάζουμε ποιός τρόπος με

[6] Για bottom-up υλοποίηση, βλ. 1974, *The String-to-string correction problem* των Robert A. Wagner και Michael J. Fischer.

τον οποίο μπορούμε να οδηγηθούμε σε αυτή την κατάσταση έχει το ελάχιστο κόστος. Δηλαδή , επιλέγουμε την ελάχιστη από τις τιμές (d[i-1, j] + 1, d[i-1, j-1] + 1, d[i, j-1] + 1).

Ο ψευδοκώδικας έχει ως εξής:

```
int LevenshteinDistance(char s[1..m], char t[1..n])
int LevenshteinDistance(char s[1..m], char t[1..n])
{
  declare int d[0..m, 0..n]
  for i from 0 to m
    d[i, 0] := i // the distance of any first
string to an empty second str/
  for j from 0 to n
    d[0, j] := j // the distance of any second
string to an empty first str.

 for j from 1 to n{
    for i from 1 to m
    {
      if s[i] = t[j] then
        d[i, j] := d[i-1, j-1]       // no
operation required
      else
        d[i, j] := minimum
                   (
                     d[i-1, j] + 1,  // a deletion
                     d[i, j-1] + 1,  // an
insertion
                     d[i-1, j-1] + 1 // a
substitution
                   )
    }
  }
  return d[m,n]
}
```

Πολυπλοκότητα:

Εάν εφαρμόσουμε ορισμένες βελτιστοποιήσεις, ο αλγόριθμος έχει **χωρική πολυπλοκότητα O(min(n,m))** καθώς μόνο η προηγούμενη και η τρέχουσα σειρά κάθε φορά χρειάζεται να αποθηκευτούν. Αντίστοιχα, αν χρησιμοποιήσουμε οκνηρή αποτίμηση και εξετάσουμε μόνο τις διαγώνιες αντί για τις σειρές, η **χρονική πολυπλοκότητα είναι O(m(1+d))**, όπου d η Levenshtein distance.Αν μας ενδιαφέρει αποκλειστικά η edit distance να είναι μικρότερη από ένα δοσμένο όριο k, η χρονική πολυπλοκότητα μπορεί να γίνει O(kl) όπου l το μήκος του μικρότερου string.

2.1.2.1.2 Αλγόριθμος Bitap(2)

Ο **αλγόριθμος bitap**[7] που αναφέραμε στο 2.1.1.1,είναι δυνατόν να χρησιμοποιηθεί και για fuzzy searching. Παραθέτουμε τις αλλαγές που απαιτούνται στον κώδικα παρακάτω :

Αλγόριθμος bitap για approximate string matching

```
#include <stdlib.h>
#include <string.h>
#include <limits.h>

const char *bitap_fuzzy_bitwise_search(const char *text, const char *pattern, int k)
{
    const char *result = NULL;
    int m = strlen(pattern);
    unsigned long *R;
    unsigned long pattern_mask[CHAR_MAX+1];
    int i, d;

    if (pattern[0] == '\0') return text;
    if (m > 31) return "The pattern is too long!";

    /* Initialize the bit array R */
    R = malloc((k+1) * sizeof *R);
    for (i=0; i <= k; ++i)
        R[i] = ~1;

    /* Initialize the pattern bitmasks */
    for (i=0; i <= CHAR_MAX; ++i)
        pattern_mask[i] = ~0;
    for (i=0; i < m; ++i)
        pattern_mask[pattern[i]] &= ~(1UL << i);

    for (i=0; text[i] != '\0'; ++i) {
        /* Update the bit arrays */
        unsigned long old_Rd1 = R[0];

        R[0] |= pattern_mask[text[i]];
        R[0] <<= 1;

        for (d=1; d <= k; ++d) {
            unsigned long tmp = R[d];
            /* Substitution is all we care about */
            R[d] = (old_Rd1 & (R[d] | pattern_mask[text[i]])) << 1;
            old_Rd1 = tmp;
        }


        if (0 == (R[k] & (1UL << m))) {
```

[7] http://en.wikipedia.org/wiki/Bitap_algorithm

```
            result = (text+i - m) + 1;
            break;
        }
    }

    free(R);
    return result;
}
```

2.1.2.1.3 Απόσταση Hamming

Η Hamming Distance ανάμεσα σε δύο strings ίσου μήκους είναι ο αριθμός των θέσεων στις οποίες οι αντίστοιχοι χαρακτήρες είναι διαφορετικοί. Για δυαδικά strings α, b , η Hamming Distance ισούται με τον αριθμό των μονάδων σε κάθε α XOR b. Ουσιαστικά πρόκειται για το άνω όριο της Levenshtein Distance αν τα strings έχουν ίσο μήκος.

2.1.2.2 Off-line Αλγόριθμοι

Παρόλο που οι on-line τεχνικές που έχουν αναπτυχθεί είναι ιδιαίτερα αποτελεσματικές, όσο μεγαλώνει ο όγκος των δεδομένων, η απόδοσή τους μειώνεται σημαντικά. Προκειμένου να πετύχουμε καλύτερα αποτελέσματα, είναι αναγκαίο να προ-επεξεργαστούμε το κείμενο ή να κατασκευάσουμε καταλόγους (indexes). Αυτό είναι αρκετά χρονοβόρο αρχικά, αλλά μετά την ολοκλήρωση της επεξεργασίας, η αναζήτηση είναι πολύ γρηγορότερη. Παραθέτουμε ακολούθως ορισμένες από τις πιο χρησιμοποιούμενες τεχνικές:

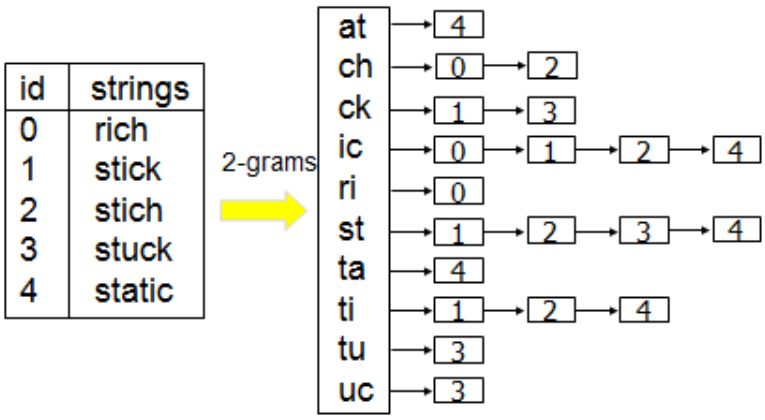
2.1.2.2.1 N-grams & Δένδρα Προθεμάτων

Με τον όρο n-gram[8] εννοούμε μία υπακολουθία μήκους n, μίας δεδομένης συμβολοσειράς. Τα ngrams μοναδιαίου μήκους αναφέρονται ως unigrams, αυτά που έχουν μήκος δύο ως bigrams κοκ. Τα ngrams αξιοποιούνται ιδιαίτερα συχνά για δημιουργία στατιστικών μοντέλων καθώς και επεξεργασία και μοντελοποίηση ομιλίας, αναγνώριση φωνής κ.ά.
Έστω ότι εξετάζουμε grams μήκους q.Εύκολα υπολογίζουμε ότι αν η edit distance μεταξύ δύο strings s1 και s2 είναι μικρότερη από k, τότε ο αριθμός των κοινών grams που έχουν τα strings είναι μεγαλύτερος ή ίσος από[9] (|s1|-q+1)-k*q. Διαπιστώνουμε λοιπόν ότι αναλογικά με την edit distance, μπορούμε να χρησιμοποιήσουμε τον αριθμό των κοινών grams ως μία μετρική για approximate string matching. Προς τούτο είναι αναγκαίο να δημιουργηθεί ένας inverted index με όλα τα grams των λέξεων τα οποία υπάρχουν στη βάση και τις λέξεις στις οποίες αυτά αντιστοιχούν.

[8] http://en.wikipedia.org/wiki/N-gram
[9] Efficient Approximate Search on String Collections, Marios Hadjieleftheriou, Chen Li

Εάν εξετάζουμε φερ ειπείν 2-grams, όλα τα πιθανά grams για τη λέξη rich είναι ri,ic,ch. Μπορούμε λοιπόν να αντιστοιχίσουμε το gram “ri” με τη λέξη rich αλλά και με όποια ακόμη λέξη της βάσης μας το περιέχει. Εάν αναζητούμε ένα string που περιέχει το gram “ri”, δεν χρειάζεται να αναζητήσουμε όλα τα string στη βάση μας αλλά μόνο αυτά με τα οποία έχει αντιστοιχηθεί το gram μας. Αυτή η τεχνική δημιουργίας αντεστραμμένων λιστών με grams φαίνεται καλύτερα αν εξετάσουμε το παρακάτω σχήμα:

Σχήμα 1:

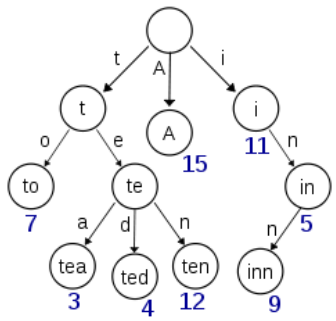


Κάνοντας χρήση αντεστραμμένων λιστών διαπιστώνουμε ότι μπορούμε να μειώσουμε κατά πολύ το χρόνο αναζήτησης σε μεγάλα datasets, διότι δεν εξετάζουμε όλα τα strings στη βάση μας αλλά μόνο αυτά που έχουν κοινά grams με τον όρο που αναζητούμε. Επίσης γλιτώνουμε το κόστος εφαρμογής κάποιου αλγορίθμου σύγκρισης για κάθε στοιχείο της βάσης, γεγονός που συνιστά σημαντικό πλεονέκτημα έναντι των on-line αλγορίθμων για μεγάλο πλήθος δεδομένων.

Τις αντεστραμμένες λίστες μπορούμε να τις αναπαραστήσουμε και ως **tries**. Με τον όρο **trie** ή **prefix tree**, εννοούμε μία διατεταγμένη δενδρική δομή δεδομένων που χρησιμοποιείται συνήθως για την αποθήκευση strings. Αντίθετα με ένα δυαδικό δένδρο, κανένας κόμβος στο δένδρο δεν αποθηκεύει το κλειδί (key) που σχετίζεται με αυτό τον κόμβο. Όλοι οι απόγονοι ενός κόμβου έχουν ένα κοινό πρόθεμα της συμβολοσειράς που σχετίζεται με αυτό τον κόμβο, ενώ η ρίζα είναι συσχετισμένη με το κενό string.

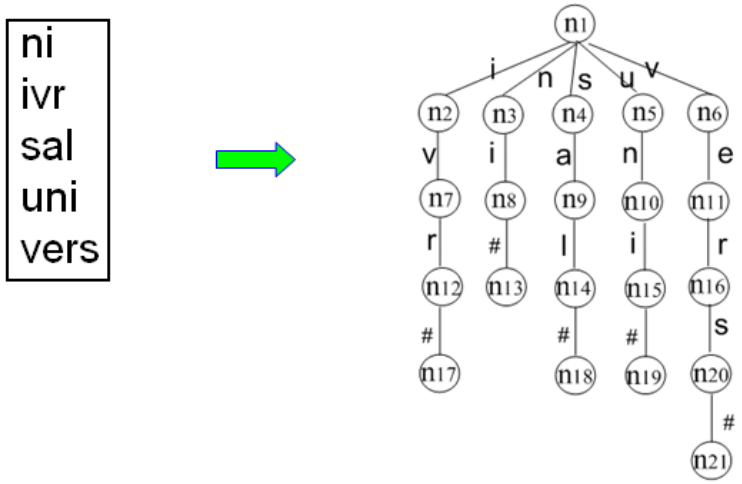
Οι τιμές συνήθως δεν σχετίζονται με κάθε κόμβο αλλά μόνον με τα φύλλα και με μερικούς εσωτερικούς κόμβους που αντιστοιχούν σε κλειδιά ενδιαφέροντος. Για παράδειγμα, έστωσαν τα κλειδιά "A", "to", "tea", "ted", "ten", "i", "in", and "inn". Μπορούμε να κατασκευάσουμε το παρακάτω trie:

Σχήμα 2:



Παρατηρούμε ότι όπως αναφέρθηκε, μόνο τα κλειδιά συνοδεύονται από τιμές, ενώ όλοι οι απόγονοι για κάθε κόμβο έχουν κοινό πρόθεμα με τον κόμβο (τα παιδιά δηλαδή ενός κόμβου είναι strings που έχουν ως prefix τον κόμβο –πχ το "tea" περιλαμβάνει το "te" που ανήκει στον πατέρα-).
Αντίστοιχα , εξετάζουμε πώς τα παρακάτω grams[10] μπορούν να αναπαρασταθούν ως tries:

Σχήμα 3:



Στο παραπάνω σχήμα, ο κόμβος n1 είναι το κενό, ο κόμβος n2 το "i", ο n7 το "iv" κ.ο.κ., ενώ τα φύλλα τερματίζουν με το χαρακτήρα (#). Χρησιμοποιώντας tries, η αναζήτηση ενός κλειδιού μήκους m στην χειρότερη περίπτωση έχει πολυπλοκότητα O(m). Αν n είναι τα στοιχεία του δένδρου, ο αλγόριθμος πραγματοποιεί O(log(n)) συγκρίσεις κλειδιών, και άρα στη χειρότερη περίπτωση η **χρονική πολυπλοκότητα** του αλγορίθμου είναι **O(m*logn)**.

2.1.2.2.2 Grams μεταβλητού μεγέθους

Μία σημαντική βελτίωση στα n-grams είναι η εισαγωγή grams μεταβλητού μήκους, ανάμεσα σε μία ελάχιστη και μέγιστη τιμή αντίστοιχα. Με αυτό τον τρόπο, μπορούμε να μειώσουμε το μέγεθος του καταλόγου και το χρόνο εκτέλεσης . Επίσης, είναι ιδιαίτερα εύκολο να υιοθετηθεί το σχήμα από πολλούς και διαφορετικούς αλγορίθμους. Τα grams μεταβλητού μήκους (variable length grams) ονομάζονται **vgrams**.
Έστω S μία συλλογή από strings. Στόχος μας είναι να παράγουμε ένα gram dictionary D (λεξικό από grams), αποτελούμενο από vgrams τα οποία κυμαίνονται από ένα ελάχιστο μέγεθος q_{min} έως και ένα μέγιστο μέγεθος q_{max}[11]. Αν ένα gram g_1 είναι κανονικό πρόθεμα ενός gram g_2,τότε αυτό καλείται **prefix gram** του g_2, ενώ το g_2 καλείται extended gram του g_1. Για παράδειγμα, το gram "hi" είναι prefix gram του "hit", ενώ το δεύτερο είναι extended gram του πρώτου καθώς «παράγεται» από αυτό. Αναπαριστούμε

[10] Efficient Approximate Search on String Collections, Chen Li, Marios Hadjieleftheriou
[11] Θα μπορούσαμε να παράγουμε το λεξικό μας ανεξάρτητα από το σύνολο strings που διαθέτουμε, όμως επιλέγουμε τη λύση αυτή για λόγους απόδοσης.

το λεξικό μας D ως trie. Προκειμένου να ξεχωρίσουμε ένα gram από τα extended grams του, στο τέλος κάθε gram προσθέτουμε ένα ειδικό σύμβολο, που στην προκειμένη περίπτωση είναι το σύμβολο (#). Κάθε μονοπάτι στο trie από την ρίζα σε ένα φύλλο αντιστοιχεί σε ένα gram στο D (αγνοούμε το #). Το gram αυτό λέγεται **αντίστοιχο (corresponding) gram** του φύλλου. Αντίστροφα, για κάθε gram στο D υπάρχει ένα **αντίστοιχο μονοπάτι** από τη ρίζα στο αντίστοιχο φύλλο.

Προκειμένου να παράγουμε ένα λεξικό για συγκεκριμένο μήκος gram q, χρησιμοποιούμε ένα κυλιόμενο παράθυρο μήκους q. Για να παράγουμε ένα λεξικό από vgrams, χρησιμοποιούμε μεν ένα κυλιόμενο παράθυρο, ωστόσο το μήκος του είναι μεταβλητό και εξαρτάται από το string s που εξετάζουμε και τα grams που υπάρχουν ήδη στο D. Αναδρομικά, σε κάθε βήμα, παράγουμε ένα gram για τη μέγιστη υπακολουθία (ξεκινώντας από την τρέχουσα θέση), η οποία ταιριάζει με ένα gram του D. Αν κανένα τέτοιο gram δεν υπάρχει, παράγουμε ένα gram μήκους q_{min}. Με τη μέθοδο αυτή, και αγνοώντας τα strings τα οποία έχουμε ήδη χρησιμοποιήσει, εφαρμόζουμε τον παρακάτω αλγόριθμο[12]:

Algorithm: VGEN
Input: Gram dictionary $\mathcal{D}$, string s, bounds q_{min}, q_{max}
Output: a set of positional grams for s

```
(1)  position p = 1; VG = empty set;
(2)  WHILE (p ≤ |s| − q_min + 1) {
(3)      Find a longest gram in D using the trie to match a
             substring t of s starting at position p;
(4)      IF (t is not found) t = s[p, p + q_min − 1];
(5)      IF (positional gram (p, t) is not subsumed by any
             positional gram in VG)
(6)          Insert (p, t) to VG;
(7)      p = p + 1;
     }
(8)  RETURN VG;
```

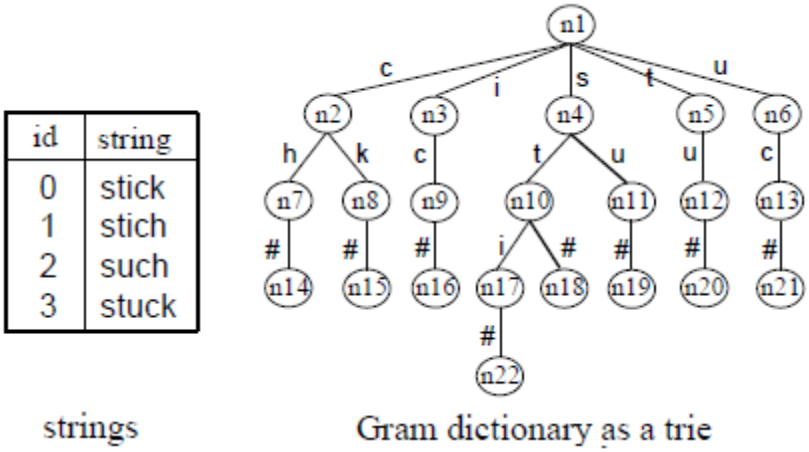
Η ανάλυση του αλγορίθμου έχει ως εξής: αρχικά θέτουμε την θέση του δείκτη μας στον πρώτο χαρακτήρα του s. Σε κάθε βήμα, από την τρέχουσα θέση, αναζητούμε για το μέγιστο substring του s το οποίο ανήκει στο D, χρησιμοποιώντας το trie που κατασκευάσαμε για την απεικόνισή του. Αν δεν εντοπίσουμε τέτοιο substring θεωρούμε ένα substring μήκους q_{min} που ξεκινά από αυτή τη θέση. Ακολούθως, εξετάζουμε αν το substring που επιλέξαμε είναι κανονικό substring ενός από τα ήδη δημιουργηθέντα substrings (δεδομένης της θέσης τους στο s). Αν είναι δεν παράγουμε ένα positional gram για αυτό το substring, καθώς έχει ήδη προστεθεί από προηγούμενο positional gram. Μετακινούμε το δείκτη μία θέση δεξιά, μέχρι η θέση να είναι μεγαλύτερη από $|s| - q_{min} + 1$. Το σύνολο από positional grams που παράγονται για το s, συμβολίζεται με VG(s,D,q_{min},q_{max}), ή απλούστερα VG(s).

Για παράδειγμα, ας θεωρήσουμε ως string s τη συμβολοσειρά “universal”, και το gram dictionary D {n1,1vr,sal,un1,vers}. Θέτουμε q_{min}=2 και q_{max}=4. Αρχικά θέτουμε p=1 και G={} και ο αλγόριθμος εκκινεί στον πρώτο χαρακτήρα “u”. Το μεγαλύτερο substring του s που ξεκινά με u και ανήκει στο λεξικό D είναι το “uni”. Ως εκ τούτου ο αλγόριθμος παράγει το positional gram (1, uni) το οποίο και εισάγει στο

[12] Vgram: Improving performance of approximate queries on string collections using variable-length grams, Chen Li, Bin Wan, Xiaochun Yang.

VG. Αντιστοίχως, μεταβαίνει στον επόμενο χαρακτήρα “n”. Το longest substring που περιέχεται στο D και αρχίζει με n είναι το “ni”. Ωστόσο, επειδή το positional gram (2,ni) ουσιαστικά περιλαμβάνεται στο (1,uni), (το “ni” είναι substring του “uni”), ο αλγόριθμος δεν το εισάγει στο γράφο. Συνεπώς προχωρά στον επόμενο χαρακτήρα του string, το “i”. Επειδή δεν υπάρχει substring που να ξεκινάει με αυτό τον χαρακτήρα και να ταιριάζει με κάποιο gram του λεξικού D, ο αλγόριθμος παράγει ένα positional gram (3,iv) ελαχίστου μήκους (q_{min}=2). Η διαδικασία επαναλαμβάνεται μέχρι το χαρακτήρα που βρίσκεται στην θέση (|s|-q_{min}+2), δηλαδή τον χαρακτήρα l. Το τελικό σύνολο από positional grams (VG) είναι το εξής:VG = {(1,uni),(3,iv),(4,vers),(7,sal)}. Αντίστοιχα, εξετάζουμε το trie που προκύπτει για το παρακάτω σύνολο από strings:

Σχήμα 4:



Ένα πρόβλημα το οποίο ανακύπτει είναι η κατασκευή ενός αποτελεσματικού λεξικού από grams δεδομένης μιας συλλογής από strings. Προκειμένου να το επιλύσουμε, μπορούμε να χρησιμοποιήσουμε έναν αλγόριθμο, ο οποίος χωρίζεται σε δύο κύρια βήματα. Στο πρώτο βήμα εξετάζουμε τη συχνότητα του κάθε gram (πόσα grams εμφανίζονται) και στο δεύτερο βήμα επιλέγουμε τα grams με τη μικρότερη συχνότητα.

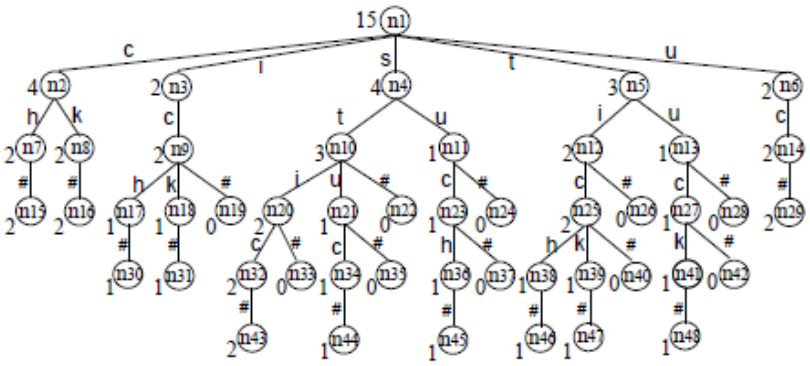
Βήμα πρώτο:

Προκειμένου να εξετάσουμε τις συχνότητες εμφάνισης του κάθε gram, ένας τρόπος είναι να παράξουμε όλα τα grams μήκους από q_{min} έως q_{max} και ακολούθως να μετρήσουμε τη συχνότητα του κάθε gram. Ένας υπολογιστικά αποτελεσματικότερος τρόπος είναι να χρησιμοποιήσουμε ένα trie προκειμένου να μετρήσουμε τις συχνότητες. Το trie αυτό το καλούμε **frequency trie** (trie συχνοτήτων). Ο αλγόριθμος αποφεύγει να παράγει όλα τα grams, βασιζόμενος στην εξής παρατήρηση: για κάθε συμβολοσειρά s, για κάθε positional q-gram (p,q), υπάρχει ένα positional gram (p,g’) για το extended q_{max}-gram g’. Για παράδειγμα, για τη συμβολοσειρά “university” και το positional 4-gram αυτής (2,niv), με q_{min} =2 και q_{max}=4, υπάρχει ακόμη το positional gram (2,nive) που ξεκινά στην ίδια θέση. Συνεπώς μπορούμε να παράγουμε q_{max}-grams για τα strings αλλά και να υπολογίσουμε τη συχνότητα των grams, παράγοντας μόνο αυτά (και όχι όλα τα μικρότερα).

Η διαδικασία που ακολουθούμε έχει ως εξής: κάθε κόμβος n στο trie συχνοτήτων, έχει μία τιμή για τη συχνότητα, την οποία συμβολίζουμε με n.freq. Αρχικά όλοι οι κόμβοι είναι κενοί. Για κάθε string s, παράγουμε όλα τα positional q_{maz}-grams, και για κάθε ένα από αυτά, εντοπίζουμε το αντίστοιχο φύλλο και το εισάγουμε στο trie αν το gram δεν υπάρχει (έχει συχνότητα 0). Ακολούθως, για κάθε κόμβο στο μονοπάτι από τη ρίζα στο αντίστοιχο φύλλο, συμπεριλαμβανομένου και του φύλλου, αυξάνουμε τη συχνότητά του (n.freq) κατά ένα. Σε κάθε ν-οστό κόμβο στο μονοπάτι, όπου $q_{min} <= ν < q_{max}$ δημιουργούμε ένα φύλλο προσθέτοντας μία ακμή με το σύμβολο τέλους #, αν ο νέος κόμβος-φύλλο δεν υπάρχει. Στον κόμβο αυτό δεν αυξάνουμε τον μετρητή συχνότητας καθώς η αύξηση έχει ήδη γίνει στον πατέρα. Συνεπώς, για κάθε θέση στο διάστημα $|s|-q_{max}+2$ … $|s|-q_{min}+1$ του s, παράγουμε ένα positional gram μήκους |s|-p+1 και επαναλαμβάνουμε την παραπάνω διαδικασία. Για παράδειγμα για το string “university”, παράγουμε τα positional grams (8,ity) και (9,ty) μήκους ανάμεσα σε 2 και 3 και αυξάνουμε το μετρητή στο trie. Αντίστοιχα, για το string set του σχήματος 4 παραθέτουμε το παραγόμενο frequency trie:

Σχήμα 5:

id	string
0	stick
1	stich
2	such
3	stuck



gram-frequency trie

Στο προηγούμενο σχήμα, η τιμή 2 στον κόμβο n_{43} σημαίνει ότι το gram “stic” εμφανίζεται συνολικά 2 φορές στο string set, ενώ η τιμή 3 του κόμβου n_{10} υποδηλώνει ότι το gram st εμφανίζεται 3 φορές.

Βήμα δεύτερο:

Στο βήμα αυτό κλαδεύουμε (prune) το trie συχνοτήτων και χρησιμοποιούμε τα grams που παραμένουν προκειμένου να σχηματίσουμε το λεξικό μας. Οι βασικές αρχές που ακολουθούμε είναι οι εξής:

- Διατηρούμε όσο γίνεται μικρότερο το μέγεθος των grams: Αν το gram g έχει μικρή συχνότητα εμφάνισης, αφαιρούμε από το trie όλα τα extended grams του g.
- Αν ένα gram εμφανίζει μεγάλη συχνότητα, διατηρούμε ορισμένα μόνο από τα extended grams του. Έτσι αναμένεται ότι ο αριθμός των strings που παράγουν αυτό το gram από τον αλγόριθμο VGEN θα μειωθεί σημαντικά.

Τα παραπάνω συνοψίζονται στην παρακάτω συνάρτηση:

```
FUNCTION Prune(Node n, Threshold T)
1.   IF (each child of n is not a leaf node) {
         // the root→ n path is shorter than q_min
2.       FOR (each child c of n);
3.           CALL Prune(c,T); // recursive call
4.       RETURN;
5.   }
     // a gram corresponds to the leaf-node child of n
6.   L = the (only) leaf-node child of n;
7.   IF (n.freq ≤ T) {
8.       Keep L, and remove other children of n;
9.       L.freq = n.freq;
10.  }
11.  ELSE {
12.      Select a maximal subset of children of n (excluding L),
             so that the summation of their freq values and
             L.freq is still not greater than T;
13.      Add the freq values of these children to
             that of L, and remove these children from n;
14.      FOR (each remaining child c of n excluding L)
15.          CALL Prune(c, T); // recursive call
16.  }
```

Διαλέγουμε ένα όριο T και κλαδεύουμε το trie αντιστοίχως.

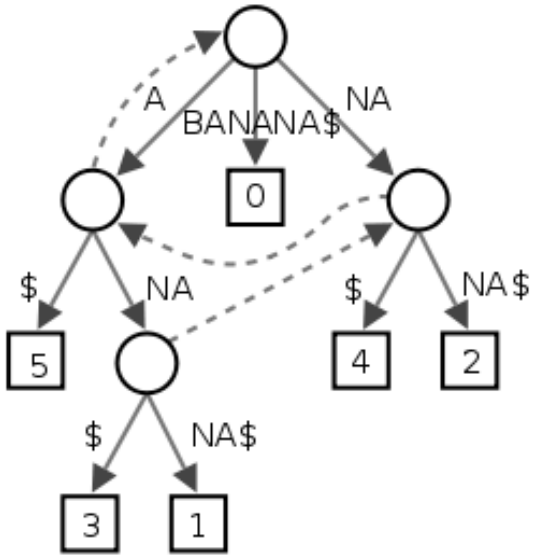
2.1.2.2.3 Δένδρα Επιθέματος

Μία εναλλακτική λύση για την αποτελεσματική αποθήκευση και διαχείριση συμβολοσειρών είναι τα **suffix trees** (δένδρα επιθέματος), που συχνά αναφέρονται και ως **PAT trees**. Το suffix tree ενός string S περιλαμβάνει ακμές που έχουν ως labels, strings, τέτοια ώστε, κάθε επίθεμα (suffix) του S να αντιστοιχεί σε ένα και μόνο ένα μονοπάτι από τη ρίζα στο αντίστοιχο φύλλο. Η κατασκευή ενός suffix tree έχει χωρική και χρονική πολυπλοκότητα γραμμική ως προς το μήκος του S. Ωστόσο, μετά την κατασκευή του, πολλές εργασίες μπορούν να επιταχυνθούν. Για τα παρακάτω, υποθέτουμε ότι έχουμε μία σταθερή αλφάβητο. Αν έχουμε ένα σύνολο από strings D ={S_1,S_2,…,S_n} με συνολικό μήκος n=$\sum_1^k |n_i|$ τότε ισχύει για συμβολοσειρές:

- Για να εξετάσουμε αν P μήκους m είναι substring ενός άλλου string απαιτείται O(m).
- Για να εντοπίσουμε την πρώτη εμφάνιση των patterns P_1,..,P_q ,συνολικού μήκους m,ως substrings, απαιτείται χρόνος O(m).
- Για να εντοπίσουμε όλες τις z εμφανίσεις των patterns P_1,..,P_q ,συνολικού μήκους m,ως substrings, απαιτείται χρόνος O(m+z).
- Για να αναζητήσουμε μία κανονική έκφραση P, απαιτείται ημιγραμμικός χρόνος.

Ακολούθως παραθέτουμε το suffix tree[13] για τη συμβολοσειρά “BANANA”. Κάθε substring τερματίζει με τον χαρακτήρα $ και τα έξι μονοπάτια από την ρίζα έως τα φύλλα (που εικονίζονται ως τετράγωνα), αντιστοιχίζουν σε έξι επιθέματα (suffixes), τα A$,NA$, ANA$, NANA$,ANANA$ και BANANA$.

Σχήμα 6:



2.2 Ταξινόμηση και Ανεύρεση Πληροφοριών

Η ανεύρεση πληροφοριών (information retrieval–IR) είναι ο κλάδος της επιστήμης των υπολογιστών που ασχολείται με την αναζήτηση κειμένων, πληροφοριών μέσα σε κείμενα, metadata κειμένων, καθώς και με την αναζήτηση σε σχεσιακές βάσεις δεδομένων και στον παγκόσμιο Ιστό. Μία διαδικασία ανεύρεσης πληροφορίας ξεκινά τη στιγμή που κάποιος χρήστης εισάγει ένα **ερώτημα** στο σύστημα. Το ερώτημα αυτό δεν προσδιορίζει μοναδικά ένα **αντικείμενο** αλλά διαφορετικά αντικείμενα μπορούν να ταιριάζουν στο ερώτημα, έχοντας το καθένα διαφορετικούς βαθμούς **συνάφειας** (**relevancy**) με το ερώτημα. Ένα αντικείμενο είναι μία οντότητα που αναπαρίσταται με κάποια μορφή πληροφορίας σε μία **βάση δεδομένων**. Τα περισσότερα συστήματα IR, υπολογίζουν ένα αριθμητικό σκορ σχετικά με το πόσο καλά ένα αντικείμενο της βάσης ανταποκρίνεται σε δεδομένο ερώτημα και ταξινομούν τα αντικείμενα βάσει αυτής της τιμής. Στη συνέχεια, κάποιο ποσοστό των αντικειμένων με τη μεγαλύτερη βαθμολογία

[13] http://en.wikipedia.org/wiki/Suffix_tree

παρουσιάζεται στο χρήστη. Ακολούθως, εξετάζουμε ορισμένα από τα μέτρα[14] που χρησιμοποιούνται προκειμένου να αξιολογηθεί η απόδοση ενός IR συστήματος.

2.2.1 Precision

Με τον όρο precision (ακρίβεια), εννοούμε το ποσοστό των documents που ανευρέθηκαν τα οποία είναι σχετικά με τις πληροφορίες που ο χρήστης ζήτησε. Συνεπώς προκύπτει ότι:

$$precision = \frac{\left|\{relevant\ \ documents\} \cap \{retrieved\ \ documents\}\right|}{\left|\{retrieved\ \ documents\}\right|}$$

Ουσιαστικά πρόκειται την θετική πρόβλεψη που έχουμε για ένα συμβάν, την ακρίβεια της μέτρησης. Για παράδειγμα, αν 50 documents ανευρεθούν ως απάντηση σε κάποιο ερώτημα και 35 από αυτά είναι σχετικά, τότε το precision όταν έχουν ανευρεθεί και τα 50 είναι P_{50}=70%. Μπορούμε να την υπολογίσουμε και για ένα δοσμένο όριο, θεωρώντας μόνο τα καλύτερα n αποτελέσματα. Στην περίπτωση αυτή μιλούμε για **precision at n** ή P@n.

2.2.2 Recall

Κατ' αναλογία με το precision, το recall αναφέρεται στο ποσοστό των μη σχετικών documents που ανευρέθηκαν από όλα τα διαθέσιμα μη σχετικά documents. Έχουμε δηλαδή ότι:

$$recall = \frac{\left|\{relevant\ \ documents\} \cap \{retrieved\ \ documents\}\right|}{\left|\{relevant\ \ documents\}\right|}$$

Ουσιαστικά αναφέρεται στην πιθανότητα ένα συναφές document να επιστραφεί ως απάντηση, δηλαδή μετρά το βαθμό στον οποίο η ανεύρεση πληροφορίας είναι εξαντλητική και ποσοτικοποιεί την κάλυψη του συνόλου απαντήσεων. Στο παράδειγμα που χρησιμοποιήσαμε για το precision, αν υπάρχουν 70 συναφή documents, το recall όταν ανευρεθούν τα πρώτα 50 θα είναι R_{50}=50%.

[14] I. H. Witten, A. Mo®at, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. http://en.wikipedia.org/wiki/Information_retrieval

2.2.3 F-measure

Το μέτρο αυτό, το οποίο είναι επίσης γνωστό και με τον όρο **balanced F-score** είναι ο αρμονικός μέσος των precision και recall και δίδεται από τη σχέση:

$$F = \frac{2 \cdot precision \cdot recall}{(precision + recall)}$$

Αν θεωρήσουμε έναν πραγματικό θετικό αριθμό β, μία γενικότερη σχέση για τον υπολογισμό του F-score είναι η παρακάτω:

$$F_\beta = \frac{(1+\beta^2) \cdot precision \cdot recall}{(\beta^2 \cdot precision + recall)}$$

2.2.4 Ταξινόμηση & Ομοιότητα

Πέραν της συνάφειας που έχουν τα ανευρεθέντα (retrieved) documents με το ερώτημα που έθεσε ο χρήστης, πολύ συχνά μας ενδιαφέρει να εξετάσουμε την ομοιότητα που έχουν δύο ή περισσότερα documents μεταξύ τους, αλλά και το ποίο document είναι περισσότερο πιθανό να ανταποκρίνεται καλύτερα σε ένα ερώτημα του χρήστη. Ένας τρόπος προκειμένου να το πετύχουμε αυτό είναι η χρήση διανυσμάτων κειμένων (**document vectors**[15]) , τα οποία συμβολίζουμε με **<$w_{d,t}$>**.

Θεωρούμε το σύνολο των λέξεων που περιλαμβάνονται σε όλα τα documents. Έτσι μπορούμε να κατασκευάσουμε για κάθε document ένα document vector, το οποίο θα έχει 0 ή 1 στην ανάλογη θέση, αν η αντίστοιχη λέξη υπάρχει ή δεν υπάρχει στο κείμενο. Για παράδειγμα, έστωσαν τα documents 1,2 τα οποία περιλαμβάνουν τις εξής φράσεις (και μόνον αυτές):

D_1="The quick brown fox jumps"

D_2="The fox ate the hen"

Εάν αγνοήσουμε τα άρθρα "the" ως στοιχεία τα οποία ως επί το πλείστον δεν προσφέρουν καμία πληροφορία, μπορούμε να σχηματίσουμε τον εξής πίνακα διανυσμάτων:

d	quick	brown	fox	ate	hen	jumps
1	1	1	1	0	0	1
2	0	0	1	1	1	0

[15] I. H. Witten, A. Mo®at, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*

Έστω τώρα ότι αναζητούμε τη φράση X="brown fox" και επιθυμούμε να λάβουμε το πιο συναφές εκ των δύο κειμένων ως απάντηση. Μπορούμε να αναπαραστήσουμε το X με το διάνυσμα (0,1,1,0,0,0). Προκειμένου να συγκρίνουμε την συνάφεια των κειμένων, θεωρούμε το εσωτερικό τους γινόμενο με το διάνυσμα του X. Ως γνωστόν, το εσωτερικό γινόμενο δύο διανυσμάτων $X=<x_i>$ και $Y=<y_i>$ δίνεται από τη σχέση:

$$X \cdot Y = \sum_{i=1}^{n} x_i y_i$$

Συνεπώς προκύπτει

$$M(X,D_1)=(1,1,1,0,0,1)*(0,1,1,0,0,0)=2$$
$$M(X,D_1)=(0,0,1,1,1,0)*(0,1,1,0,0,0)=1$$

και άρα το document 1 είναι η καλύτερη απάντηση στο ερώτημα. Η μέθοδος αυτή, παρόλο που εισάγει ένα νέο μέτρο ομοιότητας (similarity measure), έχει σημαντικά μειονεκτήματα, όπως φερ ειπείν το γεγονός ότι δεν λαμβάνει υπ' όψιν τον αριθμό των εμφανίσεων της κάθε λέξης στο κείμενο, το πόσο σημαντική είναι δηλαδή η κάθε λέξη για το νοηματικό περιεχόμενο του κειμένου. Μία λύση θα ήταν αντί για μονάδα να χρησιμοποιήσουμε τον αριθμό των εμφανίσεων της κάθε λέξης στο document. Σε κάθε περίπτωση, πρέπει να εισάγουμε ένα βάρος για κάθε όρο (**term**) μέσα στο document, το οποίο συμβολίζουμε με $\mathbf{w_{d,t}}$ και ονομάζουμε **document-term weight** και ένα ακόμη βάρος για το διάνυσμα της ερώτησης το οποίο θα συμβολίζουμε με $\mathbf{w_{q,t}}$. Το μέτρο ομοιότητας θα είναι το γινόμενο των δύο[16], δηλαδή θα ισχύει:

$$M(Q, D_d) = Q \cdot D_d = \sum_{t=1}^{n} w_{q,t} \cdot w_{d,t}$$

Μία ιδιαίτερα σημαντική παρατήρηση είναι ότι η σημασία ενός term, είναι αντιστρόφως ανάλογη της συχνότητας εμφάνισής του. Για παράδειγμα, τα άρθρα σε ένα κείμενο δεν μας δίδουν ιδιαίτερα στοιχεία διότι υπάρχουν σε όλα τα κείμενα και δυσχεραίνουν την αξιολόγηση των κειμένων ως προς την συνάφεια με ένα ερώτημα. Συνεπώς, αν w_t το «βάρος» ενός όρου t, το οποίο ισοδυναμεί με το πόσο σημαντικός είναι ο όρος για το κείμενο, και f_t ο αριθμός των documents που περιέχουν τον όρο, μπορούμε να γράψουμε:

$$w_t = \frac{1}{f_t} \quad (1)$$

Το w_t μπορούμε να το υπολογίσουμε και με τους παρακάτω τρόπους:

$$w_t = \log(1 + \frac{N}{f_t}) \quad (2)$$

[16] I. H. Witten, A. Mo®at, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*

$$w_t = \log(1 + \frac{f^m}{f_t}) (3)$$

$$w_t = \log(\frac{N - f_t}{f_t}) (4)$$

Στα παραπάνω, συμβολίζουμε με N τον αριθμό των documents που έχουμε στη διάθεσή μας, και με f^m τη μεγαλύτερη από τις συχνότητες εμφάνισης μέσα στο κείμενο ($f_{d,t}$). Η πιο ευρέως χρησιμοποιούμενη σχέση εκ των τεσσάρων είναι η (2). Το πλεονέκτημά της έναντι της (1) είναι ότι μετριάζεται λόγω του λογαρίθμου η επίδραση στα αποτελέσματα ενός π.χ. όρου που εμφανίζεται δύο φορές έναντι ενός όρου που εμφανίζεται μία. Επιπλέον, αν θεωρήσουμε ότι $\mathbf{r_{d,t}}$ είναι η σχετική συχνότητα του όρου (**relative term frequency**), μπορούμε να πολλαπλασιάσουμε το w_t me $r_{d,t}$ προκειμένου να πάρουμε το **document-term weight** $\mathbf{w_{d,t}}$, που απεικονίζει το βάρος ενός term για δεδομένο document. Αντίστοιχα, μπορούμε να υπολογίσουμε την $r_{d,t}$ ως:

$r_{d,t}=1$ (5)
$r_{d,t}=f_{d,t}$ (6)
$r_{d,t}=1+\log(f_{d,t})$ (7)

Συνεπώς, τα document vectors είναι δυνατόν να υπολογιστούν είτε ως $w_{d,t}=r_{d,t}$ είτε ως $w_{d,t}=r_{d,t}\cdot w_t$.
Η δεύτερη σχέση είναι γνωστή ως ο κανόνας TF x IDF. Δηλαδή το βάρος για να συγκεκριμένο document d ενός όρου t ισούται με τη συχνότητα του όρου επί την αντίστροφη συχνότητα του όρου στο document. Μία ευριστική μέθοδος ομοιότητας ονομάζεται «TFxIDF» όταν χρησιμοποιεί την συχνότητα του όρου $f_{d,t}$ μονοτονικά αυξανόμενη και την συχνότητα του όρου f_t μονοτονικά μειούμενη.

Καταλήγοντας, αξίζει να παρατηρήσουμε ότι συνήθως χρησιμοποιούνται κατά το ranking τα εξής μεγέθη:

$W_t=\log(1+N/f_t)$

$r_{d,t} = 1 + \log f_{d,t}$ $\qquad$ $r_{q,t} = 1$

$w_{d,t} = r_{d,t}$ $\qquad$ $w_{q,t} = r_{q,t} \cdot w_t$

Στην παρούσα εργασία, θα χρησιμοποιήσουμε τις εξής σχέσεις:

$w_1=(1 + \log f_{d,t})\cdot \log(1+N/f_t)$ (8)
$w_2=(f_{d,t}) \cdot \log(1+N/f_t)$ (9)

3

Υλοποίηση

Στην ενότητα αυτή αρχικά εξετάζουμε τα βασικά χαρακτηριστικά λειτουργίας της εφαρμογής DIANA, τους πίνακες που χρησιμοποιούνται κατά την αναζήτηση και τη σύνδεση μεταξύ των διαφορετικών τμημάτων της εφαρμογή. Ακολούθως, υλοποιούμε τα εξής:

- Τροποποιούμε τη μηχανή αναζήτησης προκειμένου να γίνεται χρήση του αλγορίθμου της απόστασης Levenshtein από την βάση δεδομένων, με την αντίστοιχη udf συνάρτηση.
- Κατασκευάζουμε μία σειρά από udf συναρτήσεις και διαδικασίες που επιτελούν λειτουργίες συναφείς με n-grams.
- Υλοποιούμε το μηχανισμό αναζήτησης αξιοποιώντας τις παραπάνω udf συναρτήσεις και κάνοντας χρήση μεθόδων που χρησιμοποιούν ευρετήρια grams.
- Τροποποιούμε το γραφικό περιβάλλον της εφαρμογής προκειμένου η αναζήτηση να γίνεται σε πραγματικό χρόνο, με χρήση των τεχνολογιών ajax και JQuery.

3.1 Περιγραφή λειτουργίας της εφαρμογής

Προτού προχωρήσουμε με περαιτέρω υλοποιήσεις, ήταν αναγκαίο να στηθεί ένας τοπικός server με υποστήριξη php, μία mysql βάση και να γίνει import μέρος της βάσης του Diana micro-T στη βάση μας. Το τελικό αποτέλεσμα ήταν ένα ακριβές[17] αντίγραφο του συστήματος που τρέχει στο DIANA micro-T. Πρόκειται για μία εφαρμογή η οποία έχει κατασκευαστεί με χρήση του yii modeling framework[18]. Το μοντέλο λειτουργίας της εφαρμογής είναι το "Model-View-Controller (MVC)".

[17] Αγνοούμε στο σημείο αυτό τις όποιες διαφορές επικοινωνίας του τοπικού server με τη βάση στο localhost με αυτές ενός πραγματικού συστήματος.
[18] http://www.yiiframework.com/

Το κύριο στοιχείο της εφαρμογής στο οποίο θα επικεντρώσουμε το ενδιαφέρον μας εντοπίζεται στη σελίδα /DianaTools/index.php?r=microtv4/index , το σημείο δηλαδή της ιστοσελίδας που αφορά στην αναζήτηση. Όταν ο χρήστης πληκτρολογήσει μία ακολουθία από όρους στη μπάρα αναζήτησης, σε γενικές γραμμές συμβαίνουν τα εξής:

- Η κλάση Microtv4Controller που ορίζεται στο αρχείο /controllers/Microtv4Controller.php, ανάμεσα σε άλλες λειτουργίες που επιτελεί, "ακούει" τις κινήσεις του χρήστη. Κάθε τέτοια κίνηση, ενεργοποιεί κάποιο action, το οποίο είναι μια συνάρτηση της προαναφερθείσας κλάσης. Για παράδειγμα, η συνάρτηση actionDetermine() ορίζει το action "determine" το οποίο ενεργοποιείται μετά το πάτημα του Enter όταν κάποιος πληκτρολογεί στο search box.
- Αυτό το action παίρνει την είσοδο του χρήστη (ό,τι είχε γραφτεί στο searchbox μεταφέρεται μέσω http με POST ή GET) και δημιουργεί ένα αντικείμενο της κλάσης Microtv4Search.
- Η κλάση Microtv4Search είναι μια κλάση "μοντέλου" και είναι υπεύθυνη για τη διαχείριση δεδομένων (π.χ. για την επικοινωνία με τη βάση). Η actionDetermine() της Microtv4Controller καλεί τη συνάρτηση determine() που έχουμε ορίσει στην Microtv4Search, η οποία ακολούθως αναλαμβάνει να ελέγξει αν τα keywords ήταν σωστά και να δώσει τις κατάλληλες τιμές στις μεταβλητές του αντικειμένου της Microtv4Search. Αν κάποιο από τα keywords είναι λάθος, τότε καλείται η ενέργεια "render" επί της σελίδας "suggestions". Κάθε σελίδα που γίνεται render αντιστοιχεί σε ένα "View".

Εξετάζουμε ακολούθως τη συνάρτηση determine() που καλείται στην κλάση Microtv4Search. Οι πίνακες στους οποίους γίνεται η αναζήτηση της πληροφορίας είναι κυρίως οι εξής:

- **microtv4_diseases**

microtv4_diseases

Field	Type	Null	Default
mirna_name	varchar(30)	No	
html_cloud	text	No	

Στον πίνακα αυτό κρατάμε τα ονόματα των microRNAs και τις ασθένειες με τις οποίες σχετίζονται.

- **microtv4_msynonyms**

microtv4_msynonyms

Field	Type	Null	Default
mima_id	varchar(20)	No	
mirna_name	varchar(20)	No	
synonym_name	varchar(20)	No	
mirna_id	smallint(5)	Yes	*NULL*

Το σύστημα DIANA micro-T παρουσίασε αλλαγές στην ονοματολογία από έκδοση σε έκδοση. Στον πίνακα διατηρούνται οι συνδέσεις μεταξύ αυτών των διαφορών

- **microtv4_mirnas**

microtv4_mirnas

Field	Type	Null	Default
name	varchar(20)	No	
alt_indicator	char(12)	Yes	*NULL*
seq	varchar(30)	No	
id	smallint(5)	No	

Στον πίνακα αυτό αποθηκεύονται όλα τα micro_RNAs, η ακολουθία τους και το όνομά τους.

- **microtv4_genes**

microtv4_genes

Field	Type	Null	Default
species	varchar(50)	No	
chromosome	varchar(10)	Yes	*NULL*
euro_id	varchar(20)	No	
biotype	varchar(30)	No	
strand	tinyint(1)	No	
transcript_id	varchar(30)	No	
name	varchar(30)	Yes	*NULL*
description	varchar(5000)	Yes	*NULL*
a_percentage	double	No	
t_percentage	double	No	
g_percentage	double	No	
c_percentage	double	No	
seq	varchar(30000)	No	
hugo_html	varchar(400)	Yes	*NULL*
id	smallint(5)	No	

Στον πίνακα διατηρούνται πληροφορίες σχετικές με τα διάφορα γονίδια (όπως το είδος το χρωμοσώματος, το αν το γονίδιο είναι ανθρώπινο κ.ο.κ.

- **microtv4_refseqs**

microtv4_refseqs

Field	Type	Null	Default
gene_euro_id	varchar(20)	Yes	*NULL*
gene_us_id	varchar(15)	Yes	*NULL*
seed_start	int(10)	No	
gene_id	smallint(5)	Yes	*NULL*

Ο πίνακας κρατά τα ids των microRNAs βάσει των ευρωπαϊκών και αμερικανικών προτύπων.

- **microtv4_interactions**

microtv4_keggs

Field	Type	Null	Default
kegg_euro_id	varchar(20)	No	
kegg_id	varchar(30)	No	
kegg_path	varchar(60)	No	
gene_id	smallint(5)	Yes	*NULL*

Εδώ αποθηκεύονται οι πληροφορίες για τις σχέσεις μεταξύ των διαφόρων microRNAs άρα και των αντιστοίχων πινάκων.

Ο βασικός πυρήνας της συνάρτησης determine, για την αναζήτηση δεδομένων, υλοποιείται στην αρχική έκδοση της πλατφόρμας με τον παρακάτω κώδικα:

determine() core:

Τμήμα αρχικής έκδοσης της συνάρτησης determine()

```
$sql_q = "  SELECT gene_euro_id,gene_us_id
            FROM microtv4_refseqs";
$sql_r = Yii::app()->getDb()->getCommandBuilder()->createSqlCommand($sql_q)-
>query();
foreach( $sql_r as $cur_res)
{
      $cur_dist = levenshtein($cur_res['gene_us_id'],$cur_tok);
      $temp =  array( array(  "term"=>$cur_res['gene_us_id'],
            "entity"=>"gene",
            "type"=>"gRef",
            "id"=>$cur_res['gene_euro_id'],
            "score"=>$cur_dist));
      foreach( $top_matches as $key=>$cur_match)
      {
            if($cur_dist <= $cur_match["score"])
            {
                  //insert
                  array_splice( $top_matches, $key, 0, $temp);
                  //pop the last
                  array_pop($top_matches);
                  break;
            }
      }
}
```

Ουσιαστικά διαλέγουμε εκείνα τα gene_euro_id, gene_us_id από τον πίνακα microtv4_refseqs, που έχουν τη μικρότερη edit distance (υπολογισμένη βάσει της Levenshtein distance) σε σχέση με τον όρο που εισήγαγε ο χρήστης ($cur_tok). Η συνάρτηση levenshtein() είναι υλοποιημένη στην php.Ο κώδικας αυτός, εκτελείται όχι μόνο για τον πίνακα microtv4_refseqs αλλά και για τον πίνακα microtv4_genes (για

τα attributes <euro_id> και <name>) και για τον πίνακα microtv4_mirnas (για τα attributes <name> και <alt_indicator>) αντιστοίχως. Δηλαδή ο παραπάνω κώδικας εκτελείται μέσα στην συνάρτηση determine συνολικά 5 φορές.

Έχοντας εξετάσει τα βασικά σημεία λειτουργίας της εφαρμογής, μπορούμε να θέσουμε σε πρώτο επίπεδο τους εξής **στόχους**:

- Προκειμένου να βελτιώσουμε τον χρόνο απόκρισης της αναζήτησης, θα επιδιώξουμε να μειώσουμε τις άσκοπες κλήσεις προς τη mysql καθώς και τον όγκο των εργασιών που η php εκτελεί. Ο βασικός λόγος για τον οποίο η php αποδίδει χειρότερα από την mysql είναι διότι, για την επιλογή των καλύτερων (συναφέστερων) αποτελεσμάτων, η ram που διαχειρίζεται ο buffer της php δεν επαρκεί για την επεξεργασία όλων των records της βάσης δεδομένων, με αποτέλεσμα να εξετάζονται τα records αυτά ανά ομάδες. Το κόστος μεταφοράς πληροφορίας από και προς την mysql συνεπώς, επιβραδύνει τη διαδικασία και ως εκ τούτου ένας βασικός στόχος μας θα είναι να «μεταφερθούν» όσο το δυνατόν περισσότερες λειτουργίες από την php στη mysql.
- Θα επιδιώξουμε να αναζητήσουμε την αποτελεσματικότερη μέθοδο για approximate string matching και παροχή προτάσεων στο χρήστη, και να υλοποιήσουμε τη μέθοδο αυτή, προκειμένου να βελτιωθεί ο χρόνος σε σχέση με τη χρήση της Levenshtein distance από την php.
- Θα προσπαθήσουμε να μειώσουμε τις καθυστερήσεις που οφείλονται στην υλοποίηση του κώδικα, όπως πχ η επανάληψη του κώδικα που παραθέσαμε προηγουμένως σε 5 σημεία, η οποία μπορεί να αποφευχθεί, οδηγώντας σε βελτιώσεις στο χρόνο εκτέλεσης.
- Θα ενσωματώσουμε στις όποιες αλλαγές και βελτιώσεις, ένα σύστημα παροχής προτάσεων πραγματικού χρόνου, με χρήση τεχνολογίας ajax στην πλατφόρμα αναζήτησης.

3.2 Το πακέτο flamingo

Το πακέτο λογισμικού flamingo αναπτύχθηκε από το τμήμα επιστήμης των υπολογιστών του πανεπιστημίου UC Irvine και περιέχει υλοποιημένους αλγορίθμους για approximate string matching. Είναι γραμμένο σε C++ , και , εκτός των προγραμμάτων που παρέχει, προσφέρει και τρείς mysql udfs για εργασίες συναφείς με edit distance. Οι udfs αυτές είναι οι εξής:

- **ed** : integer ed(string s_1, string s_2).
  Επιστρέφει την edit distance ανάμεσα στα s_1,s_2. Η συνάρτηση ωστόσο υποθέτει ότι και τα δύο strings έχουν τον ίδιο τύπο γραμμάτων, δηλαδή είναι και τα δύο γραμμένα με μικρά (μόνο) ή κεφαλαία (μόνο).
- **edth**: boolean edth(string s_1, string s_2, integer th).
  Επιστρέφει true αν η edit distance ανάμεσα στα strings s_1 και s_2 είναι μικρότερη ή ίση με th. Η συνάρτηση αυτή είναι σχετικά γρηγορότερη από την προηγούμενη καθώς η εκτέλεσή της σταματά μόλις η edit distance ξεπεράσει το όριο που θέτουμε. Και εδώ υποθέτουμε ότι τα strings είναι ίδιου τύπου (same case hypothesis).
- **edrec**: boolean edrec(string s, string rec, integer th).

Επιστρέφει true εάν υπάρχει κάποιο token (substring) στο string rec, που έχει edit distance μικρότερη ή ίση με th σε σχέση με το s. Το string rec σπάει σε tokens χρησιμοποιώντας τους εξής separators: " "(white space),",", και "." (dot). Η συνάρτηση υποθέτει ότι οι συμβολοσειρές που χρησιμοποιούμε είναι σε lowercase.

Παραθέτουμε ακολούθως από ένα παράδειγμα για την εκτέλεση της κάθε udf function.

```
mysql> SELECT ed('abc', 'ad');
2
mysql> SELECT edth('abc', 'abcd', 1);
1
mysql> SELECT edrec('ab', 'xx ad xx', 1);
1
```

Στα παραπάνω, οι boolean συναρτήσεις επιστρέφουν 0 και 1 για false και true αντίστοιχα. Μία αρχική βελτίωση που εφαρμόσαμε στον κώδικα της determine() ήταν να αντικαταστήσουμε την κλήση που γίνεται στη συνάρτηση levenshtein**()** της php με την udf **ed** του πακέτου flamingo.
Προκειμένου να συμβεί αυτό, ήταν αναγκαίο να κάνουμε compile το πακέτο στον υπολογιστή όπου έτρεχε ο mysql server και να ενσωματώσουμε τις udf στη Mysql. Ωστόσο, το flamingo δεν ήταν διαθέσιμο για Mac OS X που ήταν το λειτουργικό στο οποίο κάναμε τις δοκιμές μας. Ως εκ τούτου, ήταν αναγκαίο να κάνουμε ορισμένες αλλαγές στον κώδικα[19], προκειμένου να επεκτείνουμε το πακέτο για το νέο λειτουργικό.
Μετά την επέκταση του flamingo για υποστήριξη Mac OS X και την εγκατάσταση των udf, τροποποιήσαμε τη συνάρτηση determine() της κλάσης Microtv4Search όπως φαίνεται ακολούθως:

Τμήμα της συνάρτησης determine() με χρήση της edit distance (v1)

```
//Find the most similar gene ensembl ids or gene names.
$sql_q = "SELECT * FROM
      (
            (SELECT euro_id AS term, 'gene' AS entity,'ensGene' AS
type,euro_id AS id,ed(euro_id,'".$cur_tok."') AS score
            FROM microtv4_genes
            ORDER BY ed(euro_id,'".$cur_tok."')
            LIMIT 0,".Yii::app()->params['microtv4_num_of_suggs']."
            )
      UNION
            (SELECT name AS term, 'gene' AS entity,'gName' AS type,euro_id AS
id,ed(name,'".$cur_tok."') AS score
```

[19] Η βασική δυσκολία που έπρεπε να ξεπεράσουμε προκειμένου να λειτουργήσει το πακέτο είναι η ασυμβατότητα μεταξύ αρχιτεκτονικών x86_64 και i386 η έλλειψη ορισμένων βιβλιοθηκών από το λειτουργικό Mac OS X. Προκειμένου να ξεπεράσουμε αυτά τα εμπόδια, σε όλα αρχεία CMakeLists.txt του cmake αφαιρούμε τα flags –lrt, και τη βιβλιοθήκη rt. Αντίστοιχα, προσθέτουμε μία βιβλιοθήκη με όνομα gettime.h στο αρχείο /src/util/src . Η βιβλιοθήκη αυτή χρησιμοποιείται για τη διαχείριση του ρολογιού, η οποία διαφέρει από τη διαχείριση στις περισσότερες διανομές Linux. Ακολούθως αλλάζουμε το αρχείο looptimer.cc και το αρχείο time.h προκειμένου να συμπεριλάβουμε τη νέα βιβλιοθήκη μας και αφαιρούμε τα πακέτα τα οποία δεν υποστηρίζονται. Ο κώδικας για τη βιβλιοθήκη που προσθέσαμε είναι διαθέσιμος στην παρακάτω διεύθυνση:
http://web.imis.athena-innovation.gr/redmine/projects/recom/wiki.

```
            FROM microtv4_genes
            ORDER BY ed(name,'".$cur_tok."')
            LIMIT 0,".Yii::app()->params['microtv4_num_of_suggs']."
            )
      UNION
            (SELECT gene_us_id AS term, 'gene' AS entity,'gRef' AS
type,gene_euro_id AS id,ed(gene_us_id,'".$cur_tok."') AS score
            FROM microtv4_refseqs
            ORDER BY ed(gene_us_id,'".$cur_tok."')
            LIMIT 0,".Yii::app()->params['microtv4_num_of_suggs']."
            )
      UNION
            (SELECT name AS term, 'miRNA' AS entity,'mName' AS type,name AS
id,ed(name,'".$cur_tok."') AS score
            FROM microtv4_mirnas
            ORDER BY ed(name,'".$cur_tok."')
            LIMIT 0,".Yii::app()->params['microtv4_num_of_suggs']."
            )
      UNION
            (SELECT alt_indicator AS term, 'miRNA' AS entity,'mAlt' AS
type,name AS id,ed(alt_indicator,'".$cur_tok."') AS score
            FROM microtv4_mirnas
            ORDER BY ed(name,'".$cur_tok."')
            LIMIT 0,".Yii::app()->params['microtv4_num_of_suggs']."
            )
      )
      AS union_results
      ORDER BY score
      LIMIT 0,".Yii::app()->params['microtv4_def_res_on_page']."";
$sql_r = Yii::app()->getDb()->getCommandBuilder()->createSqlCommand($sql_q)-
>query();
foreach( $sql_r as $cur_res)
{

      $temp =  array( array(  "term"=>$cur_res['term'],
            "entity"=>$cur_res['entity'],
            "type"=>$cur_res['type'],
            "id"=>$cur_res['id'],
            "score"=>$cur_res['score']));
      foreach( $top_matches as $key=>$cur_match)
      {
            if($cur_res['score'] <= $cur_match["score"])

            {
                  //insert
                  array_splice( $top_matches, $key, 0, $temp);
                  //pop the last
                  array_pop($top_matches);
                  break;
            }
      }
}
```

Παρατηρώντας τον παραπάνω κώδικα μπορούμε να κάνουμε τις εξής **παρατηρήσεις**:

- Μειώθηκε σημαντικά το μέγεθος του κώδικα (καθώς με τις 5 επαναλήψεις, στην αρχική έκδοση, του τμήματος που παραθέσαμε στην προηγούμενη ενότητα, το μέγεθος του κώδικα ήταν τριπλάσιο).

- Πλέον δεν έχουμε 5 κλήσεις στη Mysql αλλά μόνο μία, οπότε μειώνεται σημαντικά ο χρόνος που δαπανάται σε μεταφορά δεδομένων μεταξύ php και mysql.
- Αντικαταστάθηκε η συνάρτηση levenshtein() με την udf συνάρτηση ed(), η οποία οδηγεί σε μείωση του χρόνου εκτέλεσης, μια και εκτελείται απευθείας από τη mysql και έχει περάσει από τη διαδικασία optimization της βάσης δεδομένων.
- Προκειμένου να μην εισάγουμε μεγάλο όγκο δεδομένων στους buffers, επιστρέφουμε τα N καλύτερα αποτελέσματα στο χρήστη, και η επιλογή γίνεται στη Mysql, μέσω του

```
LIMIT 0,".Yii::app()->params[]."";
```

Μετά τις παραπάνω αλλαγές στον κώδικα παρατηρήθηκε ότι η συνήθης αναζήτηση του χρήστη (μέση περίπτωση) είναι περίπου **δέκα φορές γρηγορότερη** από την αρχική.

3.3 Mysql udfs

Το επόμενο βήμα προκειμένου να βελτιώσουμε περαιτέρω τη μηχανή αναζήτησης, είναι να κατασκευάσουμε μία ομάδα από mysql udfs ώστε να υλοποιήσουμε το μηχανισμό αναζήτησης με χρήση ngrams. Υπάρχουν 3 τρόποι προκειμένου κανείς να εισάγει στη Mysql μία συνάρτηση udf:

1. Μέσω του ειδικού interface που παρέχει η mysql.
2. Κάνοντάς compile τον πηγαίο κώδικα των συναρτήσεων μαζί με τον mysqld server.
3. Με δημιουργία αποθηκευμένων διαδικασιών ή συναρτήσεων (stored procedures and functions), οι οποίες γράφονται σε SQL.

Αναλύουμε ακολούθως τους παραπάνω τρόπους:

3.3.1 Προσθήκη μιας udf μέσω της διεπαφής της mysql

Η mysql παρέχει το εξής statement προκειμένου να εισάγει ένας χρήστης μία udf:

```
CREATE [AGGREGATE] FUNCTION function_name
  RETURNS {STRING|INTEGER|REAL|DECIMAL}
  SONAME shared_library_name
```

Στο παραπάνω statement , function_name είναι το όνομα της udf με το οποίο θα την καλούμε στα SQL statements. Ο τύπος RETURNS καθορίζει την τιμή που η udf επιστρέφει, η οποία είναι αποκλειστικά INTEGER ή STRING ή REAL ή DECIMAL. Ο όρος *shared_lib-rary_name* αντιστοιχεί στο αντικειμενικό αρχείο (object file) το οποίο περιέχει τον compiled κώδικα της συνάρτησης και το οποίο έχει την επέκταση (.so).
Ο κώδικας αυτός μπορεί να είναι γραμμένος σε C ή C++, ή σε άλλη γλώσσα που χρησιμοποιεί τις κλήσεις της C. Προφανώς, προκειμένου να εισάγει κανείς μία udf θα πρέπει να έχει δικαιώματα INSERT

στη βάση, ενώ μετά την εισαγωγή της, αυτή είναι διαθέσιμη μόνον όταν χρησιμοποιείται η αντίστοιχη βάση. Εάν ο χρήστης επιθυμεί να διαγράψει την αντίστοιχη udf, δίνει την εντολή

```
DROP FUNCTION function_name
```

Μία udf κανονικά εκτελείται σε μία γραμμή ενός πίνακα κάθε φορά. Αν η συνάρτηση δηλωθεί ως aggregate function, τότε εκτελείται σε μία ομάδα σειρών και εμφανίζει συμπεριφορά όμοια με αυτή των γνωστών μας aggregate functions, όπως η sum, η count κτλ. Ωστόσο, κάθε πρόγραμμα C ή C++ το οποίο προορίζεται να ενσωματωθεί στη mysql ως udf, οφείλει να τηρεί τα παρακάτω:

- Αν θεωρήσουμε ότι xxx() είναι το όνομα της συνάρτησης, είναι απαραίτητη η ύπαρξη μίας συνάρτησης xxx στον κώδικα C ή C++, η οποία είναι και η main function.
- Επιθυμητή είναι η ύπαρξη μιας συνάρτησης xxx_init(). Η συνάρτηση αυτή ελέγχει τον αριθμό των ορισμάτων της xxx(), τον τύπο τους, και εκχωρεί όποια μνήμη ή δομές δεδομένων απαιτούνται από την udf. Επίσης, εδώ ορίζονται τα μηνύματα λαθών τα οποία επιστρέφουμε στο χρήστη.
- Αντίστοιχα, επιθυμητή είναι η ύπαρξη μιας συνάρτησης xxx_deinit() προκειμένου να ελευθερώσουμε όποιες δομές δεδομένων ή κομμάτια μνήμης εκχωρήθηκαν με τη συνάρτηση xxx_init().

Εάν πρόκειται για aggregate udf, είναι απαραίτητη η ύπαρξη των παρακάτω συναρτήσεων μέσα στον κώδικα της udf:

- xxx_add(): Η συνάρτηση αυτή καλείται σε κάθε νέα σειρά που εξετάζουμε σε ένα group και ουσιαστικά συνιστά το σώμα της κυρίως συνάρτησης.
- xxx_clear(): Η συνάρτηση αυτή καλείται κάθε φορά που εξετάζουμε ένα νέο group και αρχικοποιεί τις δομές και τους μετρητές που χρησιμοποιούνται από τη συνάρτηση.

Σε μία aggregate function δηλαδή, η mysql ακολουθεί τα παρακάτω βήματα κατά την εκτέλεση της συνάρτησης:

- Αρχικά καλείται η συνάρτηση xxx_init().
- Ακολούθως ταξινομείται ο πίνακας βάσει της έκφρασης GROUP BY.
- Καλείται η συνάρτηση xxx_clear() για την πρώτη σειρά σε κάθε νέο group.
- Καλείται η συνάρτηση xxx_add() για κάθε σειρά στο group.
- Καλείται η συνάρτηση xxx() προκειμένου να πάρουμε το αποτέλεσμα μόνον όταν αλλάζει το group ή έχουμε τελειώσει με την τελευταία σειρά.

Όλες οι συναρτήσεις πρέπει να είναι thread-safe. Προκειμένου να υπάρχει συμβατότητα με εκδόσεις mysql server πριν την έκδοση 5.5, είναι απαραίτητο να χρησιμοποιήσουμε και μία συνάρτηση xxx_reset() η οποία επιτελεί το ρόλο της clear(). Μία καλή πρακτική για προς τα πίσω συμβατότητα είναι να καλούμε τη reset μέσα από τη clear() (και αυτό να είναι το μόνο στοιχείο στο σώμα της clear()).

Η δήλωση μιας συνάρτησης udf διαφέρει ανάλογα με τον τύπο που η συνάρτηση επιστρέφει:

- Για συναρτήσεις xxx() που επιστρέφουν string:
  char *xxx(UDF_INIT *initid, UDF_ARGS *args,char *result, unsigned long *length, char *is_null, char *error);
  Στην παραπάνω δήλωση, initid είναι ο δείκτης στις δομές που αρχικοποιήθηκαν μέσω της init function, args είναι ο δείκτης των ορισμάτων, length είναι το μέγιστο μήκος που επιστρέφουμε το οποίο ορίζεται στην xxx_init(), is_null είναι η ειδική παράμετρος για επιστροφή κενών τιμών και διαχείριση λαθών ενώ error είναι ο δείκτης που κρατά τα μηνύματα λαθών.
- Για συναρτήσεις xxx() που επιστρέφουν integer:
  long long xxx(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char *error);
- Για συναρτήσεις xxx() που επιστρέφουν real:
  double xxx(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char *error);
- Για συναρτήσεις xxx() που επιστρέφουν decimal, η δήλωση είναι ίδια με τις συναρτήσεις που επιστρέφουν strings.

Η δηλώσεις των επιμέρους συναρτήσεων που χρησιμοποιούνται στις udf έχουν ως εξής:

Για τις συναρτήσεις init() και deinit():

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

Η παράμετρος initid περνάει σε όλες τις συναρτήσεις και «δείχνει» σε μία δομή UDF_INIT, ενώ ελευθερώνεται (της ανατίθεται η τιμή NULL) στη συνάρτηση deinit().

Για τις συναρτήσεις clear() και reset():

```
void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,char *is_null, char *error);
```

Τέλος σημειώνουμε ότι, για αυτή την κατηγορία udf, κατά το compilation της συνάρτησης πρέπει να κάνουμε τροποποιήσεις ως προς τον compiler και τις βιβλιοθήκες που χρησιμοποιούμε ανάλογα με το λειτουργικό σύστημα στο οποίο τρέχει ο mysql server.

3.3.2 Προσθήκη μιας udf στον mysqld server

Μπορούμε, ακολουθώντας τις βασικές αρχές που περιγράψαμε στην προηγούμενη ενότητα να ενσωματώσουμε τις udf στον πηγαίο κώδικα του mysql server μαζί με τις υπόλοιπες built-in συναρτήσεις της mysql. Ωστόσο πρέπει να δώσουμε ιδιαίτερη έμφαση στους περιορισμούς που τίθεται από τις διαφορετικές εκδόσεις του server προκειμένου να υπάρχει προς τα πίσω συμβατότητα με τις προηγούμενες εκδόσεις της mysql αλλά και δυνατότητα compilation στα διαφορετικά λειτουργικά συστήματα. Προκειμένου να χρησιμοποιηθούν οι UDFs, είναι αναγκαίο να συνδέσουμε (link) το mysqld δυναμικά, μέσω της εντολής –rdynamic.

3.3.3 Δημιουργία αποθηκευμένων διαδικασιών και συναρτήσεων

Ο τρίτος και τελευταίος τρόπος δημιουργίας μιας udf συνάρτησης είναι μέσω αποθηκευμένων συναρτήσεων (functions) και διαδικασιών (procedures). Ουσιαστικά, δεν πρόκειται για udf με τον ορισμό που δώσαμε στις δύο προηγούμενες ενότητες, δηλαδή δεν γράφουμε πηγαίο κώδικα τον οποίο ακολούθως κάνουμε compile και τον ενσωματώνουμε στη mysql, αλλά γράφουμε απευθείας sql κώδικα, τον οποίο και **αποθηκεύουμε** με ένα συγκεκριμένο τρόπο. Ακολούθως, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση ή διαδικασία που γράψαμε καλώντας την σε ένα sql statement με το όνομά της. Επί της ουσίας δημιουργούμε μία συντόμευση για το κομμάτι sql κώδικα που κατασκευάσαμε, και κάθε φορά που καλούμε τη συνάρτηση εκτελείται ο κώδικας που γράψαμε αυτούσιος, όπως περίπου συμβαίνει και με την εντολή view της sql. Ο χρόνος εκτέλεσης είναι αρκετά μεγαλύτερος από αυτό των udf που είναι γραμμένες σε C ή C++, ωστόσο η υλοποίηση του κώδικα είναι πολύ ευκολότερη. Οι udf αυτού του τύπου χωρίζονται σε διαδικασίες (procedures) και συναρτήσεις (functions), που γενικότερα καλούνται ρουτίνες (routines).

3.3.3.1 Συναρτήσεις

Μία udf stored function μπορεί να διαβάζει ή να μη διαβάζει δεδομένα ενώ επιστρέφει ένα και μόνο ένα αποτέλεσμα το οποίο ανήκει σε κάποιους από τους γνωστούς τύπους της mysql (CHAR(),VARCHAR(), INT,REAL κοκ). Η δήλωσή της γίνεται, όπως και για τις κλασσικές udf με την εντολή CREATE FUNCTION, η οποία όμως για stored functions έχει ορισμένα επιπλέον χαρακτηριστικά (characteristics) όπως φαίνεται ακολούθως[20]:

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  FUNCTION sp_name ([func_parameter[,...]])
  RETURNS type
  [characteristic ...] routine_body

characteristic:
  COMMENT 'string'
 | LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
 | SQL SECURITY { DEFINER | INVOKER }
```

Παρατηρούμε ότι στην παραπάνω δήλωση δεν υπάρχει ο τύπος SONAME που συναντήσαμε στην ενότητα 3.3.1. Τα χαρακτηριστικά DEFINER και SQL SECURITY ορίζουν τα access levels που

[20] http://dev.mysql.com/doc/refman/5.0/en/create-procedure.html

αφορούν στην εκτέλεση της συνάρτησης, ενώ sp_name είναι το όνομα της συνάρτησης. Οι παράμετροι που περνάμε στην συνάρτηση στο τμήμα func_parameter θεωρούνται ότι είναι μόνον παράμετροι εισόδου, γεγονός το οποίο δεν ισχύει για τις διαδικασίες (procedures). Ακολούθως εξετάζουμε τα χαρακτηριστικά που μπορεί να διαθέτουν οι αποθηκευμένες ρουτίνες:

- Μία συνάρτηση είναι ντετερμινιστική (DETERMINISTIC) όταν για συγκεκριμένο input παράγει συγκεκριμένο output. Ειδικότερα αυτό συνεπάγεται ο κώδικας της συνάρτησης να μην περιέχει τυχαία χαρακτηριστικά, όπως είναι οι κλήσεις σε συναρτήσεις σαν την NOW() ή τη RAND() που παράγουν τυχαιότητα.
- Ο όρος CONTAINS SQL υποδεικνύει ότι η συνάρτηση δεν περιέχει δηλώσεις που διαβάζουν ή γράφουν δεδομένα (όπως η εντολή SET @x=1). Οι υπόλοιποι όροι της κατηγορίας υποδεικνύουν τις αντίστοιχες λειτουργίες (παραδείγματος χάριν, ο όρος NO SQL δηλώνει ότι η συνάρτηση δεν περιέχει sql statements κοκ.)
- Εάν δηλώσουμε κάποιο χρήστη στο χαρακτηριστικό DEFINER, θα πρέπει να αναφερόμαστε σε έναν υπάρχον λογαριασμό χρήστη της βάσης στη μορφή 'user'@'host' ή στον CURRENT_USER. Προφανώς ο χρήστης που θα εκτελέσει τη ρουτίνα θα πρέπει να έχει δικαιώματα EXECUTE επί αυτής, όπως επίσης και ο λογαριασμός που ορίζεται στο DEFINER.

3.3.3.2 Διαδικασίες

Αντίθετα με μία συνάρτηση, μία διαδικασία μπορεί να επιστρέφει περισσότερα του ενός αποτελέσματα, τα οποία δεν είναι αναγκαστικά τύποι της mysql, όπως πίνακες κτλ. Η βασική εντολή δημιουργίας μίας διαδικασίας είναι η εξής:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body
```

Τα χαρακτηριστικά που ορίζουμε για τη διαδικασία με το αναγνωριστικό (identifier) "characterstic" είναι τα ίδια με αυτά που αναφέραμε για συναρτήσεις στην προηγούμενη ενότητα. Ωστόσο ιδιαίτερη έμφαση πρέπει να δοθεί στο κομμάτι της δήλωσης που αφορά στα ορίσματα της διαδικασίας(*proc_parameter)*. Κάθε όρισμα (παράμετρος) χαρακτηρίζεται ως IN, OUT ή INOUT. IN είναι η προεπιλεγμένη κατάσταση και δηλώνει ότι με την παράμετρο αυτή η διαδικασία διαβάζει μία τιμή. OUT είναι η παράμετρος η οποία χρησιμοποιείται για output και INOUT η παράμετρος από την οποία διαβάζουμε δεδομένα αλλά και στην οποία μπορούμε να γράψουμε δεδομένα.
Τέλος σημειώνουμε ότι τόσο στις procedures όσο και στις functions, δεν μπορούμε να δώσουμε εντολή χρησιμοποίησης μιας συγκεκριμένης βάσης δεδομένων (USE database), αλλά προϋποτίθεται ότι κάνουμε χρήση μίας συγκεκριμένης βάσης.

3.3.4 Ngram udfs

Προκειμένου να διευκολύνουμε διάφορες εργασίες συναφείς με ngrams, κατασκευάσαμε μία σειρά από mysql udfs σαν προσθήκη των udfs του πακέτου flamingo. Ανήκουν στην $1^{η}$ (3.3.1) και $3^{η}$ (3.3.3) κατηγορία των mysql udfs, ενώ ο πηγαίος κώδικας είναι διαθέσιμος σε αυτό το σύνδεσμο. Είναι οι εξής:

3.3.4.1 C/C++ udfs:

- **grams**: integer grams(string s_1, string s_2, int n)
  Πρόκειται για μία συνάρτηση γραμμένη σε C η οποία επιστρέφει τον αριθμό των κοινών grams μεγέθους n ανάμεσα στις συμβολοσειρές (strings) s_1 και s_2. Προς τούτο, παράγουμε τα grams μεγέθους n για κάθε μία από τις συμβολοσειρές και απαριθμούμε τα κοινά.

- **gram_app**: integer gram_app(string gram, string s)

  Πρόκειται για μία συνάρτηση γραμμένη σε C++, η οποία επιστρέφει τον αριθμό των εμφανίσεων του gram "gram" στο string s.

- **gram_select**: string gram_select(string gram_column_name, string term, int gram_size)

  Πρόκειται για μία συνάρτηση γραμμένη σε C++, η οποία διευκολύνει τη δημιουργία udfs για inverted indexes. Εάν έχουμε δημιουργήσει έναν inverted index και το όνομα της στήλης που περιέχει τα grams είναι 'gr', ενώ ο όρος που αναζητούμε είναι 'bomb', με μέγεθος gram 3, καλούμε τη udf ως εξής:

  select gram_select('gr','bomb',3);

  και αυτή επιστρέφει

  'gr='bom' OR gr='omb' '.

  Δηλαδή σπάει τον όρο που αναζητούμε σε tokens μεγέθους gram_size και δημιουργεί μία ακολουθία από OR clauses προκειμένου αυτά να χρησιμοποιηθούν σε ένα where statement. Η udf αυτή δεν κάνει κάτι διαφορετικό από την συνάρτηση SUBSTRING της mysql, ωστόσο μία udf γραμμένη σε C δεν μπορεί να καλέσει τις συναρτήσεις αυτές καθώς βρίσκονται ένα επίπεδο κώδικα υψηλότερα. Ως εκ τούτου, ο κώδικας της gram_select δεν έχει ιδιαίτερη χρησιμότητα ως απομονωμένη οντότητα, μπορεί όμως να ενσωματωθεί σε κάποια άλλη udf που θα χρειάζεται παρόμοιες λειτουργίες επί συμβολοσειρών στο χαμηλό επίπεδο των γλωσσών C ή C++.

Προκειμένου να χρησιμοποιήσουμε τις παραπάνω συναρτήσεις, είναι απαραίτητο να προβούμε στις εξής ενέργειες:

- Κάνουμε compile[21] τον πηγαίο κώδικα χρησιμοποιώντας τις απαραίτητες βιβλιοθήκες, στο σύστημα στο οποίο είναι εγκατεστημένος ο mysql server.

[21] Για τις udf που αναφέραμε και για λειτουργικό σύστημα Mac OS X , οι απαραίτητοι compilers και βιβλιοθήκες είναι οι εξής αντίστοιχα για C και C++:

- Αντιγράφουμε το αντικειμενικό αρχείο στο φάκελο όπου είναι εγκατεστημένα τα πρόσθετα (plugins) της mysql (συνήθως /mysql/lib/plugin).
- Επανεκκινούμε το mysql server.
- Επιλέγουμε τη βάση στην οποία επιθυμούμε να χρησιμοποιήσουμε τη udf και δίνουμε την εντολή CREATE FUNCTION σύμφωνα με την ενότητα 3.3.1.[22]

3.3.4.2 Αποθηκευμένες Διαδικασίες/Συναρτήσεις:

- **get_score**: procedure get_score(string s, string table, int gram_size, int limit)

  Η διαδικασία αυτή υποθέτει ότι υπάρχει ήδη ένας inverted index με όνομα table και grams συγκεκριμένου gram_size. Βάσει του limit, επιστρέφει τα n καλύτερα αποτελέσματα της αναζήτησης του όρου term στον index, χρησιμοποιώντας ως μετρική τον αριθμό των κοινών grams. Πχ αν καλέσουμε την διαδικασία ως εξής:

  call get_score('hihi','laugh',3,5)

  Θα μας επιστρέψει τις 5 συμβολοσειρές του πίνακα 'laugh' που έχουν τα περισσότερα κοινά grams με τη συμβολοσειρά 'hihi'. Οι συμβολοσειρές αυτές είναι οι ίδιες με εκείνες με τις οποίες έχει γίνει η συσχέτιση βάσει του inverted index, όπως περιγράψαμε στην ενότητα 2.1.2.2.1.

  Ο τρόπος λειτουργίας της udf προσδιορίζεται από τα εξής:

  1. Προϋπόθεση για την χρήση της είναι, όπως αναφέρθηκε, η ύπαρξη ενός πίνακα, ο οποίος θα λειτουργεί ως inverted index και θα έχει τη δομή [<gram>,<term>,<origin>]. Gram είναι το gram, term ο όρος από τον οποίο προέρχεται και origin ο πίνακας στον οποίο ανήκει ο όρος term.
  2. Ακολούθως επιλέγουμε από τον πίνακα table όλους τους όρους term που έχουν κοινά grams με τη συμβολοσειρά s. Αυτό γίνεται παράγοντας όλα τα πιθανά grams μεγέθους gram_size της s και εκτελώντας μία εντολή sql που στο τμήμα WHERE περιέχει clauses της μορφής gram='gram1' OR gram='gram2' κοκ.
  3. Ομαδοποιούμε τα δεδομένα μας βάσει του πεδίου term, και τα κατηγοριοποιούμε βάσει του count(origin). Αυτό είναι και το κρισιμότερο σημείο της υλοποίησης. Ομαδοποιώντας (με την εντολή GROUP BY) τα δεδομένα μας, εξασφαλίζουμε ότι θα μετρήσουμε πόσα είναι τα κοινά grams της συμβολοσειράς που αναζητούμε με κάθε term του πίνακα.

     Αυτό διότι, εάν ένας όρος έχει περισσότερα του ενός κοινά grams με τη συμβολοσειρά που αναζητούμε, έστωσαν 2 αυτά, τότε η εντολή group by θα δώσει 2 εγγραφές στην αντίστοιχη ομάδα, μία για κάθε gram. Κανονικά[23], η μέτρηση count(term) μετά την εντολή

---

gcc -Wall -O3 -I/usr/local/mysql/include/ -shared –o grams.so grams.c
g++ -Wall -bundle -bundle_loader /usr/local/mysql/bin/mysqld -o gram_app.so
`/usr/local/mysql/bin/mysql_config --cflags` gram_app.cc

[22] Είναι απαραίτητο σε περίπτωση που η συνάρτηση υπάρχει ήδη να δώσουμε προηγουμένως την εντολή DROP FUNCTION IF EXISTS, προκειμένου να μην υπάρχουν δύο συναρτήσεις ή διαδικασίες με το ίδιο όνομα.

[23] Σε μία απλοποιημένη περίπτωση, όπου τα δεδομένα μας προέρχονται μόνον από ένα πίνακα

ομαδοποίησης, θα μας έδινε τον αριθμό των κοινών grams, αφού για κάθε κοινό gram, το αντίστοιχο term θα εμφανίζεται μία φορά στην ομάδα που του αντιστοιχεί. Ωστόσο, επειδή τα δεδομένα μας μπορεί να προέρχονται από διαφορετικούς πίνακες χρησιμοποιούμε το μέτρο COUNT(origin) προκειμένου να τα ταξινομήσουμε. Ο λόγος για τον οποίο δεν μετράμε βάσει του πεδίου term αλλά του πεδίου origin είναι ότι ενδεχομένως κάποιο term να προέρχεται από διαφορετικούς πίνακες και λόγω διπλοτύπων (duplicates) να ευνοείται έναντι άλλου term, το οποίο κατά τα άλλα είναι συναφέστερο με την συμβολοσειρά που αναζητούμε.
Προκειμένου να καταστούν σαφέστερα τα παραπάνω, έστω ότι αναζητούμε την συμβολοσειρά 'miR13' σε ένα πίνακα, του οποίου ο αντίστοιχος inverted index για grams μεγέθους 2 είναι ο εξής:

gram	term	origin
mi	miR12	A
iR	miR12	A
R1	miR12	A
12	miR12	A
mi	miB12	B
iB	miB12	B
B1	miB12	B
12	miB12	B
mi	miB12	A
iB	miB12	A
B1	miB12	A
12	miB12	A

Μόλις επιλέξουμε τις εγγραφές εκείνες με τις οποίες η συμβολοσειρά έχει κοινά grams θα λάβουμε:

gram	term	origin
mi	miR12	A
iR	miR12	A
R1	miR12	A
mi	miBKR1	B
R1	miBKR1	B
mi	miBKR1	A
R1	miBKR1	A

Εάν χρησιμοποιούσαμε την εντολή count(term) αντί για count(origin), καθίσταται εναργές βάσει του παραδείγματος ότι η συμβολοσειρά 'miBKR1' θα θεωρούνταν συναφέστερη με τη 'miR13' συγκριτικά με τη 'miR12', κάτι το οποίο προφανώς δεν ισχύει (με όρους string matching).

4. **get_score2**: procedure get_score2(string term,string table,int gram_size, string w, int limit)

   Υλοποιεί ό,τι και η προηγούμενη procedure, ωστόσο εφαρμόζεται σε inverted indexes που χρησιμοποιούν κάποιο βάρος για τα grams (πχ tfxidf). Το όνομα της στήλης που περιέχει το βάρος αυτό το εισάγουμε σαν όρισμα με το string w. Ως εκ τούτου, δεν χρησιμοποιούμε την εντολή GROUP BY term ORDER BY count(origin) αλλά αντικαθιστούμε την aggregate function με την count(w). Με τον τρόπο αυτό ενσωματώνουμε στη μηχανή αναζήτησης τις τεχνικές IR που υποδεικνύουν πόσο σημαντικά είναι ορισμένα δεδομένα έναντι άλλων. Έτσι ευνοούνται τα δεδομένα που έχουν μεγαλύτερο βάρος και όχι αυτά που εμφανίζονται συχνότερα.

5. **gram_score**: procedure gram_score(string gram, string term, int output)

   Η διαδικασία αυτή επιστρέφει τον αριθμό των εμφανίσεων του gram "gram" μέσα στο term "term".Ουσιαστικά υλοποιεί ό,τι και η gram_app (προηγούμενη ενότητα) αλλά με stored procedure.
   Την καλούμε ως εξής:

   ```
   set @s=0;
   call gram_score(<gram>,<term>,@s);
   select @s;
   ```

6. **fgram_score**: function fgram_score(string gram, string term) returns int

   Υλοποιεί την ίδια λειτουργία με την gram_score, την καλούμε όμως απλούστερα, καθώς πρόκειται για συνάρτηση:

   ```
   select fgram_score(<gram>,<term>);
   ```

7. **suggest**: procedure suggest(string term,string table,int gram_size, string w, int limit)

   Η procedure αυτή υλοποιεί ό,τι και η get_score2,μόνο που αποθηκεύει τα αποτελέσματα της αναζήτησης σε ένα πίνακα με το όνομα suggestions. Σε μελλοντικές κλήσεις της συνάρτησης ο πίνακας ενημερώνεται και προστίθενται οι νέες προτάσεις χωρίς να διαγράφονται οι παλαιότερες.

Προκειμένου να χρησιμοποιήσουμε τις παραπάνω συναρτήσεις και διαδικασίες, απλά εκτελούμε την εντολή cat udf.sql | mysql -u root -p <password> σε ένα τερματικό (υποθέτουμε ότι έχουμε mysql server se unix-based λειτουργικό, αλλιώς προσαρμόζουμε την εντολή για συστήματα σε περιβάλλον windows). udf.sql είναι το αντίστοιχο script που περιέχει τον κώδικα για κάθε μία από τις παραπάνω ρουτίνες. Τα scripts αυτά είναι διαθέσιμα στον παρακάτω σύνδεσμο[24].

[24] http://web.imis.athena-innovation.gr/redmine/projects/recom/wiki/UDF_for_n-grams

3.4 Υλοποίηση του μηχανισμού αναζήτησης με ngrams

Προκειμένου να υλοποιήσουμε τη μηχανή αναζήτησης με ngrams, αρχικά ήταν αναγκαίο να δημιουργηθεί ένας inverted index με όλα τα grams των strings που υπήρχαν στη βάση δεδομένων μας. Προς τούτο γράψαμε ένα script σε Perl, ο κώδικας του οποίου είναι διαθέσιμος στον παρακάτω σύνδεσμο. Εάν ο διαχειριστής του συστήματος Diana micro-T, μετά την προσθήκη ορισμένων νέων records επιθυμεί να ενημερώσει και τη μηχανή αναζήτησης, απλά καλεί το εκτελέσιμο το οποίο κατασκευάσαμε με την εντολή perl grams.pl, και ακολούθως, επιλέγει το μέγεθος του gram που επιθυμεί.

Με το perl script που κατασκευάσαμε, κάνουμε τις εξής παραδοχές:

- Το μέγεθος των grams κυμαίνεται από 2 έως και 10, ωστόσο ο χρήστης έχει τη δυνατότητα να δημιουργήσει πίνακες με διαφορετικά μεγέθη. Κάθε πίνακας διαφορετικού μεγέθους θα έχει και διαφορετικό όνομα, οπότε κανείς μπορεί να επιλέξει ως μέγεθος gram το 2 και να δημιουργηθεί ο πίνακας 2grams, και ακολούθως να επιλέξει ως μέγεθος το 4, δημιουργώντας τον πίνακα 4grams, χωρίς να διαγραφεί ο πίνακας 2grams. Δημιουργία εκ νέου πίνακα που προϋπάρχει διαγράφει τον παλαιότερο. Τέλος, δίδεται η δυνατότητα στο χρήστη να δώσει εντολή 'a', όταν ερωτάται για το μέγεθος των grams, και να δημιουργήσει όλους τους πίνακες για μεγέθη από 2 έως και 10.
- Διατηρούμε για κάθε gram τρείς διαφορετικές στήλες στον αντίστοιχο πίνακα, για τον υπολογισμό του tfxidf weight με τρείς διαφορετικούς τρόπους, προκειμένου ο διαχειριστής να μπορεί να έχει μεγαλύτερη ευελιξία επιλογών για τις παραμέτρους της αναζήτησης.

Ακολούθως παραθέτουμε ενδεικτικά τα πεδία των πινάκων που δημιουργεί το script grams.pl:

Field	Type	Null	Default
gram	varchar(2)	No	
term	varchar(100)	No	
origin	varchar(100)	No	
repetition	int(11)	Yes	1
dummyweight	int(11)	No	1
weight1	double	No	1
weight2	double	No	1

- Το πεδίο gram αναφέρεται στο gram αυτό καθεαυτό.
- Το πεδίο term στον όρο από τον οποίο προέρχεται το gram.
- Το πεδίο origin αναφέρεται στον πίνακα από τον οποίο προέρχεται ο term,καθώς πολλά terms είναι κοινά φερ ειπείν στα microtv4_genes και microtv4_mirnas.
- Το πεδίο repetition αναφέρεται στον αριθμό των εμφανίσεων του gram.

- Το πεδίο dummyweight ουσιαστικά δεν αποτιμά κάποιο βάρος για τα grams αλλά είναι πάντοτε 1. Κατά βάση εξυπηρετεί σκοπούς μελλοντικής επέκτασης, ώστε ο χρήστης να χρησιμοποιήσει κάποιο δικό του μέτρο.
- Το weight1 υπολογίζεται βάσει της σχέσης (8) της ενότητας 2.2.4.
- Το weight2 υπολογίζεται βάσει της σχέσης (9) της ενότητας 2.2.4.

Έχοντας υλοποιήσει τον inverted index, η επόμενη απαραίτητη κίνηση είναι να τροποποιήσουμε την συνάρτηση determine της κλάσης Microtv4Search, προκειμένου η αναζήτηση να γίνεται αξιοποιώντας τα ngrams. Συγκεκριμένα, για κάθε όρο που εισάγει ο χρήστης, θα αναζητούμε με ποιά terms στη βάση μας ο όρος αυτός έχει τα περισσότερα κοινά grams, αξιοποιώντας τον inverted index. Αυτό πραγματοποιείται με τη διαδικασία get_score την οποία περιγράψαμε στην ενότητα 3.3.4.2. Ο κώδικας της κλάσης Microtv4Search τροποποιείται ως εξής:

```
$sql_q = "call
get_score(".$cur_tok."'2grams',2,'weight1'".Yii::app()>params['microtv4_num_of_suggs'].")";

$sql_r = Yii::app()->getDb()->getCommandBuilder()->createSqlCommand($sql_q)->query();
foreach( $sql_r as $cur_res) {
      $temp =  array( array(  "term"=>$cur_res['term'],
                                          "entity"=>$cur_res['entity'],
                                          "type"=>"None",
                                          "id"=>"Id",
                                          "score"=>$cur_res['score']));
foreach( $top_matches as $key=>$cur_match) {
      //insert
                  array_splice( $top_matches, $key, 0, $temp);
      //pop the last
                  array_pop($top_matches);
                  break;
}
```

Στα παραπάνω, $cur_tok είναι ο όρος που αναζητούμε, 'weight1' το βάρος των gram, ενώ '2grams' ο inverted index που περιέχει τα 2-grams για τη βάση μας. Θα μπορούσαμε να καλέσουμε οποιοδήποτε άλλο πίνακα επιθυμούμε, ενώ εναλλακτικά, μπορούμε να χρησιμοποιήσουμε κάποια άλλη μετρική για τα βάρη των grams.Παρατηρούμε σε πρώτο επίπεδο ότι το μέγεθος του κώδικα μειώθηκε αισθητά (από λίγες σελίδες σε λίγες γραμμές), ενώ μεγάλη ήταν και η βελτίωση που παρατηρήθηκε στο μέσο χρόνο εκτέλεσης των ερωτημάτων. Συγκεκριμένα, σε σχέση με την edit distance, ο μέσος χρόνος εκτέλεσης για ένα ερώτημα μειώθηκε από **5 έως και 10 φορές**, όπως θα εξετάσουμε και στις επόμενες ενότητες.
Ως προς την τελική υλοποίηση, όπως θα παρουσιάσουμε και στην ενότητα 5, επιλέξαμε να χρησιμοποιήσουμε μέγεθος gram ίσο με 3, καθώς έτσι πετυχαίνουμε να μειώσουμε αισθητά το χρόνο εκτέλεσης, διατηρώντας υψηλά ποσοστά ακρίβειας σε σχέση με τα αντίστοιχα αποτελέσματα που παρουσιάζει η edit distance. Επιπλέον, θα χρησιμοποιήσουμε ευρετήριο grams για τους πίνακες εκείνους που χαρακτηρίζονται από μεγαλύτερα μήκη συμβολοσειρών, και την μέθοδο της συντακτικής απόστασης για τους πίνακες των οποίων οι εγγραφές έχουν μικρότερα μήκη. Ο λόγος για τον οποίο συμβαίνει αυτό είναι ότι οι τεχνικές ευρετηρίων, για διάφορους παράγοντες οι οποίοι θα αναλυθούν εκτενέστερα στην ενότητα 5, αποδίδουν χειρότερα σε σύνολα δεδομένων που χαρακτηρίζονται από μικρά μήκη.

3.5 Παρουσίαση των αποτελεσμάτων σε πραγματικό χρόνο

Προκειμένου να είναι περισσότερο εύχρηστη η εφαρμογή DIANA-microT, ήταν ένα επιθυμητό να προσαρμόσουμε τη μηχανή αναζήτησης ούτως ώστε να παρέχονται προτάσεις στον χρήστη σε πραγματικό χρόνο, καθώς αυτός πληκτρολογεί κάποιο όρο. Δηλαδή, αν ο χρήστης έχει πληκτρολογήσει σε δεδομένη χρονική στιγμή τη συμβολοσειρά "mi", η μηχανή αναζήτησης θα πρέπει να του παράσχει προτάσεις για microRNAs ή genes που σχετίζονται με τη συμβολοσειρά "mi", χωρίς ο χρήστης να έχει πατήσει επιστροφή (Enter). Εάν ο χρήστης ακολούθως πληκτρολογήσει και άλλους χαρακτήρες, και η συμβολοσειρά είναι πλέον "mirna", οι προτάσεις της μηχανής αναζήτησης θα πρέπει να προσαρμοστούν αντιστοίχως.
Κάτι τέτοιο απαιτεί μία ασύγχρονη επικοινωνία της ιστοσελίδας με τον server, δηλαδή αλλαγή μέρους της ιστοσελίδας χωρίς να πραγματοποιηθεί ανανέωσή της. Μέχρι το 1995 όπου και εμφανίστηκαν τα πρώτα παραδείγματα ασύγχρονης επικοινωνίας, κάτι τέτοιο ήταν αδύνατο, καθώς κάθε επικοινωνία της ιστοσελίδας με το server απαιτούσε ο server να διαβάσει ολόκληρο το περιεχόμενο της σελίδας, και συνεπώς, στην περίπτωση της μηχανής αναζήτησης, οι χαρακτήρες που ο χρήστης θα είχε πληκτρολογήσει θα διαγράφονταν με την ανανέωση αυτή. Ορισμένες από τις τεχνικές που συντέλεσαν στο να ξεπεραστούν τέτοια εμπόδια, περιγράφονται ακολούθως:

3.5.1 Asynchronous JavaScript and XML (AJAX)

Με τη συντομογραφία AJAX αναφερόμαστε σε ένα σύνολο μεθόδων που χρησιμοποιούνται σε τεχνολογίες ιστού και δρουν στην πλευρά του πελάτη (client) προκειμένου να δημιουργηθούν διαδραστικές εφαρμογές ιστού. Όπως υποδηλώνει και ο όρος, πρόκειται για μία σύζευξη μεταξύ των τεχνολογιών Javascript και XML. Με τη χρήση της τεχνολογίας αυτής είναι δυνατό να στείλουμε και να λάβουμε δεδομένα από ένα server ασύχρονα, δηλαδή χωρίς να έχουμε ανανέωση όλου του περιεχομένου μιας ιστοσελίδας.
Τα δεδομένα συνήθως ανευρίσκονται χρησιμοποιώντας το αντικείμενο XMLHttpRequest. Ωστόσο είναι δυνατό να παρακάμψουμε τη χρήση της XML, χρησιμοποιώντας άλλες τεχνολογίες, όπως φερ ειπείν την τεχνολογία JSON. Επίσης δεν είναι αναγκαίο οι αιτήσεις (requests) να είναι ασύγχρονες. Προκειμένου να χειριστεί την εμφάνιση των δεδομένων, η τεχνολογία AJAX αξιοποιεί ένα συνδυασμό από HTML και CSS. Το DOM προσπελάζεται με χρήση Javascript προκειμένου να πετύχουμε δυναμική παρουσίαση των αποτελεσμάτων. Στις περισσότερες περιπτώσεις, οι διάφορες τεχνολογίες συνδυάζονται ως εξής:

- HTML ή XHTML και CSS χρησιμοποιούνται για την παρουσίαση των δεδομένων.
- Το μοντέλο DOM (Document Object Model) χρησιμοποιείται για δυναμική αναπαράσταση και αλληλεπίδραση με τα δεδομένα.
- Η τεχνολογία XML χρησιμοποιείται για την ανταλλαγή δεδομένων και η XSLT για τη διαχείρισή τους.
- Το αντικείμενο XMLHttpRequest χρησιμοποιείται για την ασύγχρονη επικοινωνία
- Η Javascript χρησιμοποιείται για την ενοποίηση των παραπάνω τεχνολογιών.

Η τεχνολογία AJAX έχει, εκτός από τα πλεονεκτήματά της, και ορισμένα μειονεκτήματα, τα οποία οφείλονται κυριότερα σε δυσκολίες υλοποίησης. Παραθέτουμε μερικά παραδείγματα:

- Το πρώτο είναι η αδυναμία των περιηγητών ιστού να χειριστούν αποδοτικά τα στοιχεία που αφορούν στο Ιστορικό, δηλαδή το σημείο στο οποίο ο χρήστης αλληλεπίδρασε με τη σελίδα. Εξαιτίας της δυναμικής φύσης της ανταλλαγής δεδομένων, είναι δυνατό ο χρήστης να πατήσει το πλήκτρο επιστροφής στην προηγούμενη σελίδα την οποία επισκέφτηκε, αλλά να οδηγηθεί σε διαφορετικό σημείο.
- Οι αναρριχητές ιστού που πραγματοποιούν την δεικτοδότηση (indexing) στο Διαδίκτυο, δεν εκτελούν κώδικα javascript και συνεπώς απαιτείται εναλλακτικός τρόπος προκειμένου να επιτραπεί η δεικτοδότηση στην ιστοσελίδα.
- Όμοια, αν ο περιηγητής ενός χρήστη δεν υποστηρίζει Javascript ή XMLHttpRequests, ή έχει απενεργοποιήσει αυτή την ιδιότητα, η ιστοσελίδα δεν απεικονίζεται σωστά, εκτός και αν γίνουν ειδικές προσαρμογές σε επίπεδο υλοποίησης.
- Οι διεπαφές AJAX συνήθως αυξάνουν τον αριθμό των αιτήσεων που δημιουργούνται από το χρήστη και αποστέλλονται προς το server και τα υπόλοιπα συστήματα (τη βάση δεδομένων κτλ). Αυτό έχει ως επακόλουθο την αύξηση του χρόνου απόκρισης για μία δεδομένη εφαρμογή.
- Τέλος, τεχνολογίες ανάγνωσης οθόνης, όπως η JAWS, που χρησιμοποιούνται συχνά από άτομα με ειδικές ανάγκες, δεν είναι δυνατό να επεξεργαστούν ορισμένες εφαρμογές AJAX.

3.5.2 jQuery

Η jQuery είναι μία βιβλιοθήκη Javascript η οποία χρησιμοποιείται κατά κόρον σε εφαρμογές προκειμένου να διευκολύνει την εκτέλεση HTML στην πλευρά του πελάτη (client). Μέσω της σύνταξης του jQuery ,μπορούμε να επιλέξουμε στοιχεία DOM, να δημιουργήσουμε κινούμενα θέματα (animations) και να χειριστούμε γεγονότα (events), όπως φερ ειπείν το πάτημα το ποντικιού του χρήστη, το πάτημα ενός πλήκτρου ή το πάτημα κάποιας εικόνας στην ιστοσελίδα. Διευκολύνει ιδιαίτερα τη δημιουργία Ajax εφαρμογών, και παρέχει τη δυνατότητα να δημιουργηθούν πρόσθετα (plugins), με χρήση Javascript, στον υπάρχων κώδικα της ιστοσελίδας.

Ουσιαστικά ευνοείται η αφαίρεση σε επίπεδο κώδικα μεταξύ των διαφόρων στοιχείων μιας εφαρμογής και καθίσταται εφικτή η αλληλεπίδραση διαφορετικών τμημάτων τόσο σε χαμηλό επίπεδο εφαρμογής, όσο και σε υψηλότερο. Προκειμένου να αξιοποιηθεί η βιβλιοθήκη, απλά την ενσωματώνουμε στα

headers της ιστοσελίδας, όπως συμβαίνει με ένα συνηθισμένο κώδικα Javascript, με την παρακάτω δήλωση:

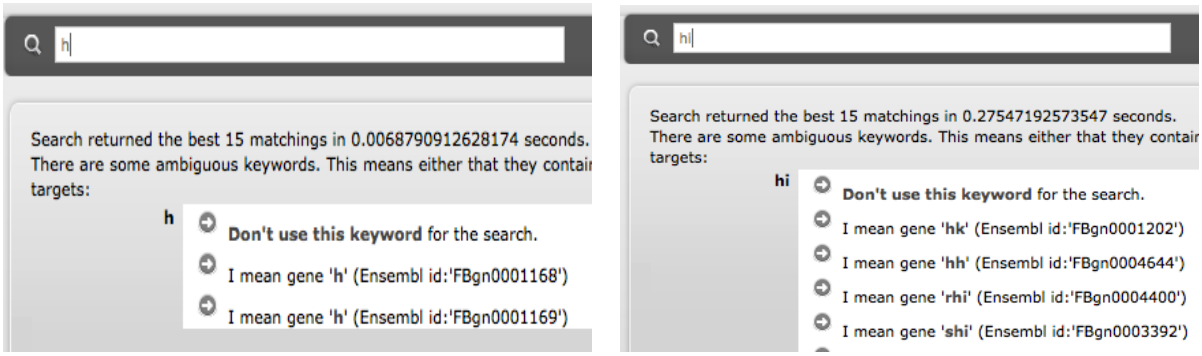
<script type="text/javascript" src="jquery.js"></script>

3.5.3 Τροποποίηση της εφαρμογής με χρήση των μεθόδων του yii framework

Το πακέτο yii παρέχει ειδικές μεθόδους για τον χειρισμό των εφαρμογών AJAX και τη χρήση JQuery. Τα βήματα που ακολουθούμε για την τροποποίηση της μηχανής αναζήτησης είναι τα εξής:

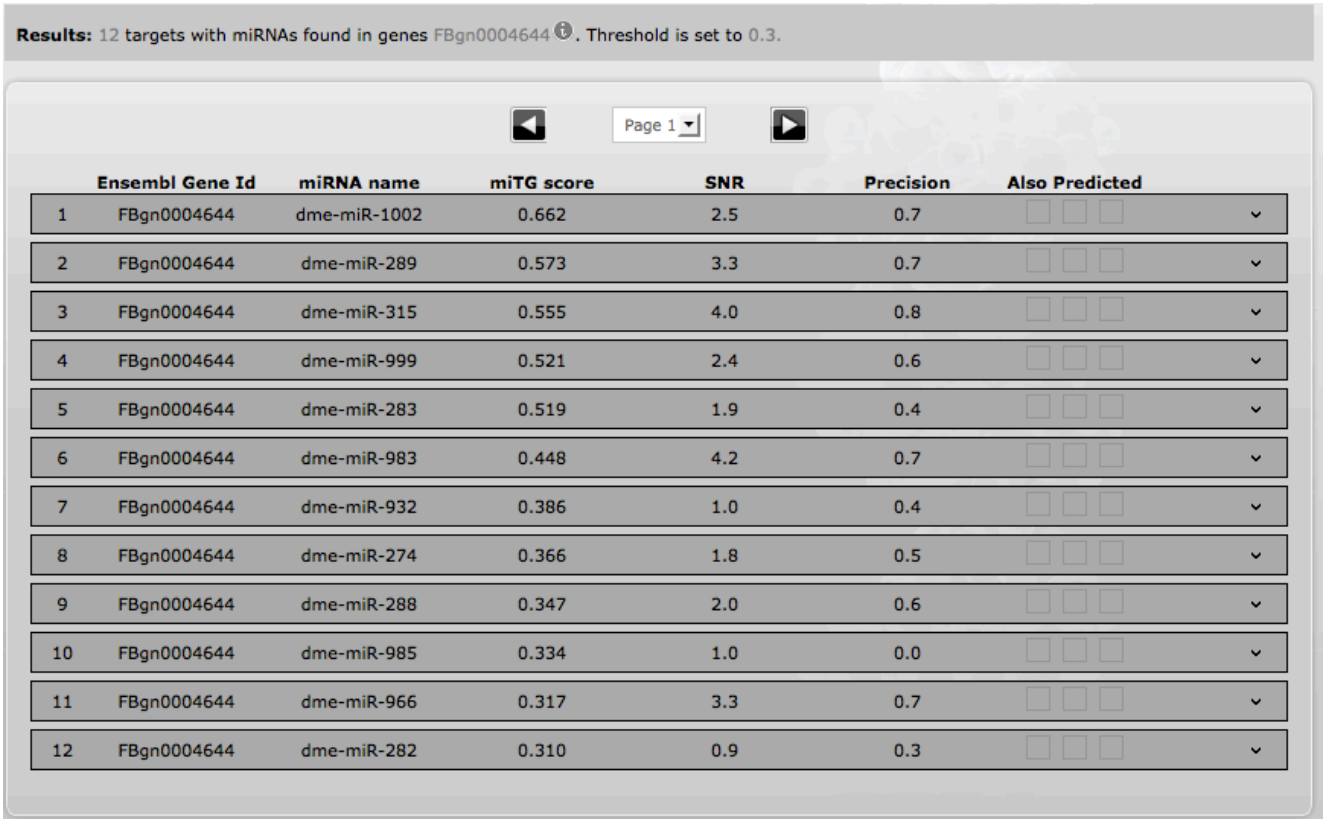
- Δημιουργία μιας δυναμικής φόρμας που αξιοποιεί τη συνάρτηση CHtml::ajax. Πρόκειται για μία συνάρτηση η οποία εγκαθιστά μία ajax κλήση με όλες τις παραμέτρους που υποστηρίζει η τεχνολογία ajax (POST, GET, onkeydown κτλ)
- Την κλήση αυτή χειριζόμαστε εντός του Microtv4Controller, με το action DisplayInfo το οποίο ορίζουμε στη φόρμα μέσω της ενσωματωμένης συνάρτησης του yii Yii::app()->createUrl("microtv4/displayInfo");
- Μέσα στο action DisplayInfo, καλούμε τη συνάρτηση Microtv4AjaxSearch, η οποία επιτελεί τις ίδιες λειτουργίες με την Microtv4Search αλλά χειρίζεται διαφορετικά τις μεταβλητές που δέχεται ως ορίσματα, προκειμένου να υποστηρίζει την δυναμική φύση της ajax κλήσης.
- Ακολούθως, μετασχηματίζουμε το αντίστοιχο view για να παράσχουμε στο χρήστη τα αποτελέσματα της αναζήτησης.

Ακολούθως παραθέτουμε την αντίστοιχη εικόνα για το νέο μηχανισμό προτάσεων:



Καθώς ο χρήστης πληκτρολογεί μέσα στη μπάρα αναζήτησης, οι προτάσεις αλλάζουν με δυναμικό τρόπο. Ακολούθως, επιλέγοντας κάποια πρόταση μπορεί να εξετάσει τις διάφορες επιλογές που παρέχονται από την εφαρμογή. Οι επιλογές αυτές εμφανίονται δυναμικά, όταν ο χρήστης επιλέξει το βέλος που βρίσκεται αριστερά από κάθε πρόταση, πάλι μέσω της τεχνολογίας ajax, ωστόσο με χρήση της συνάρτησης ajaxButton που προσφέρει το yii framework.

Παραθέτουμε τις επιλογές που παρέχονται εάν επιλέξουμε το gene 'hh':



Results: 12 targets with miRNAs found in genes FBgn0004644. Threshold is set to 0.3.

Page 1

	Ensembl Gene Id	miRNA name	miTG score	SNR	Precision	Also Predicted
1	FBgn0004644	dme-miR-1002	0.662	2.5	0.7	
2	FBgn0004644	dme-miR-289	0.573	3.3	0.7	
3	FBgn0004644	dme-miR-315	0.555	4.0	0.8	
4	FBgn0004644	dme-miR-999	0.521	2.4	0.6	
5	FBgn0004644	dme-miR-283	0.519	1.9	0.4	
6	FBgn0004644	dme-miR-983	0.448	4.2	0.7	
7	FBgn0004644	dme-miR-932	0.386	1.0	0.4	
8	FBgn0004644	dme-miR-274	0.366	1.8	0.5	
9	FBgn0004644	dme-miR-288	0.347	2.0	0.6	
10	FBgn0004644	dme-miR-985	0.334	1.0	0.0	
11	FBgn0004644	dme-miR-966	0.317	3.3	0.7	
12	FBgn0004644	dme-miR-282	0.310	0.9	0.3	

4

Αξιολόγηση

Στην ενότητα αυτή πραγματοποιούμε μία σειρά πειραμάτων προκειμένου:

1. να αξιολογήσουμε την *αποδοτικότητα εκτέλεσης* και την *ακρίβεια αποτελεσμάτων* των μεθόδων παροχής προτάσεων που χρησιμοποιούν ευρετήρια grams σε σχέση με τη μέθοδο που χρησιμοποιεί τη συντακτική απόσταση (edit distance).
2. να μελετήσουμε την επίδραση που έχει το μέγεθος των grams στην αποδοτικότητα και την ακρίβεια των μεθόδων, προκειμένου να καταλήξουμε στο μέγεθος εκείνο, που είναι προτιμότερο να χρησιμοποιηθεί για το σύστημά μας.

Επιπλέον, πραγματοποιούμε κάποια πειράματα τα οποία μας βοηθάνε να κατανοήσουμε κάτω από ποιές συνθήκες είναι προτιμότερο να χρησιμοποιηθεί η παροχή προτάσεων με βάση τη συντακτική απόσταση αντί για την παροχή προτάσεων με βάση τα grams.

Τα συμπεράσματα των πειραμάτων αυτής της ενότητας χρησιμοποιήθηκαν για την υιοθέτηση των ιδανικών σχεδιαστικών επιλογών κατά τη διάρκεια ανάπτυξης του νέου συστήματος παροχής προτάσεων των εφαρμογών DIANA.

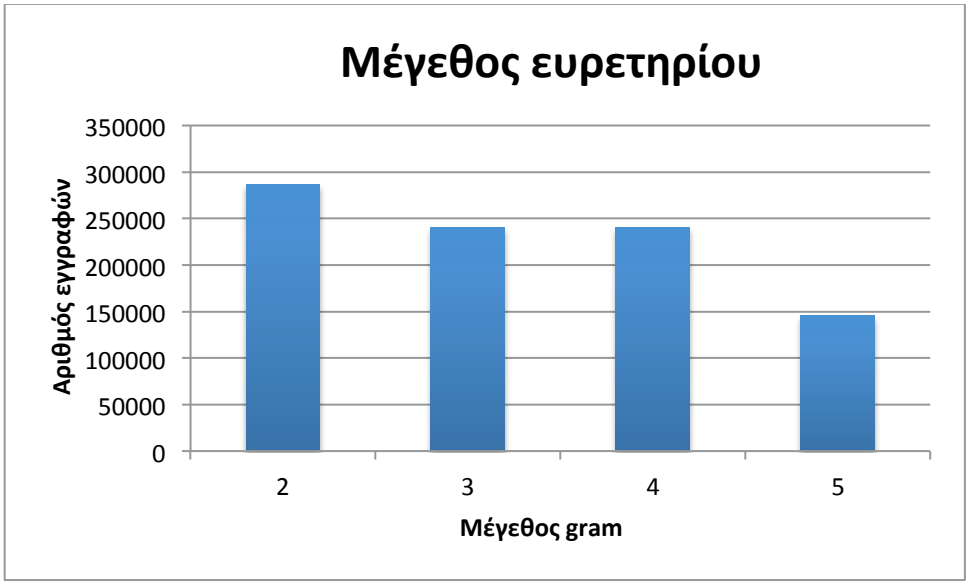
4.1 Περιγραφή των συνόλων δεδομένων (datasets)

Τα πειράματα που παρουσιάζονται σε αυτή την ενότητα πραγματοποιήθηκαν πάνω σε δύο είδη συνόλων δεδομένων: (α) σε ένα *πραγματικό σύνολο δεδομένων* (real dataset), το οποίο περιέχει εγγραφές από τη βάση δεδομένων των εφαρμογών DIANA και (β) σε ένα *συνθετικό σύνολο δεδομένων* (synthetic dataset), το οποίο κατασκευάστηκε με ψευδοτυχαίο τρόπο.

Το **πραγματικό σύνολο** δεδομένων κατασκευάστηκε από την ένωση των εγγραφών που περιέχονται στους πίνακες "microtv4_mirnas" (1884 εγγραφές), "microtv4_msynonyms" (494 εγγραφές), "microtv4_diseases"

(1003 εγγραφές) της εφαρμογής DIANA. Πρόκειται για τους ίδιους πίνακες που παρουσιάστηκαν στην ενότητα 3.1.Το αποτέλεσμα είναι ένα σύνολο 3381 όρων με ελάχιστο μήκος 6 χαρακτήρες και μέγιστο 29 χαρακτήρες. Πάνω σε αυτό το σύνολο όρων κατασκευάστηκαν ευρετήρια grams για μεγέθη grams από 2 έως 5 χαρακτήρες. Καθένα από τα ευρετήρια αυτά υλοποιήθηκε ως ένας πίνακας στη βάση δεδομένων, του οποίου κάθε εγγραφή αντιστοιχεί στο συνδυασμό ενός όρου με ένα gram το οποίο περιέχεται στον όρο αυτόν. Ο πίνακας αυτός έχει τα ίδια πεδία με τα πεδία του ευρετηρίου που παρουσιάστηκαν στην ενότητα 3.4. Το παρακάτω διάγραμμα συγκεντρώνει το πλήθος εγγραφών των συγκεκριμένων ευρετηρίων:

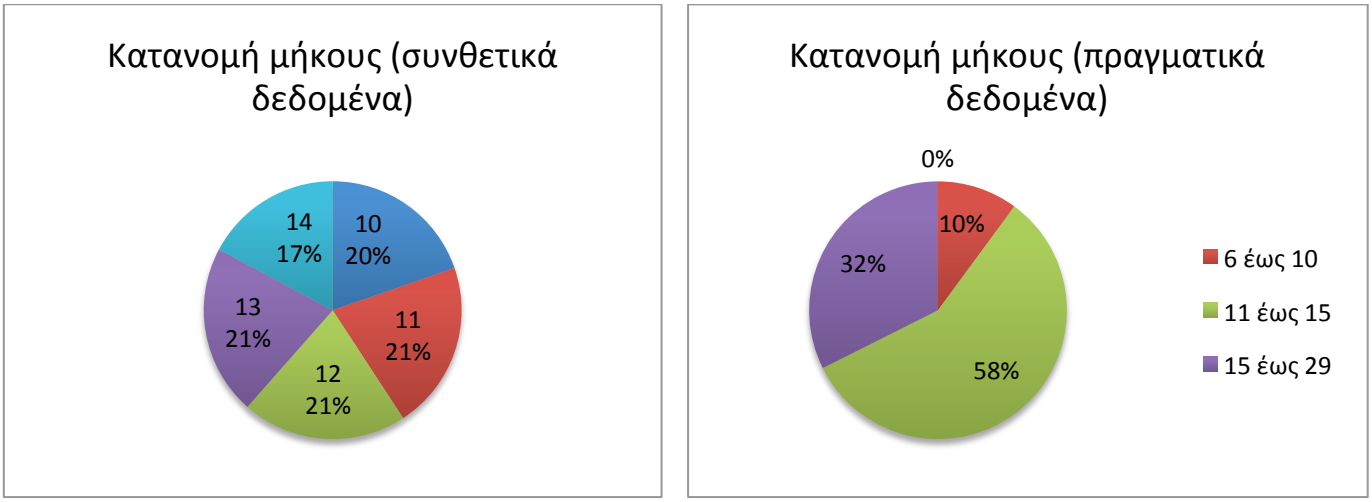
Γράφημα 4.1:



Το **συνθετικό σύνολο** δεδομένων κατασκευάστηκε ώστε να περιλαμβάνει τυχαίες ακολουθίες χαρακτήρων. Το συγκεκριμένο σύνολο περιέχει αρκετά υποσύνολα δεδομένων, καθένα από τα οποία χρησιμοποιείται για διαφορετικό πείραμα. Διάφορα μεγέθη που σχετίζονται με τις ακολουθίες των υποσυνόλων δεδομένων (π.χ., πλήθος ακολουθιών που περιέχονται, μήκος ακολουθιών κτλ), προσαρμόζονται ανάλογα με τις ανάγκες του κάθε πειράματος. Ανεξάρτητα από το πείραμα όμως, το αλφάβητο πάνω στο οποίο κατασκευάστηκαν όλες οι ακολουθίες περιλαμβάνει το σύνολο:S={[a,…,z],[A,…,Z],[0,..,9],[-]}.Η επιλογή αυτή δεν είναι τυχαία, καθώς βασίζεται στους χαρακτήρες που υπάρχουν στις εγγραφές της βάσης δεδομένων της εφαρμογής DIANA. Κατά την κατασκευή αυτών των ακολουθιών έγινε η θεώρηση ότι κάθε χαρακτήρας του αλφαβήτου έχει την ίδια πιθανότητα με τους υπόλοιπους να εμφανιστεί σε κάποιο σημείο μιας ακολουθίας και ότι η εμφάνιση αυτή είναι ανεξάρτητη από το ποιοί άλλοι χαρακτήρες παρουσιάζονται στην ίδια ακολουθία (uniform Bernoulli model). Πάνω σε καθένα από τα υποσύνολα συνθετικών δεδομένων, κατασκευάστηκαν ευρετήρια grams με παρόμοιο τρόπο όπως και για το πραγματικό σύνολο δεδομένων.

Ακολούθως παραθέτουμε τα αντίστοιχα κυκλικά γραφήματα που περιγράφουν την κατανομή του μήκους των χαρακτήρων για κάθε σύνολο δεδομένων αντίστοιχα:

Γράφημα 4.2:

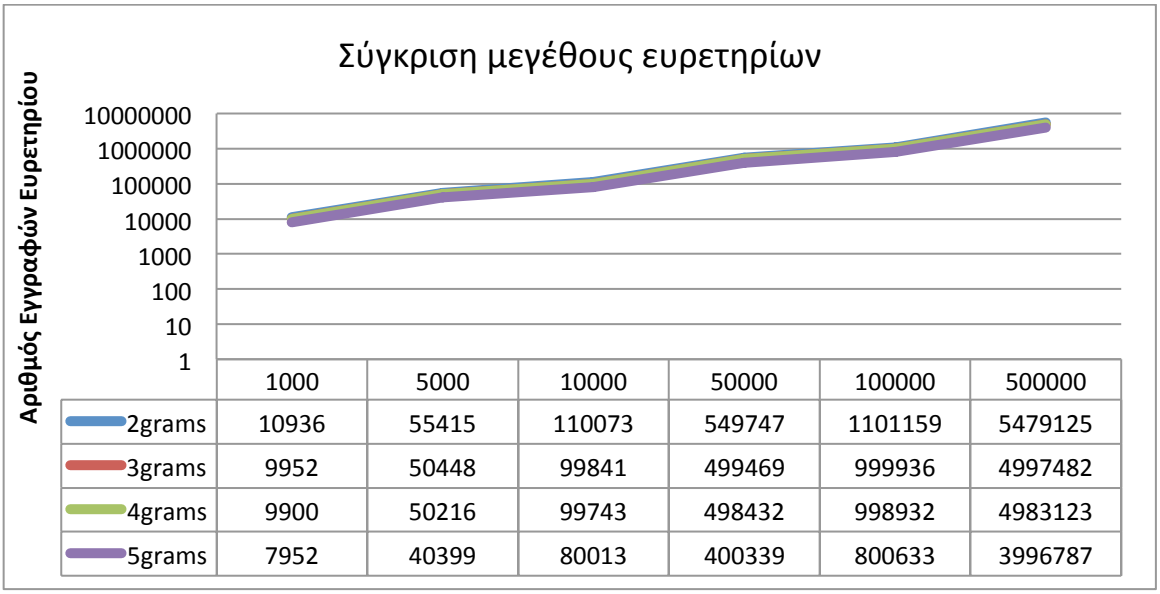


Παρατηρούμε ότι το μεγαλύτερο ποσοστό των πραγματικών δεδομένων που χρησιμοποιούμε στα πειράματά μας έχει μήκος 10 έως 15 χαρακτήρες και αυτός είναι ο κυριότερος λόγος για τον οποίο επιλέξαμε το αντίστοιχο μήκος για τα συνθετικά δεδομένα.
Στο γράφημα 4.3 συγκεντρώνεται το πλήθος εγγραφών (και επομένως το σύνολο των grams) των συγκεκριμένων ευρετηρίων για υποσύνολα διαφορετικού πλήθους ακολουθιών (από 1000 ως 100000 χαρακτήρες) στα οποία το μήκος της κάθε ακολουθίας κυμαίνεται μεταξύ 10 και 14 χαρακτήρων (το μήκος έχει επιλεγεί με τυχαίο τρόπο).
Συγκεκριμένα, κατασκευάζουμε τα εξής υποσύνολα δεδομένων (datasets):

dataset name	records
A	1000
B	5000
C	10000
D	50000
E	100000
F	500000

Επιλέξαμε να χρησιμοποιήσουμε αυτές τις ομάδες δεδομένων προκειμένου να εξετάσουμε αναλυτικά τη συμπεριφορά της εκάστοτε μεθόδου σε διαφορετικές συνθήκες εκτέλεσης. Ο διαφορετικός αριθμός εγγραφών συντελεί στην ανάδειξη των ισχυρών σημείων και των αδυναμιών που παρουσιάζονται κατά την απόδοση των διαφόρων αλγορίθμων, και μας δίνει μία καλύτερη εικόνα για το μέγεθος των πόρων που απαιτούνται από την εφαρμογή ιστού. Χαρακτηριστικό είναι το γεγονός ότι το μέγεθος του ευρετηρίου (inverted index) μεταβάλλεται σχεδόν γραμμικά όσο αυξάνονται τα δεδομένα, όπως παρουσιάζεται στο γράφημα που ακολουθεί:

Γράφημα 4.3:



	1000	5000	10000	50000	100000	500000
2grams	10936	55415	110073	549747	1101159	5479125
3grams	9952	50448	99841	499469	999936	4997482
4grams	9900	50216	99743	498432	998932	4983123
5grams	7952	40399	80013	400339	800633	3996787

Η αύξηση αυτή, συνεπάγεται ένα υπολογιστικό κόστος για τη δημιουργία και τη συντήρηση του ευρετηρίου σε ένα σύστημα που είναι ενεργό, και είναι μία παράμετρος η οποία πρέπει να αξιολογηθεί ως προς τα πλεονεκτήματα και τα μειονεκτήματά της. Στις ενότητες που ακολουθούν, αξιολογούμε την απόδοση των μεθόδων που αξιοποιούν ευρετήρια grams ως προς τα χρονικά αποτελέσματα και την ακρίβεια των προτάσεων. Μετράμε τη απόκριση στα ερωτήματα του χρήστη και συγκρίνουμε με τα αντίστοιχα που παράγουν οι μέθοδοι που αξιοποιούν τη συντακτική απόσταση. Οι μετρήσεις μας χωρίζονται σε δύο ενότητες. Η πρώτη αφορά στην αποδοτικότητα ως προς το χρόνο απόκρισης και η δεύτερη ως προς την συνάφεια των αποτελεσμάτων. Κάθε ενότητα περιλαμβάνει αντιστοίχως δύο υποενότητες, μία για πραγματικά και μία για συνθετικά δεδομένα

4.2 Μέτρηση της αποδοτικότητας των μεθόδων που χρησιμοποιούν grams

4.2.1 Πραγματικό σύνολο δεδομένων

Για τις μετρήσεις επί του πραγματικού συνόλου δεδομένων, χρησιμοποιούμε αυτούσια τη βάση δεδομένων της εφαρμογής ιστού DIANA, με βάση τα όσα αναλύθηκαν στην προηγούμενη ενότητα. Το πρώτο πείραμα το οποίο διεξάγουμε, αφορά στη μέτρηση του μέσου χρόνου εκτέλεσης ενός ερωτήματος στη μηχανή αναζήτησης, για υλοποίηση με χρήση συντακτικής απόστασης και ευρετηρίων grams. Προκειμένου να εκτελέσουμε το πείραμα αυτό, κατασκευάσαμε ένα πρόγραμμα σε Perl, μέσω του οποίου πραγματοποιούμε όλες τις ενέργειες που απαιτούν αλληλεπίδραση με τη βάση δεδομένων. Προκειμένου να εξασφαλίσουμε όσο πιο αμερόληπτες τιμές γίνεται, ακολουθήσαμε την εξής διαδικασία:

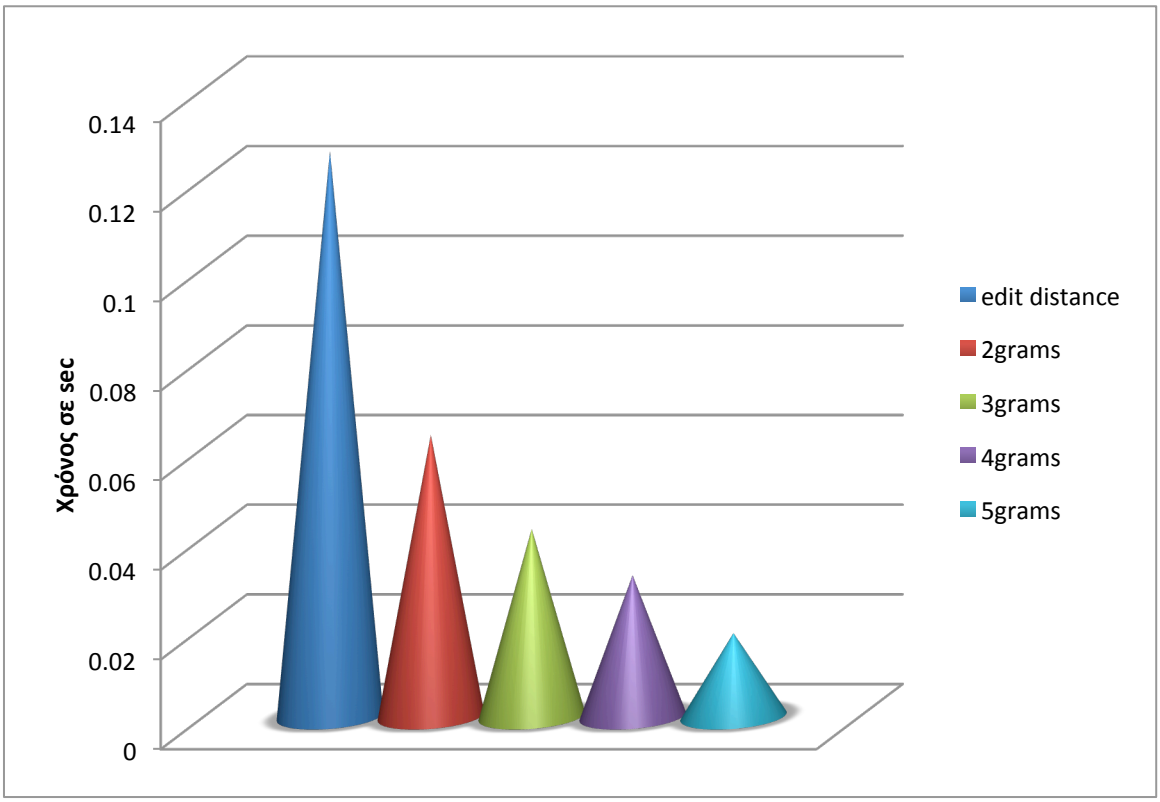
- Κατασκευάζουμε με τυχαίο τρόπο ένα τυχαίο σύνολο ερωτημάτων από τα πραγματικά δεδομένα. Προς τούτο επιλέγουμε με τυχαίο τρόπο ένα υποσύνολο των δεδομένων και ακολούθως τα τροποποιούμε αλλάζοντας τυχαία 1 έως 3 χαρακτήρες σε κάθε όρο.
- Κατασκευάζουμε ένα ερώτημα για κάθε όρο των πειραματικών μας δεδομένων.
- Για κάθε ερώτημα υπολογίζουμε το μέσο όρο εκτέλεσης στη βάση δεδομένων.
- Λαμβάνουμε το μέσο όρο των τιμών του πειράματος.
- Επαναλαμβάνουμε το ίδιο πείραμα με ίδια πειραματικά δεδομένα προκειμένου να αποκλείσουμε τις όποιες αστοχίες υλικού.
- Αλλάζουμε τα πειραματικά δεδομένα και επαναλαμβάνουμε τις μετρήσεις για να εξασφαλίσουμε την αντικειμενικότητα των αποτελεσμάτων μας, το ότι δηλαδή αυτά αντιπροσωπεύουν τη μέση περίπτωση χρονικής απόδοσης για το συγκεκριμένο υπολογιστικό σύστημα.

Ορισμένες επιπλέον παραδοχές που ακολουθούμε είναι οι εξής:

- Η μέτρηση του χρόνου γίνεται με τη χρήση της συνάρτησης gettimeofday(), του πακέτου Time::HiRes της perl. Μετράμε το χρόνο σε δευτερόλεπτα με ακρίβεια πολλών δεκαδικών ψηφίων χωρίς ωστόσο να μετράμε system clocks.
- Ανάμεσα στα ερωτήματα, εκτελούμε ένα sql select statement σε ένα μεγάλο πίνακα (με 700000 εγγραφές περίπου), προκειμένου να μην διατηρείται κάποια πληροφορία στη βάση δεδομένων λόγω caching από προηγούμενες αναζητήσεις, και τα αποτελέσματα των χρόνων να είναι όσο το δυνατόν πιο αξιόπιστα. Ο πίνακας στον οποίο εφαρμόζουμε το select statement περιέχει δεδομένα εντελώς διαφορετικά από τα πειραματικά.
- Συνολικά εκτελούμε κάθε πείραμα 5 φορές και υπολογίζουμε το μέσο όρο των αποτελεσμάτων.
- Τα πειράματα εκτελούνται σε server του ερευνητικού κέντρου ΙΠΣΥΠ προκειμένου να ελαχιστοποιήσουμε τυχόν αστοχίες υλικού και καθυστερήσεις λόγω φόρτου εργασιών στον υπολογιστή.
- Προκειμένου να έχουμε όσο το δυνατόν αποδοτικότερη αναζήτηση, δεικτοδοτήσαμε τις στήλες εκείνες των πινάκων οι οποίες εμπλέκονται σε αυτή, οργανώνοντάς τες σε B-δένδρα, ενώ επιστρέψαμε τα καλύτερα 10 αποτελέσματα από τα 100 και με τις δύο μεθόδους.

Οι χρόνοι αναζήτησης για τις αντίστοιχες μεθόδους είναι οι εξής:

Γράφημα 4.4:



Παρατηρούμε ότι τα αποτελέσματα της αναζήτησης που αξιοποιεί ευρετήριο είναι σαφώς ταχύτερα έναντι των αντιστοίχων μεθόδων που αξιοποιούν τη συντακτική απόσταση (edit distance). Αυτό είναι αναμενόμενο καθώς δεν είναι αναγκαίο να διατρέξουμε όλο τον πίνακα και να πραγματοποιήσουμε πράξεις υπολογισμού σε κάθε εγγραφή ξεχωριστά.

Γενικότερα, οι μηχανισμοί που αξιοποιούν ευρετήρια με grams, όπως θα σχολιάσουμε και στην επόμενη ενότητα, έχουν την τάση να αποδίδουν χειρότερα από τους αντιστοίχους που αξιοποιούν συντακτική απόσταση, σε σύνολα συμβολοσειρών μικρού μήκους. Αυτό οφείλεται στην πολύ μικρή επιλεκτικότητα που παρουσιάζει ο μηχανισμός αναζήτησης όσο αυξάνονται τα κοινά grams που έχουν οι όροι της αναζήτησης με τις εγγραφές του ευρετηρίου. Δηλαδή, κατά την εισαγωγή ενός ερωτήματος στο μηχανισμό αναζήτησης, όταν το σύνολο δεδομένων έχει τα χαρακτηριστικά που περιγράφησαν, επιστρέφονται πάρα πολλές εγγραφές, οι οποίες μάλιστα λαμβάνουν παραπλήσιες βαθμολογίες κατά την ταξινόμηση βάσει του βάρους των gram.

Έτσι ως απάντηση σε κάθε ερώτημα επιστρέφεται μεγάλο τμήμα της βάσης συγκριτικά με τα μεγέθη που επιστρέφουν ευρετήρια κατασκευασμένα για σύνολα συμβολοσειρών μεγαλύτερου μήκους. Αυτή η μεταφορά έχει συνέπειες στο χρόνο απόκρισης, και δεδομένου ότι οι πίνακες ευρετηρίων είναι σημαντικά μεγαλύτεροι από τους πίνακες των δεδομένων από τα οποία προέρχονται, για μικρά μήκη εγγραφών, ενδεχομένως η μέθοδοι ευρετηρίων να δώσουν αποτελέσματα χειρότερα από την edit distance, ειδικότερα για μικρά μεγέθη gram. Χαρακτηριστικό είναι άλλωστε ότι ο χρόνος εκτέλεσης μειώνεται όσο

αυξάνεται το μέγεθος του gram ,καθώς αυξάνεται η επιλεκτικότητα. Είναι πολύ μικρότερη η πιθανότητα ένας όρος να έχει πολλά κοινά 3-grams ή 4-grams με τα στοιχεία του ευρετηρίου από ό,τι να έχει κοινά 2-grams. Στα συνθετικά δεδομένα όπου τα δεδομένα παρήχθησαν με τυχαίο τρόπο, και τυχαία ήταν και η κατανομή των χαρακτήρων σε κάθε εγγραφή, αναμένουμε οι διαφορές στους χρόνους εκτέλεσης για διαφορετικά μεγέθη gram να είναι μικρότερες. Φυσικά οι διαφορές αυτές εντείνονται καθώς αυξάνεται το μέγεθος των δεδομένων, καθώς αυξάνονται πολλαπλάσια και τα μεγέθη των αντιστοίχων ευρετηρίων.

4.2.2 Συνθετικό σύνολο δεδομένων

Η διαδικασία που ακολουθούμε για την διεξαγωγή αυτού του πειράματος είναι η εξής:

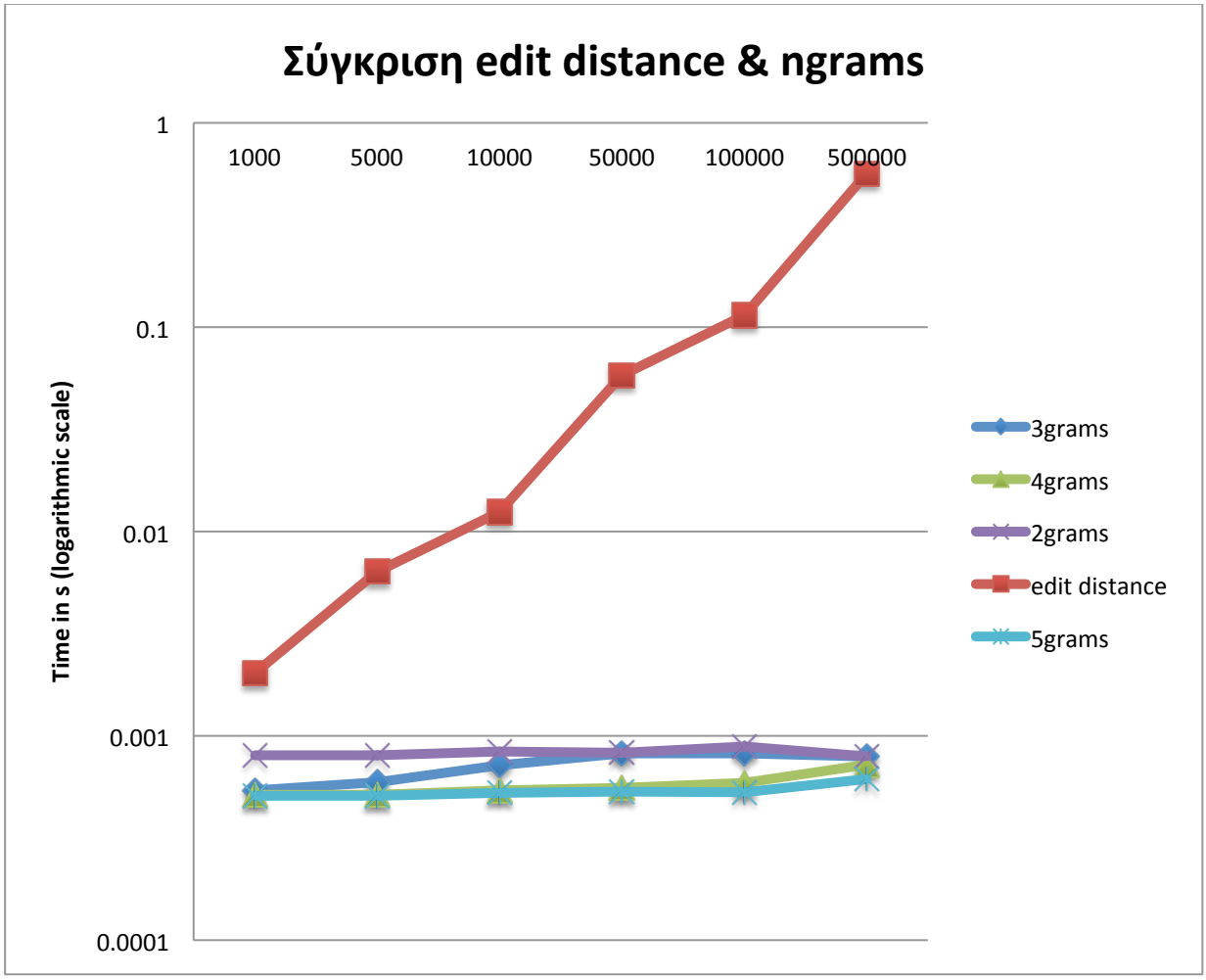
- Από κάθε σύνολο δεδομένων που δημιουργήσαμε επιλέγουμε με τυχαίο τρόπο 100 εγγραφές.
- Για κάθε εγγραφή από αυτές που επιλέχθηκαν, αλλάζουμε με τυχαίο τρόπο 1-5 χαρακτήρες της συμβολοσειράς, σε τυχαία θέση.
- Δημιουργούμε με τις τροποποιημένες εγγραφές ένα καινούργιο πίνακα ο οποίος αποτελεί τον πίνακα των πειραματικών μας τιμών (testcases).
- Για κάθε εγγραφή του πίνακα testcases υπολογίζουμε το χρόνο εκτέλεσης του ερωτήματος αναζήτησης της αντίστοιχης εγγραφής στη βάση, με χρήση edit distance και ngrams.
- Λαμβάνουμε τη μέση τιμή για τους χρόνους ολοκλήρωσης των 100 ερωτημάτων.

Προκειμένου να πραγματοποιήσουμε τα παραπάνω, γράψαμε ένα πρόγραμμα σε Perl, με το οποίο πραγματοποιούμε την αλληλεπίδραση με τη βάση δεδομένων και τις διάφορες λειτουργίες που απαιτούνται (διαγραφή και δημιουργία πινάκων, εκτέλεση εντολών mysql, μέτρηση χρόνων κοκ). Κατά τη διεξαγωγή του πειράματός μας, κάναμε τις εξής παραδοχές:

- Η μέτρηση του χρόνου γίνεται με τη χρήση της συνάρτησης gettimeofday(), του πακέτου Time::HiRes της perl. Μετράμε το χρόνο σε δευτερόλεπτα με ακρίβεια πολλών δεκαδικών ψηφίων χωρίς ωστόσο να μετράμε system clocks.
- Ανάμεσα στα ερωτήματα, εκτελούμε ένα sql select statement σε ένα μεγάλο πίνακα (με 700000 εγγραφές περίπου), προκειμένου να μην διατηρείται κάποια πληροφορία στη βάση δεδομένων λόγω caching από προηγούμενες αναζητήσεις, και τα αποτελέσματα των χρόνων να είναι όσο το δυνατόν πιο αξιόπιστα. Ο πίνακας στον οποίο εφαρμόζουμε το select statement περιέχει δεδομένα εντελώς διαφορετικά από τα πειραματικά.
- Συνολικά εκτελούμε το πείραμα 10 φορές για κάθε σύνολο πειραματικών δεδομένων και υπολογίζουμε το μέσο όρο των αποτελεσμάτων.
- Τα πειράματα εκτελούνται σε server του ερευνητικού κέντρου ΙΠΣΥΠ προκειμένου να ελαχιστοποιήσουμε τυχόν αστοχίες υλικού και καθυστερήσεις λόγω φόρτου εργασιών στον υπολογιστή.
- Προκειμένου να έχουμε όσο το δυνατόν αποδοτικότερη αναζήτηση, δεικτοδοτήσαμε τις στήλες εκείνες των πινάκων οι οποίες εμπλέκονται σε αυτή, οργανώνοντάς τες σε B-δένδρα, ενώ επιστρέψαμε τα καλύτερα 50 αποτελέσματα από τα 100 και με τις δύο μεθόδους.

Ο μέσος όρος των μετρήσεων για τα διάφορα μεγέθη grams και την edit distance, παρατίθεται ακολούθως για κάθε σύνολο πειραματικών δεδομένων (οι χρόνοι δίδονται σε δευτερόλεπτα):

Γράφημα 4.5:



Είναι εμφανής από τα παραπάνω η διαφορά στο χρόνο εκτέλεσης μεταξύ των μεθόδων που αξιοποιούν n-grams και αυτών της συντακτικής απόστασης. Οι χρόνοι στον κατακόρυφο άξονα εικονίζονται σε λογαριθμική κλίμακα, συνεπώς εύκολα διαπιστώνει κανείς ότι ο χρόνος εκτέλεσης για την υλοποίηση με edit distance είναι δεκαπλάσιος έως και εκατονταπλάσιος σε σχέση με τα n-grams. Όπως είναι αναμενόμενο, καθώς μεγαλώνει το μέγεθος των πειραματικών δεδομένων, οι διαφορές οξύνονται περισσότερο, μια και οι on-line τεχνικές για προσεγγιστικό ταίριασμα χαρακτήρων (approximate string matching) αποδίδουν χειρότερα σε σχέση με τις offline στα μεγάλα σύνολα δεδομένων.
Βέβαια, το κόστος για τη δημιουργία ενός inverted index, που απαιτούν οι τεχνικές που αξιοποιούν n-grams, δεν είναι αμελητέο, καθώς τα μεγέθη των αντιστοίχων πινάκων είναι συγκριτικά πολύ μεγαλύτερα

από τον αρχικό πίνακα, του οποίου παράγουμε τα grams. Επίσης σημαντικό είναι και το κόστος υπολογισμού του βάρους για κάθε gram (βλ. Γράφημα 4.1).
Ωστόσο, οι υπολογισμοί αυτοί λαμβάνουν χώρα μόνον κάθε φορά που χρειάζεται να προσθέσουμε νέες εγγραφές στη βάση δεδομένων. Ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής, μπορούμε να αξιολογήσουμε αν ο χρόνος για την διατήρηση ενός μηχανισμού αναζήτησης με grams είναι ή όχι απαγορευτικός, σε σχέση με την απόδοση της edit distance. Επίσης, είναι δυνατό να γίνουν προσαρμογές στη διαδικασία κατασκευής και ενημέρωσης του inverted index, όπως φερ ειπείν να προσθέτουμε απλά τις νέες τιμές στον ήδη υπάρχοντα κατάλογο, να διαγράφουμε και να δημιουργούμε εξ αρχής τον κατάλογο κοκ. Αντίστοιχα μπορούμε να προσαρμόσουμε και τη μηχανή αναζήτησης.
Εντύπωση προκαλεί εκ πρώτης όψεως η διαφορά στους χρόνους εκτέλεσης, αν αναλογιστούμε ότι ο πίνακας τον οποίο εξετάζουν οι μηχανισμοί με n-grams είναι συνήθως δεκαπλάσιος σε σχέση με τον πίνακα τον οποίο εξετάζει η edit distance (τον πίνακα δηλαδή από τον οποίο παραγάγαμε τα n-grams). Δεδομένου ότι έχουμε κατασκευάσει B-δένδρα για τους όρους σε κάθε έναν από τους πίνακες, θα έπρεπε οι μεγαλύτεροι πίνακες να παρουσιάζουν μεγαλύτερους χρόνους απόκρισης. Ο λόγοι για τους οποίους κάτι τέτοιο δεν παρατηρείται είναι οι εξής:

- Η mysql udf που χρησιμοποιείται για τον υπολογισμό της edit distance δεν έχει αμελητέα χρονική και χωρική πολυπλοκότητα, και δεδομένου ότι δεν εισάγουμε κάποιο περιορισμό στην τιμή της, ο αλγόριθμος δεν εκτελείται στον ελάχιστο δυνατό χρόνο (βλ ενότητα 2.1.2.1.1). Επιπλέον, η απόδοσή της μειώνεται όσο μεγαλώνει το μέγεθος των συμβολοσειρών που εξετάζουμε.
- Οι μέθοδοι που χρησιμοποιούν την edit distance εξετάζουν γραμμή γραμμή όλη τη βάση δεδομένων και πραγματοποιούν σύγκριση του όρου που συνιστά το αντικείμενο της αναζήτησης με κάθε εγγραφή χωριστά. Αντίθετα, η τεχνική που χρησιμοποιούμε με grams διαφόρων μεγεθών, δεν εξετάζει εν γένει κάθε εγγραφή του πίνακα. Όπως περιγράψαμε και στην ενότητα 3.4, ο όρος τον οποίο αναζητούμε, χωρίζεται σε επιμέρους συμβολοσειρές μεγέθους όσο και το μέγεθος του gram. Ως εκ τούτου, αξιοποιώντας τον inverted index, αναζητούμε μόνο τους όρους εκείνους με τους οποίους ο όρος μας μοιράζεται ένα τουλάχιστον gram. Η αναζήτηση αυτή πραγματοποιείται με χρήση μιας aggregate function και με την αξιοποίηση της εντολής GROUP BY, που μειώνει σημαντικά το ποσοστό του πίνακα το οποίο εξετάζουμε, δεδομένου ότι έχει προηγηθεί η δεικτοδότηση (indexing), των κατάλληλων πεδίων.
- Η εντολή GROUP BY συνιστά τον καθοριστικό παράγοντα και για τις διαφοροποιήσεις του χρόνου εκτέλεσης σε μηχανές αναζήτησης που έχουν υλοποιηθεί με grams παραπλησίου μεγέθους. Για παράδειγμα, ενώ το μέγεθος των πινάκων για 2-grams δεν είναι πολύ μεγαλύτερο από το αντίστοιχο μέγεθος για 3-grams, οι χρόνοι εκτέλεσης παρουσιάζουν αξιοσημείωτη διαφορά. Ο βασικός λόγος είναι ότι κατά την εκτέλεση της udf get_score(), ομαδοποιούμε τα grams βάσει της προέλευσής τους (τον πίνακα που περιέχει το term από το οποίο παρήχθη το gram), και ταξινομούμε τα αποτελέσματα βάσει του μεγαλυτέρου βάρους, όπως περιγράφηκε στην ενότητα 3.3.4.2. Όταν χρησιμοποιείται μεγαλύτερο μέγεθος gram, δημιουργούνται περισσότερες μικρές ομάδες λόγω της εντολής group by, αλλά επειδή δεν είναι πιθανό να υπάρχουν πολλές ομάδες που να περιέχουν κοινά grams με τη συμβολοσειρά που αναζητούμε για μεγάλο μέγεθος gram, εν τέλει καταλήγουμε να εξετάζουμε λιγότερες εγγραφές.

4.3 Μέτρηση της ακρίβειας των μεθόδων που χρησιμοποιούν grams

4.3.1 Πραγματικό σύνολο δεδομένων

Ένα ερώτημα το οποίο ανακύπτει μελετώντας τις προηγούμενες ενότητες, είναι κατά πόσον οι διαφορές στην χρονική απόδοση των δύο κυριοτέρων τεχνικών που μας έχουν απασχολήσει, δηλαδή των μηχανισμών που αξιοποιούν την συντακτική απόσταση (edit distance), και αυτών που χρησιμοποιούν grams, έχουν αντίκτυπο στα χαρακτηριστικά των παρεχόμενων αποτελεσμάτων. Στο πείραμα αυτό, θα αξιολογήσουμε τη συνάφεια μεταξύ των συνόλων που επιστρέφουν οι δύο τεχνικές ως απάντηση σε κοινά σύνολα ερωτημάτων.

Παραλλάσσοντας τη γνωστή μέθοδο που χρησιμοποιείται κατά την ανεύρεση πληροφοριών, και που, θεωρώντας ως αληθές ένα σύνολο δεδομένων, μετρά τα μεγέθη recall και precision για ένα άλλο σύνολο από documents, στο πείραμα αυτό, θα θεωρήσουμε αληθή τα αποτελέσματα που επιστρέφουν οι μηχανές αναζήτησης που χρησιμοποιούν edit distance. Θα μετρήσουμε δηλαδή το recall και το precision που εμφανίζουν τα διαφορετικά μεγέθη grams, ως προς τα αποτελέσματα της edit distance, και όχι βάσει των πραγματικά ορθών αποτελεσμάτων, τα οποία θα έπρεπε να προκύψουν έπειτα από αξιολόγηση των ειδικών βιολόγων που δημιούργησαν τους αλγορίθμους της εφαρμογής DIANA. Η συγκεκριμένη τακτική θα μας εξασφαλίσει ένα μέτρο αξιολόγησης του βαθμού στον οποίο η αντικατάσταση μίας μηχανής αναζήτησης που αξιοποιεί συντακτική απόσταση με μία που χρησιμοποιεί ευρετήρια, επιφέρει αλλαγές στη φύση των αποτελεσμάτων. Ως εκ τούτου, θα γνωρίζουμε και τις επιπτώσεις της αντικατάστασης αυτής στην εφαρμογή ιστού DIANA micro-T.

Η διαδικασία εκτέλεσης του πειράματος έχει ως εξής:

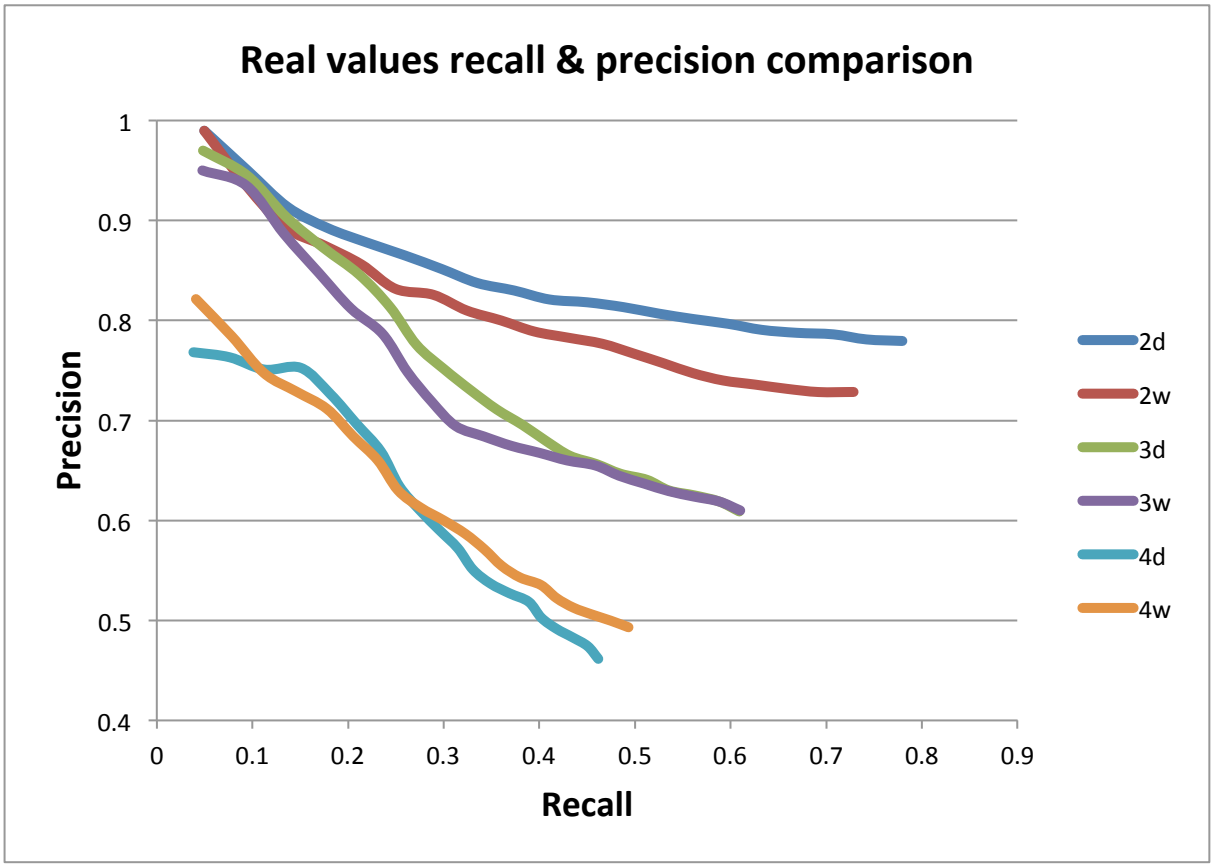
- Κατασκευάζουμε με τυχαίο τρόπο ένα σύνολο ερωτημάτων ως εξής: Επιλέγουμε τυχαία ορισμένες εγγραφές από τη βάση δεδομένων και τις τροποποιούμε τυχαία αλλάζοντας 1 έως 3 χαρακτήρες σε κάθε εγγραφή. Με τον τρόπο αυτό κατασκευάζουμε τα πειραματικά μας δεδομένα.
- Για κάθε ένα όρο από το σύνολο των ερωτημάτων που κατασκευάσαμε (testcase set), εκτελούμε ένα ερώτημα αναζήτησης στη βάση δεδομένων και επιστρέφουμε τα 20 καλύτερα αποτελέσματα, βάσει της συντακτικής απόστασης.
- Εξετάζουμε την τιμή της συντακτικής απόστασης για το τελευταίο από τα αποτελέσματα που επιστρέψαμε (το χειρότερο από τα 20). Έστω k αυτή.
- Επιστρέφουμε όλες τις τιμές από το σύνολο δεδομένων , που η συντακτική τους απόσταση από τον όρο που θέσαμε ως ερώτημα είναι μικρότερη ή ίση με k. Έστω A το σύνολο τον όρων που λάβαμε ως απάντηση.
- Επιστρέφουμε τα 20 καλύτερα αποτελέσματα για μεταβλητό μέγεθος gram και διαφορετικές μεθόδους υπολογισμού του βάρους για κάθε gram. Έσω $B_{i,j,n}$ το αντίστοιχο σύνολο αποτελεσμάτων, με μέγεθος gram i, και βάρος j, όταν έχουμε λάβει n αποτελέσματα.
- Υπολογίζουμε για τα δύο σύνολα A, B, τα μεγέθη recall και precision στο βήμα n.

Με την παραπάνω μέθοδο εξασφαλίζουμε ότι λαμβάνουμε μία πλήρη εικόνα των συνόλων των απαντήσεων που λαμβάνουμε με τις δύο μεθόδους και συγκεκριμένα, αποκτούμε πληρέστερη εικόνα ως προς το αν οι απαντήσεις που λαμβάνουμε με χρήση ευρετηρίου, περιέχονται με συγκεκριμένο βαθμό

βεβαιότητας στις απαντήσεις που λαμβάνουμε με χρήση της συντακτικής απόστασης. Αυτό είναι και το κύριο σημείο που μας ενδιαφέρει σε ένα σενάριο αντικατάστασης του μηχανισμού αναζήτησης, καθώς, δεδομένου ότι η ποιότητα των αποτελεσμάτων βρίσκεται ήδη σε ικανοποιητικά επίπεδα, δεν επιθυμούμε να πετύχουμε χειρότερα αποτελέσματα με τη νέα μέθοδο.

Η γραφική παράσταση των μετρήσεων που λάβαμε με την εκτέλεση του πειράματος είναι η εξής:

Γράφημα 4.6:



Παρατηρούμε ότι τα μεγέθη recall και precision παρουσιάζουν μειωμένες τιμές όσο αυξάνεται το μέγεθος των gram. Βέβαια, για μικρά μεγέθη gram η ακρίβεια παραμένει σε πολύ υψηλά επίπεδα, τουλάχιστον για τις 10 πρώτες επιστρεφόμενες τιμές. Η γενικότερα παρατηρούμενη μείωση είναι αναμενόμενη λόγω της μη ομοιόμορφης κατανομής των χαρακτηριστικών του συνόλου δεδομένων, η οποία οδηγεί σε διαφορετικές συμπεριφορές μεταξύ των μεθόδων που αξιοποιούν grams και των αντίστοιχων της συντακτικής απόστασης. Όπως αναφέρθηκε στις προηγούμενες ενότητες , η επιλεκτικότητα αυξάνεται όσο αυξάνεται το μέγεθος του gram, με την έννοια ότι το εύρους των επιλογών που παρέχεται ως απάντηση σε μία ερώτηση μειώνεται δραματικά. Ως εκ τούτου, για μέγεθος gram 4, επιστρέφονται πολύ λιγότερα αποτελέσματα σε σχέση με τα αντίστοιχα των 2-grams, οπότε είναι λογικό το μέγεθος recall, δηλαδή το ποσοστό των συνολικά σωστών απαντήσεων που επεστράφησαν, ως προς το συνολικό αριθμό των σωστών απαντήσεων, να μειώνεται. Η ακρίβεια αντίστοιχα μειώνεται για τα 4-grams, λόγω της

φύσης των αποτελεσμάτων που επιστρέφουν οι τεχνικές ευρετηρίου και οι τεχνικές συντακτικής απόστασης. Για παράδειγμα, εάν κάποιος χρήστης αναζητήσει ως όρο τον όρο 'hsa-cat' και η μέθοδος που χρησιμοποιεί συντακτική απόσταση επιστρέψει 10 όρους της μορφής 'hsa-cat-1' ως αποτελέσματα (δηλαδή όρους που έχουν συντακτική απόσταση το πολύ 2), θα αγνοηθούν αποτελέσματα της μορφής 'hsa-cat-12-ca', παρόλο που αυτά μπορεί να είναι συναφή με το ερώτημα. Το αντίστοιχο αποτέλεσμα δεν θα περάσει απαρατήρητο από μία μηχανή αναζήτησης που χρησιμοποιεί τεχνικές ευρετηρίου, ωστόσο το μέγεθος της ακρίβειας θα εμφανιστεί μειωμένο στο αντίστοιχο πείραμα.

Η ανομοιομορφία στην κατανομή των χαρακτήρων συντελεί προσθετικά στην ενίσχυση των διαφορών μεταξύ των αποτελεσμάτων που επιστρέφουν τα διαφορετικά μεγέθη gram. Επιπρόσθετα έχει και ως συνέπεια να υπάρχουν εντονότερες διαφοροποιήσεις και στα αποτελέσματα που πετυχαίνουμε χρησιμοποιώντας διαφορετικά βάρη για την ταξινόμηση των gram. Χαρακτηριστικό είναι ότι τα αποτελέσματα που λαμβάνουμε χρησιμοποιώντας το βάρος ‘weight1’ είναι τις περισσότερες φορές χειρότερα σε σχέση με τα αντίστοιχα του ‘dummyweight’ (το οποίο αντιστοιχεί σε απλή απαρίθμηση των grams). Ωστόσο, αυτές οι διαφοροποιήσεις αφορούν πάντα στη συνάφεια με τα αποτελέσματα της συντακτικής απόστασης και όχι σε σχέση με τα πραγματικά μεγέθη, τα οποία θα έπρεπε να προκύψουν από ομάδες ειδικών βιολόγων που θα αξιολογούσαν την ποιότητα των αποτελεσμάτων σε σχέση με τα κατά την κρίση τους ορθά.

4.3.2 Συνθετικό σύνολο δεδομένων

Ακολουθώντας παρόμοιες μεθόδους με τις αντίστοιχες της ενότητας 4.3.1, υπολογίζουμε τη συνάφεια των πειραματικών αποτελεσμάτων των μεθόδων που αξιοποιούν ευρετήρια gram και της συντακτικής απόστασης (edit distance). Κατασκευάζουμε ένα σύνολο πειραματικών δεδομένων διαφορετικό από αυτό που χρησιμοποιήθηκε στα προηγούμενα ερωτήματα. Το σύνολο αυτό αποτελείται από ένα πίνακα που περιέχει 100000 εγγραφές. Κάθε εγγραφή είναι μία συμβολοσειρά που αποτελείται από 11 χαρακτήρες.
Η διαδικασία που ακολουθούμε προκειμένου να κατασκευάσουμε τα πειραματικά μας δεδομένα είναι η εξής:

- Παράγουμε 10.000 τυχαίες συμβολοσειρές με τα χαρακτηριστικά που περιγράψαμε
- Για κάθε μία από αυτές παράγουμε δέκα συμβολοσειρές με αλλαγμένους 1 έως 2 χαρακτήρες με τυχαίο τρόπο. Συνεπώς, εισάγονται συνολικά 100000 εγγραφές που διαθέτουν ανά 10 (τουλάχιστον) κοινά χαρακτηριστικά (edit distance <=2).

Ο λόγος για τον οποίο παράγουμε τα πειραματικά μας δεδομένα κατ' αυτό τον τρόπο είναι προκειμένου να προσομοιώσουμε όσο το δυνατόν καλύτερα τα δεδομένα ενός συστήματος όπως το DIANA micro-T, όπου υπάρχουν μεγάλες ομάδες δεδομένων που παρουσιάζουν κοινά χαρακτηριστικά μεταξύ τους. Κάτι τέτοιο δεν ίσχυε στα προηγούμενα συνθετικά datasets όπου τα δεδομένα ήταν εντελώς τυχαία. Σε περίπτωση που χρησιμοποιούσαμε τις πειραματικές τιμές που κατασκευάσαμε για τα συνθετικά δεδομένα στις προηγούμενες ενότητες, και επιστρέφαμε τα καλύτερα 10 αποτελέσματα για κάποιο ερώτημα, ο μηχανισμός που χρησιμοποιεί edit distance θα επέστρεφε ως επί το πλείστον τιμές που δεν είναι συναφείς με το ερώτημα.

Για παράδειγμα, στα συνθετικά δεδομένα της ενότητας 4.2.2, έστω ό,τι εφαρμόζεται ένα ερώτημα στην βάση και ζητείται να επιστραφούν ως απάντηση τα 10 καλύτερα αποτελέσματα, ταξινομημένα ως

προς τη συντακτική απόσταση των όρων της βάσης σε σχέση με τον όρο που εισήχθη ως ερώτημα. Ο όρος είναι μία συμβολοσειρά που προέκυψε παραλλάσσοντας κατά ένα χαρακτήρα κάποιο στοιχείο της βάσης (και μόνον αυτό). Σχεδόν πάντοτε, τα αποτελέσματα που θα επέστρεφε το αντίστοιχο ερώτημα sql θα είχαν, αν εξαιρέσουμε το πρώτο (που είναι και το στοιχείο που τροποποιήσαμε προκειμένου να κατασκευάσουμε το ερώτημα), edit distance >=6 σε σχέση με τη συμβολοσειρά που εισάγαμε. Ο κύριος λόγος είναι η τυχαία φύση της κατασκευής των δεδομένων. Η μεγάλη συντακτική απόσταση των αποτελεσμάτων, πρακτικά σημαίνει ότι εάν φερ ειπείν αναζητούσαμε τον όρο 'cat' και ζητούσαμε τα 10 καλύτερα αποτελέσματα από κάθε μηχανή, ενδεχομένως να λαμβάναμε ως απάντηση και τον όρο 'tomato' από την edit distance, ο οποίος προφανώς δεν είναι συναφής με το ερώτημα. Αντίθετα, τα n-grams θα μας έδιναν αποκλειστικά τους όρους με τους οποίους η λέξη 'cat' έχει κοινά 2-grams ή 3-grams, οι οποίοι όμως ενδεχομένως να είναι λιγότεροι από 10. Κάτι τέτοιο δεν συμβαίνει σε μία βάση δεδομένων όπως αυτή του συστήματος DIANA micro-T όπου υπάρχουν πολλοί όροι με κοινά γνωρίσματα. Συνεπώς, επιλέγοντας τη νέα μέθοδο δημιουργίας πειραματικών δεδομένων, εξασφαλίζουμε ότι και οι δύο τεχνικές (ευρετηρίων και συντακτικής απόστασης) θα μας επιστρέψουν τον ίδιο αριθμό αποτελεσμάτων και τα αποτελέσματα αυτά θα έχουν κάποια συνάφεια.

Η διαδικασία που ακολουθούμε για τη διεξαγωγή του πειράματος έχει ως εξής:

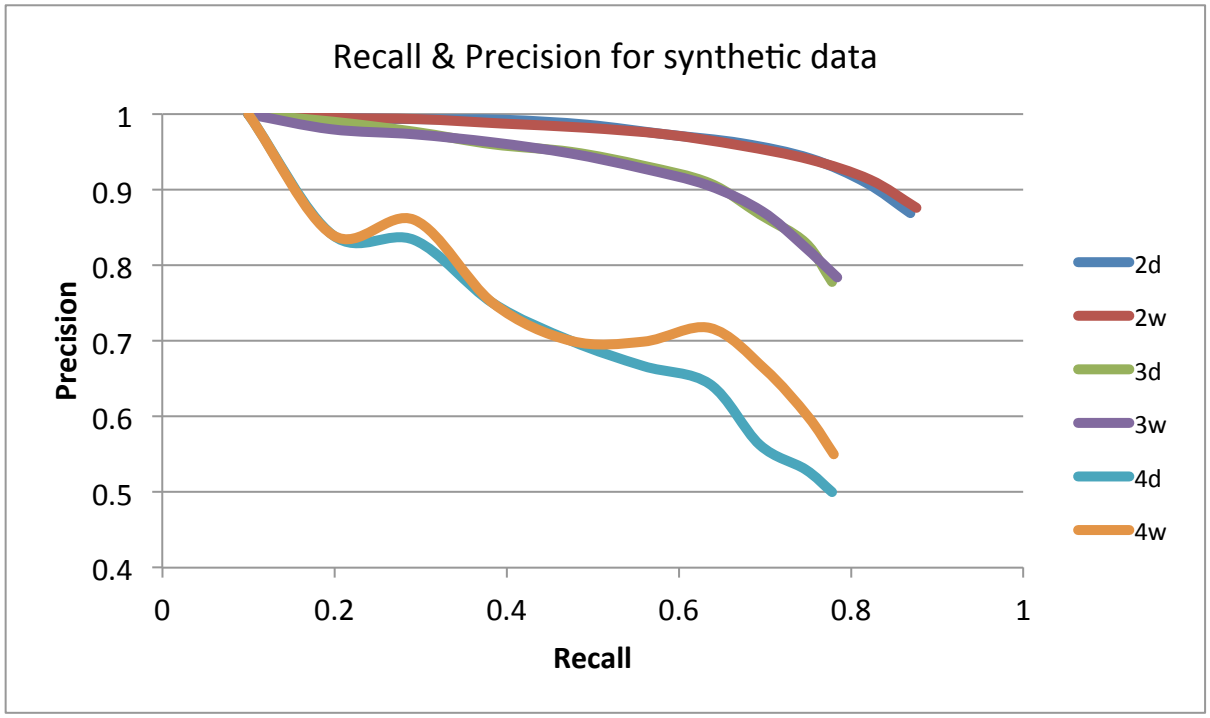
- Για κάθε ένα όρο από το σύνολο των ερωτημάτων που κατασκευάσαμε (testcase set), επιστρέφουμε τα 20 καλύτερα αποτελέσματα, βάσει της συντακτικής απόστασης.
- Εξετάζουμε την τιμή της συντακτικής απόστασης για το τελευταίο από τα αποτελέσματα που επιστρέψαμε (το χειρότερο από τα 20), σε σχέση με τον όρο που συνιστά το ερώτημα. Έστω k αυτή.
- Επιστρέφουμε όλες τις τιμές από το σύνολο δεδομένων, που η συντακτική τους απόσταση από τον όρο που θέσαμε ως ερώτημα είναι μικρότερη ή ίση με k. Έστω A το σύνολο τον όρων που λάβαμε ως απάντηση.
- Επιστρέφουμε τα 20 καλύτερα αποτελέσματα για μεταβλητό μέγεθος gram και διαφορετικές μεθόδους υπολογισμού του βάρους για κάθε gram. Έσω $B_{i,j,n}$ το αντίστοιχο σύνολο αποτελεσμάτων, με μέγεθος gram i, και βάρος j, όταν έχουμε λάβει n αποτελέσματα.
- Υπολογίζουμε για τα δύο σύνολα A, B, τα μεγέθη recall και precision στο βήμα n.
- Χρησιμοποιούμε μεγέθη gram από 2 έως 4.
- Οι πειραματικές τιμές λαμβάνονται για δύο βάρη.

Από τα βάρη που χρησιμοποιήθηκαν στο πείραμα, το πρώτο λαμβάνεται ίσο με τη μονάδα (αντιστοιχεί στο dummyweight των πινάκων του inverted index), ενώ το δεύτερο δίδεται από τη σχέση 8 της ενότητας 2.2.4 (αντιστοιχεί στο weight1). Στα δεδομένα στα οποία εφαρμόσαμε το πείραμά μας, χρησιμοποιήσαμε και τη σχέση 9 της ενότητας 2.2.4, ωστόσο τα αποτελέσματα ήταν παρόμοια με αυτά της σχέσης 8, και ως εκ τούτου τα παραλείπουμε εδώ. Ο λόγος για τον οποίο προκύπτει αυτή η ομοιότητα είναι ότι στην πλειονότητα των περιπτώσεων, $(1 + \log f_{d,t}) = f_{d,t} = 1$, καθώς ένα gram δεν εμφανίζεται δύο φορές μέσα στην ίδια συμβολοσειρά στα τυχαία κατασκευασμένα δεδομένα. Ωστόσο, μεγάλη ομοιότητα παρουσιάζεται παρά τη διαφοροποίηση στα βάρη και όταν το μέγεθος του gram είναι μικρό. Ο λόγος είναι ότι τα δεδομένα είναι κατασκευασμένα με τυχαίο τρόπο και η κατανομή αυτή ουσιαστικά διατηρείται ανεξαρτήτως μετρικής σύγκρισης. Δηλαδή, τα σχετικά αποτελέσματα ως προς το βάρος μιας λέξης έναντι μιας άλλης, διατηρούνται ανεξαρτήτως βάρους λόγω της ομοιόμορφης κατανομής. Κάτι τέτοιο δεν ισχύει

για τα πραγματικά μας δεδομένα, όπου οι διαφορές για ίδιο αριθμό gram και διαφορετικά βάρη ήταν εντονότερες.

Τα αποτελέσματα που λάβαμε για μεγέθη gram 2 έως 4 από την πειραματική διαδικασία που ακολουθήσαμε, εικονίζονται ακολούθως:

Γράφημα 4.7:



Παρατηρούμε ότι η ακρίβεια (precision) διατηρείται πολύ υψηλή για τις πρώτες 10 τουλάχιστον τιμές στο σύνολο των απαντήσεων. Δηλαδή, οι πρώτες 10 έως και 15 τιμές που επιστρέφουν οι μέθοδοι που αξιοποιούν ευρετήρια gram, στο 85% ανήκουν στο σύνολο των αποτελεσμάτων που επιστρέφουν οι μέθοδοι που χρησιμοποιούν συντακτική απόσταση. Η ακρίβεια των αποτελεσμάτων είναι μεγαλύτερη όσο το μέγεθος των gram μειώνεται. Αυτό είναι κάτι το αναμενόμενο καθώς, όσο μικρότερο είναι το μέγεθος του gram, τόσο αυξάνει και η λεπτομέρεια με την οποία εξετάζουμε τα δομικά στοιχεία μιας συμβολοσειράς, δηλαδή τις υπακολουθίες που την συνθέτουν. Αντίστοιχα, όσο αυξάνει το μέγεθος του gram, τόσο λιγότερες εγγραφές του ευρετηρίου ταυτίζονται με τον όρο προς αναζήτηση και ως εκ τούτου μειώνεται ο χρόνος αναζήτησης (όπως δείξαμε στο γράφημα 4.5 της προηγούμενης ενότητας). Παρατηρούμε στη γραφική παράσταση ότι η χρήση διαφορετικών βαρών για τα grams δεν φαίνεται να επηρεάζει ιδιαίτερα τα αποτελέσματα για το συγκεκριμένο σύνολο δεδομένων. Ιδιαίτερα για μεγέθη gram 2 ή 3 οι καμπύλες ταυτίζονται, ενώ για μέγεθος gram 4 παρατηρούμε ένα μικρό προβάδισμα του weight1 ως προς την ποιότητα των αποτελεσμάτων.

4.4 Μέτρηση χρονικής απόκρισης για μεταβλητό μέγεθος αποτελεσμάτων

Ακολούθως εξετάζουμε αν επηρεάζεται η συμπεριφορά των δύο μεθόδων ανάλογα με τον αριθμό των αποτελεσμάτων που επιστρέφονται στο χρήστη. Στα πειράματα που πραγματοποιήσαμε παρατηρήθηκε μία σταθερή συμπεριφορά παρά τις αλλαγές στο άνω όριο των αποτελεσμάτων που λαμβάνει ο χρήστης είτε με edit distance είτε με n-grams. Χαρακτηριστικά παραθέτουμε το παρακάτω γράφημα:

Γράφημα 4.8:

Στο γράφημα αυτό, μεταβάλλουμε για μέγεθος gram 2, το όριο των αποτελεσμάτων που λαμβάνουμε αν εκτελέσουμε ένα ερώτημα στο σύνολο D. Παρατηρούμε ότι οι χρόνοι εκτέλεσης για edit distance (edD) και 2-grams (grD), παραμένουν σταθεροί, παρουσιάζοντας μια μικρή αύξηση καθώς το όριο των αποτελεσμάτων αυξάνεται από 20 έως και 50 εγγραφές. Ωστόσο, καθώς τα πειράματα εκτελέστηκαν σε προσωπικό υπολογιστή και όχι σε κάποιο μεγαλύτερο υπολογιστικό σύστημα, οι διαφορές αυτές οφείλονται σε αστοχίες διαχείρισης μνήμης, γεγονός το οποίο και επαληθεύτηκε όταν εκτελέσαμε το πείραμα σε υπολογιστή με περισσότερους πόρους.

4.5 Πλεονεκτήματα και μειονεκτήματα της κάθε μεθόδου

Στις προηγούμενες ενότητες, δόθηκε κυριότερα έμφαση στο κατά πόσον οι τεχνικές που χρησιμοποιούν ευρετήρια gram μπορούν να βελτιώσουν τη χρονική απόκριση μιας μηχανής αναζήτησης, σε σχέση με τις μεθόδους συντακτικής απόστασης, διατηρώντας παράλληλα τα ποιοτικά χαρακτηριστικά των τελευταίων ως προς τα παρεχόμενα αποτελέσματα. Ωστόσο, είναι αναγκαίο να καταστεί σαφές ότι πρόκειται για διαφορετικές στη βάση τους μεθόδους, με διαφορετική φύση αποτελεσμάτων, η οποία ανακύπτει ακριβώς από τα δομικά στοιχεία που τις διακρίνουν σε επίπεδο αλγοριθμικό.
Συγκεκριμένα, οι μηχανές οι οποίες χρησιμοποιούν τη συντακτική απόσταση, έχουν την τάση να αποδίδουν καλύτερα σε μικρά σύνολα δεδομένων αλλά και σε μικρότερα μεγέθη συμβολοσειρών.

Ωστόσο, αγνοούν ενδεχομένως ποιοτικά χαρακτηριστικά που μόνον οι τεχνικές ευρετηρίων μπορούν να αποδώσουν. Για παράδειγμα, αν τεθεί προς αναζήτηση σε μία μηχανή που χρησιμοποιεί συντακτική απόσταση, ο όρος ‘hsa-miR-cat’, πιθανότατα θα επιστραφούν ως αποτελέσματα όροι της μορφής 'hsa-miR-bat','hsa-miR-rat','hsa-miR21at' . Η μηχανή θα αγνοήσει δηλαδή όρους της μορφής 'hsa-miR-cat-c1123', hsa-miR-cat-cad123', οι οποίοι έχουν μεγάλη συντακτική απόσταση από τον όρο που εξετάζουμε, παρόλο που τα αποτελέσματα αυτά ενδεχομένως να είναι περισσότερο συναφή με τον όρο. Ένας μηχανισμός ευρετηρίου, θα απέδιδε καλύτερα στην περίπτωση αυτή.

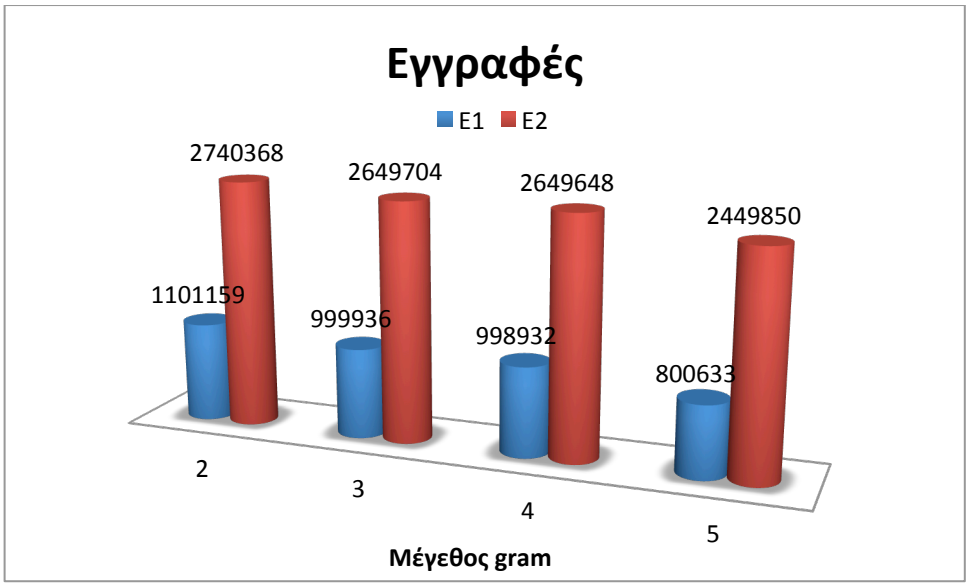
Αντίστοιχα, εάν στη βάση δεδομένων υπάρχει ο όρος 'hsa-cat' και κάποιος χρήστης πληκτρολογήσει εσφαλμένα στη μηχανή αναζήτησης τον όρο 'haa-cat', ένας μηχανισμός με ευρετήριο gram, δεν θα επιστρέψει ποτέ αποτέλεσμα που θα έχει ως πρόθεμα τη συμβολοσειρά 'hsa', καθώς δεν υπάρχει αυτή η συμβολοσειρά στον όρο που τέθηκε προς αναζήτηση. Συνεπώς, ένας μηχανισμός συντακτικής απόστασης αποδίδει καλύτερα σε αυτή την περίπτωση. Με το παράδειγμα αυτό καθίσταται σαφέστερος ο λόγος για τον οποίο οι μηχανισμοί ευρετηρίων αποδίδουν χειρότερα σε δεδομένα που χαρακτηρίζονται από μικρά μήκη συμβολοσειρών. Σε αυτού του τύπου τα δεδομένα, ένα λάθος σε ένα και μόνο χαρακτήρα, ενδεχομένως να αποκλείσει μία ολόκληρη ομάδα αποτελεσμάτων, καθώς η μηχανή αδυνατεί να εντοπίσει κοινά grams με τον εσφαλμένα πληκτρολογηθέντα όρο. Συνεπώς, όπως συμβαίνει με τις περισσότερες τενικές, δεν υπάρχει πανάκεια ως προς την αποτελεσματικότερη μέθοδο. Αντίθετα, πρέπει κάθε περίπτωση να υφίσταται ειδική αξιολόγηση ως προς τις απαιτήσεις και τους διαθέσιμους πόρους και να επιλέγεται η κατάλληλη τεχνική, σύμφωνα με τις εκάστοστε ανάγκες της κάθε εφαρμογής.

Ως προς το μέγεθος των δεδομένων, όπως φάνηκε και από τα σχετικά γραφήματα της ενότητας 4, οι μηχανισμοί που χρησιμοποιούν συντακτική απόσταση, αποδίδουν χειρότερα με την αύξηση του όγκου των δεδομένων. Ο λόγος είναι ότι στις περισσότερες των περιπτώσεων, είναι αναγκαίο να διατρέξουμε όλο τον όγκο των δεδομένων και να επιτελέσουμε μία πράξη υπολογισμού σε κάθε εγγραφή. Το συνολικό κόστος είναι πολύ μεγαλύτερο από αυτό της απλής σύγκρισης που απαιτούν στη μέση περίπτωση οι μηχανισμοί ευρετηρίου. Ωστόσο και στους τελευταίους, ο μεγάλος όγκος δεδομένων και κυριότερα το μέγεθος των συμβολοσειρών, αυξάνουν γραμμικά το μέγεθος του ευρετηρίου, προσθέτοντας έτσι φόρτο στον συνολικό χρόνο επεξεργασίας των ερωτημάτων. Ο φόρτος αυτός αυξάνεται και από το γεγονός ότι ένα μεγαλύτερο ευρετήριο, συνήθως οδηγεί και σε περισσότερα αποτελέσματα ανά ερώτημα, και άρα σε μεγαλύτερο υπολογιστικό χρόνο. Το φαινόμενο αυτό εντείνεται περισσότερο όσο μειώνεται το μέγεθος του gram, καθώς η επιλεκτικότητα, με την έννοια που τη χρησιμοποιούμε στην παρούσα εργασία, τείνει να μειώνεται. Σε ορισμένες περιπτώσεις, ανάλογα και με την στατιστική κατανομή των χαρακτήρων του κειμένου, μπορεί για τον ίδιο αριθμό εγγραφών να εμφανιστούν πολύ μεγαλύτεροι inverted indexes, οι οποίοι μάλιστα δεν θα παρουσιάζουν αξιοσημείωτη μείωση στον αριθμό των εγγραφών καθώς αυξάνεται το μέγεθος του gram.

Προκειμένου να κατανοήσουμε τον αντίκτυπο που έχει το μήκος της συμβολοσειράς και η φύση των δεδομένων στο μέγεθος του ευρετηρίου, εξετάζουμε το ακόλουθο παράδειγμα: Έστω ότι παράγουμε ένα σύνολο δεδομένων Ε1, το οποίο αποτελείται από 100.000 εγγραφές, μήκους 10 έως 14 χαρακτήρων η κάθε μία. Έστω Ε2 ένα άλλο σύνολο 100.000 εγγραφών, το οποίο κατασκευάσαμε επιλέγοντας για κάθε συμβολοσειρά να εισάγουμε και 9 παρόμοιές της (που έχουν μεταξύ τους συντακτική το πολύ 2). Ωστόσο τα μήκη των εγγραφών για το σύνολο Ε2 κυμαίνονται μεταξύ 19-30 χαρακτήρων.

Παραθέτουμε τα μεγέθη των αντιστοίχων ευρετηρίων:

Γράφημα 5.1:



Παρατηρούμε ότι για το δεύτερο σύνολο (εικονίζεται με κόκκινο στο παραπάνω γράφημα), ο αριθμός των εγγραφών του ευρετηρίου είναι παραπάνω από διπλάσιος σε σχέση με το πρώτο, παρόλο που ο συνολικός αριθμός εγγραφών της βάσης δεδομένων είναι ο ίδιος σε κάθε περίπτωση. Συνεπώς, η διαδικασία κατασκευής του ευρετηρίου αλλά και οι επιπτώσεις που έχει η φύση των δεδομένων σε αυτό δεν είναι σε καμία περίπτωση αμελητέες, και είναι αναγκαίο σε κάθε περίπτωση να αξιολογούμε αν το κόστος δημιουργίας του ευρετηρίου, το κόστος αναζήτησης αλλά και το κόστος συντήρησής του είναι τέτοιο που να δικαιολογεί τη χρήση του.

Τέλος, οφείλουμε να παρατηρήσουμε ότι οι τεχνικές αναζήτησης που δομούνται επί της συντακτικής απόστασης, παρουσιάζουν εν γένει σταθερή συμπεριφορά ανεξαρτήτως του υλικού επί του οποίου εκτελείται η όποια εφαρμογή τις χρησιμοποιεί. Δηλαδή, οι όποιες μεταβολές στην απόδοση, είναι ανάλογες των πόρων του υλικού. Κάτι τέτοιο δεν συμβαίνει απαραίτητα και με τις τεχνικές ευρετηρίου, όπου σημαντική είναι η επίδραση της διαχείρισης μνήμης από το υλικό. Συγκεκριμένα, ενδεχομένως σε ένα υπολογιστικό σύστημα με επαρκή μνήμη RAM και ταχύτερη cache μνήμη οι μηχανισμοί ευρετηρίου να αποδίδουν καλύτερα από τους αντίστοιχους συντακτικής απόστασης, και σε κάποιο άλλο υπολογιστικό σύστημα όπου δεν υπάρχουν αρκετοί πόροι για τη διαχείριση της μνήμης, να αποδίδουν χειρότερα. Αυτή η αστάθεια είναι ένας παράγοντας που πρέπει να ληφθεί υπ'όψιν προτού επιλέξουμε το καταλληλότερο σύστημα για μία εφαρμογή.

4.6 Επιλογή των παραμέτρων αναζήτησης για το σύστημα DIANA

Βάσει όλων των προηγούμενων παρατηρήσεων αλλά και των πειραματικών μετρήσεων που πραγματοποιήθηκαν στην ενότητα 4, επιλέγουμε τις εξής παραμέτρους για το μηχανισμό αναζήτησης του συστήματος DIANA micro-T.

- Όλες οι εργασίες αναζήτησης θα γίνονται πλέον από τη βάση δεδομένων, είτε πρόκειται για αναζητήσεις με χρήση συντακτικής απόστασης, είτε για αναζητήσεις με χρήση ευρετηρίου.
- Επιλέγεται ως μέγεθος gram το 3, προκειμένου να εξασφαλίσουμε ταχύτερα αποτελέσματα με ικανοποιητική απόδοση στην ακρίβεια των αποτελεσμάτων. Μεγέθη gram μεγαλύτερα του 3 παρουσιάζουν μη αμελητέες διαφορές σε σχέση με τον τρέχοντα μηχανισμό αναζήτησης ως προς τα αποτελέσματα, ενώ τα 2-grams εμφανίζουν χρονικές καθυστερήσεις έναντι των 3-grams, οι οποίες δεν αντιστοιχούν σε δραματική βελτίωση της ποιότητας των αποτελεσμάτων, και ως εκ τούτου μπορούμε να τα απορρίψουμε.
- Θα χρησιμοποιηθούν υβριδικές μέθοδοι αναζήτησης ανάλογα με το είδος των συμβολοσειρών που εισάγει ο χρήστης. Σε πίνακες όπου το μήκος των εγγραφών είναι μικρότερα θα χρησιμοποιηθεί η συντακτική απόσταση, υλοποιημένη αυτή τη φορά στη βάση δεδομένων, ενώ στους υπόλοιπους πίνακες θα χρησιμοποιηθεί ευρετήριο.

5

Επίλογος

5.1 Σύνοψη

Στην παρούσα εργασία εξετάσαμε ορισμένες από τις δημοφιλέστερες μεθόδους για την κατασκευή μηχανών αναζήτησης για εφαρμογές ιστού όπως αυτή του συστήματος DIANA micro-T. Αναπτύχθηκαν κατάλληλα εργαλεία για τη διαχείριση και την κατασκευή τέτοιων μηχανών αναζήτησης, με κύρια έμφαση σε τεχνικές συντακτικής απόστασης και ευρετηρίων gram. Τα εργαλεία αυτά περιλαμβάνουν κατά κύριο λόγο προγράμματα γραμμένα σε Perl για την κατασκευή ευρετηρίων και μία σειρά από mysql udfs που επιτελούν εργασίες συναφείς με n-grams. Παράλληλα, έγινε χρήση της php και τεχνολογιών ajax για την τροποποίηση του γραφικού περιβάλλοντος του συστήματος DIANA, μέσω του yii framework.

Αξιολογήσαμε την απόδοση των παραπάνω τεχνικών αναζήτησης τόσο με συνθετικά όσο και με πραγματικά δεδομένα. Τα δεδομένα κατασκευάστηκαν κατά τρόπο ώστε να είναι όσο το δυνατόν αντικειμενικότερα ως προς την απόδοση των αποτελεσμάτων. Βάσει των πειραμάτων που διεξήχθησαν, επιλέξαμε τις παραμέτρους αναζήτησης τις οποίες και ενσωματώσαμε στην εφαρμογή ιστού DIANA, με παρεμβάσεις τόσο στον αλγόριθμο που χρησιμοποιείται στην αναζήτηση, όσο και στο γραφικό περιβάλλον και τον τρόπο παροχής των αποτελεσμάτων στο χρήστη.

Συνοπτικά, τα κυριότερα αποτελέσματα της παρούσης εργασίας είναι τα εξής:

- Πετύχαμε σημαντική βελτίωση των χρόνων απόκρισης της μηχανής αναζήτησης του συστήματος DIANA.
- Βελτιώθηκε η ποιότητα των παρεχόμενων αποτελεσμάτων της εφαρμογής σε επίπεδο εύρους προτάσεων.
- Βελτιώθηκε του περιβάλλον αναζήτησης για τους χρήστες οι οποίοι πλέον λαμβάνουν αποτελέσματα ενώ πληκτρολογούν.
- Δημιουργήθηκαν εργαλεία κατασκευής inverted index και διαχείρισης της βάσης δεδομένων για τους διαχειριστές τόσο της εφαρμογής DIANA όσο και οποιουδήποτε άλλου συστήματος. Ο διαχειριστής του συστήματος έχει τη δυνατότητα να επιλέξει την κατασκευή index με

οποιοδήποτε αριθμό grams και να την επιλογή του βάρους που θα χρησιμοποιηθεί για τα grams.
- Δημιουργήθηκε μία σειρά από mysql udfs για approximate string matching με τεχνικές n-grams.
- Επεκτάθηκε το πακέτο λογισμικού flamingo ώστε να είναι συμβατό με το λειτουργικό σύστημα Mac OS X.

5.2 Μελλοντικές εργασίες

Προκειμένου να επεκτείνουμε την παρούσα εργασία, θα μπορούσαμε να κινηθούμε σε δύο βασικούς άξονες. Ο πρώτος αφορά στην κατασκευή εργαλείων που πραγματοποιούν μερικό ταίριασμα συμβολοσειρών για διαφορετικές εφαρμογές και αρχιτεκτονικές, και ο δεύτερος αφορά στην επέκταση των μηχανισμών που χρησιμοποιούνται στην εφαρμογή Ιστού DIANA.
Ως προς την πρώτη επιλογή, μία επιθυμητή επέκταση θα ήταν η κατασκευή udf συναρτήσεων οι οποίες πραγματοποιούν μερικό ταίριασμα συμβολοσειρών και για άλλες βάσεις δεδομένων, όπως η postgresql και η oracle database. Οι συγκεκριμένες βάσεις διαθέτουν ορισμένες τέτοιες udf , ωστόσο δεν είναι τόσο στραμμένες σε τεχνικές ευρετηρίων όσο συντακτικής απόστασης. Παράλληλα, θα μπορούσαν να αναπτυχθούν εργαλεία προκειμένου να αυτοματοποιείται η διαδικασία κατασκευής ευρετηρίων για ένα δεδομένο πίνακα ή μία βάση δεδομένων, ανεξαρτήτως συστήματος βάσεων δεδομένων και λειτουργικού συστήματος. Αυτή η λύση ενδεχομένως να παρέχεται από τη βάση ή από κάποιο άλλο πρόγραμμα κατασκευασμένο σε γλώσσα ανεξάρτητη αρχιτεκτονικής, όπως η Perl ή η Java.
Κινούμενοι προς τη δεύτερη κατεύθυνση, θα μπορούσαμε να κατασκευάσουμε ένα μηχανισμό συλλογής στατιστικών στοιχείων από προηγούμενες αναζητήσεις χρηστών, προκειμένου να παρέχουμε προτάσεις στο χρήστη βάσει ενός στατιστικού-πιθανοτικού μοντέλου. Η λύση αυτή θα μπορούσε να υλοποιηθεί τροποποιώντας απλά τη μετρική η οποία χρησιμοποιείται στα ευρετήρια της βάσης δεδομένων για την ταξινόμηση των gram, ή με την διατήρηση στοιχείων σε ξεχωριστούς πίνακες, προκειμένου αυτά να είναι προσβάσιμα από οποιαδήποτε μηχανή πραγματοποιεί την αναζήτηση.
Αντίστοιχα, θα μπορούσε να υιοθετηθεί στο σύστημα DIANA micro-T ένας μηχανισμός που αξιοποιεί v-grams, ευρετήρια δηλαδή που χρησιμοποιούν grams μεταβλητού και όχι σταθερού μήκους. Με αυτό τον τρόπο, μπορούμε να μειώσουμε το μέγεθος του καταλόγου και το χρόνο εκτέλεσης . Επίσης, είναι ιδιαίτερα εύκολο να υιοθετηθεί το σχήμα από πολλούς και διαφορετικούς αλγορίθμους αναζήτησης. Η επέκταση αυτή θα επέλυε και το πρόβλημα της απόδοσης των gram σε εγγραφές μικρότερου μήκους, όπου η μέθοδος των n-gram υστερεί έναντι της συντακτικής απόστασης.

6

Βιβλιογραφία

[1] E. Sutinen and J. Tarhio. On Using q-Grams Locations in Approximate String Matching. In ESA, pages 327–340, 1995.

[2] Mysql resources from: http://dev.mysql.com/doc/refman/5.0/

[3] String matching information from: http://en.wikipedia.org/wiki/String_matching

[4]Bitap algorithm information from: http://en.wikipedia.org/wiki/Bitap_algorithm

[5] Yii framework resources from : http://www.yiiframework.com/

[6] Information retrieval resources from :http://en.wikipedia.org/wiki/information_retrieval

[7] Suffix Trees resources from: http://en.wikipedia.org/wiki/Suffix_tree

[8] Vgram: Improving performance of approximate queries on string collections using variable-length grams, Chen Li, Bin Wan, Xiaochun Yang.

[9] Efficient Approximate Search on String Collections, Chen Li, Marios Hadjieleftheriou

[10] Fast Indexes and Algorithms for Set Similarity Selection Queries, Marios Hadjieleftheriou, Amit Chandel , Nick Koudas , Divesh Srivastava

[11] Text Processing in Linux; N-grams, Massimo Poesio, 2003

[12] Efficient set joins on similarity predicates. Sunita Sarawagi, Alok Kirpal. SIGMOD 2004

[13] Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. Chuan Xiao, Wei Wang, Xuemin Lin. PVLDB 2008

[14] Efficient similarity joins for near duplicate detection. Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu. WWW 2008

[15] Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently. Xiaochun Yang, Bin Wang, and Chen Li. SIGMOD 2008

[16] SEPIA: Estimating Selectivities of Approximate String Predicates in Large Databases. Liang Jin, Chen Li, Rares Vernica. VLDBJ08

[17] Record linkage: Similarity measures and algorithms. Nick Koudas, Sunita Sarawagi, Divesh Srivastava. SIGMOD 2006.

[18] Efficient Merging and Filtering Algorithms for Approximate String Searches. Chen Li, Jiaheng Lu, and Yiming Lu. ICDE 2008.
[19] Extending Q-Grams to Estimate Selectivity of String Matching with Low Edit Distance. Hongrae Lee, Raymond T. Ng, Kyuseok Shim. VLDB 2007
[20] VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams, Chen Li, Bin Wang, and Xiaochun Yang. VLDB 2007
[21] Estimating the selectivity of approximate string queries. Arturas Mazeika, Michael H. Böhlen, Nick Koudas, Divesh Srivastava. ACM TODS 2007
[22] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison Wesley, May 1999.
[23] G. Navarro, "A guided tour to approximate string matching," ACM Computing Surveys, vol. 33, no. 1, pp. 31–88, 2001.
[24] E. Sutinen and J. Tarhio. Filtration with q-Samples in Approximate String Matching. In CPM, 1996.
[25] E. Ukkonen. Approximae String Matching with q-Grams and Maximal Matching. Theor. Comut. Sci., 1:191–211, 1992.
[26] I. H. Witten, A. Moffat, and T. C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann Publishing, 1999.
[27] Php information on: http://www.php.net/
[28] jQuery information on: http://jquery.com/