ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Μέθοδοι Προσπέλασης για την Επεξεργασία Μεγάλων Βιολογικών Βάσεων Δεδομένων

Ανδρουλάκης Ανδρέας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος
Βασιλακόπουλος Μιχάλης
Αναπληρωτής Καθηγητής

Λαμία, 2012

Περίληψη

Ο όγκος των βιολογικών δεδομένων που παράγονται, αυξάνεται συνεχώς. Για την επεξεργασία αυτών των δεδομένων, είναι απαραίτητη η εφαρμογή ιδιαίτερα αποδοτικών μεθόδων. Μελετήσαμε λοιπόν κάποιες μεθόδους που χρησιμοποιούνται για την αναζήτηση μέσα σε μεγάλες βιολογικές βάσεις δεδομένων. Παρατηρήσαμε πως οι μέθοδοι που κάνουν χρήση κάποιου ευρετηρίου, παρουσιάζουν πολύ καλή απόδοση. Ο miBLAST είναι ένας αλγόριθμος που συγκρίνει ένα σύνολο νουκλεοτιδικών ακολουθιών, έναντι μιας μεγάλης βάσης δεδομένων που περιέχει τέτοιου είδους ακολουθίες. Ο miBLAST καταφέρνει να εκτελεί αυτά τα ερωτήματα σε σχετικά μικρό χρόνο, χρησιμοποιώντας ένα ευρετήριο για την βάση, το οποίο αποθηκεύεται στον δίσκο.

Επειδή το ευρετήριο που παράγεται είναι συνήθως μεγάλο και δύσχρηστο, αποφασίσαμε να μετατρέψαμε τον miBLAST, έτσι ώστε να χειρίζεται ένα συμπιεσμένο ευρετήριο. Εφαρμόσαμε δηλαδή κάποιους αλγόριθμους συμπίεσης, οι οποίοι ταιριάζουν στην υπάρχουσα δομή του ευρετηρίου. Με αυτόν τον τρόπο καταφέρνουμε να μειώσουμε κατά πολύ τον χώρο που καταλαμβάνει το ευρετήριο στο δίσκο, άλλα και να επιταχύνουμε την κατασκευή του. Επίσης επιτυγχάνουμε την περεταίρω βελτίωση της απόδοσης του miBLAST, στην εκτέλεση ερωτημάτων σύγκρισης πολλαπλών ακολουθιών.

Abstract

The volume of biological data is growing explosively. To analyze these large datasets, there is a pressing and growing need for extremely efficient computational methods. We study some methods which are designed for querying large databases. We found that index-based methods are very efficient. miBLAST algorithm evaluates a batch of nucleotide sequence queries against a large sequence database. miBLAST speeds up the execution of such queries, by employing a disk-based index for the database.

Since the generating index often is large; we decided to modify miBLAST so he uses a compressed index. So we employ some compressing algorithms which suit to the index structure. In such way we manage to reduce the index storage space and we succeed to speed up its construction. Also we succeed to improve furthermore the miBLAST efficiency in evaluating batch queries.

Περιεχόμενα

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΓΡΑΦΗΜΑΤΩΝ

Εισαγωγή

Τις τελευταίες δεκαετίες, η πρόοδος στις βίο-επιστήμες έχει φέρει σαν συνεπακόλουθο την δημιουργία τεράστιου όγκου βιολογικών δεδομένων. Μέχρι και σήμερα, το πλήθος των δεδομένων αυξάνεται ραγδαία, και σε κάποιες περιπτώσεις με ρυθμό μεγαλύτερο και από το νόμο του Moore. Για να ανταπεξέλθουμε στην ραγδαία αυτή αύξηση των δεδομένων, παρουσιάζεται η επιτακτική ανάγκη προσφυγής σε ιδιαίτερα αποδοτικές μεθόδους για την επεξεργασία του μεγάλου αυτού συνόλου βιολογικών δεδομένων. Ωστόσο, οι περισσότερες από τις υπάρχουσες μεθόδους που έχουν δημιουργηθεί για τον χειρισμό τέτοιου είδους δεδομένων είναι υπολογιστικά απαιτητικές και παρουσιάζουν δυσκολίες στην επεξεργασία δεδομένων μεγάλου όγκου.

Μια λογική σκέψη για τον χειρισμό των συνεχώς αυξανόμενων βιολογικών δεδομένων είναι η εφαρμογή μεθόδων που βασίζονται στην δημιουργία ευρετηρίου. Τέτοιες μέθοδοι χρησιμοποιούνται ήδη σε διάφορες βιολογικές εφαρμογές. Εξ αιτίας της αποτελεσματικότητας του στον εντοπισμό ομοιοτήτων μεταξύ συμβολοσειρών, το q-gram ευρετήριο είναι συνηθισμένο να χρησιμοποιείται σε εφαρμογές αναζήτησης ακολουθιών [1,2]. Ωστόσο, σε αυτές τις εφαρμογές, το q-gram ευρετήριο εφαρμόζεται με τον κλασσικό τρόπο της αποθήκευσης του στην κύρια μνήμη. Η χρήση ενός ευρετηρίου κύριας μνήμης είναι απλή στην υλοποίηση, αλλά παρουσιάζει ένα βασικό πρόβλημα ότι δεν μπορεί να χειριστεί μεγάλα σύνολα δεδομένων, εξαιτίας του περιορισμού στο μέγεθος της μνήμης. Η εναλλακτική λύση είναι η εφαρμογή ευρετηρίου που αποθηκεύεται στον σκληρό δίσκο.

Μια συνήθης λειτουργία στις σύγχρονες εφαρμογές βιοπληροφορικής είναι η σύγκριση ενός συνόλου νουκλεοτιδικών ακολουθιών, έναντι μίας μεγάλης βάσης δεδομένων με ακολουθίες. Όταν το πλήθος των ακολουθιών του ερωτήματος είναι μεγάλο, ο χρόνος εκτέλεσης αυτής της λειτουργίας αυξάνεται αρκετά. Ο miBLAST είναι ένας αποδοτικός και προσαρμοσμένος αλγόριθμος για την αναζήτηση των ακολουθιών μιας βάσης δεδομένων, ο οποίος προσπαθεί να λύσει αυτό το πρόβλημα που παρουσιάζεται. Για αυτόν τον λόγο θα δώσουμε ιδιαίτερη έμφαση στον συγκεκριμένο αλγόριθμο. Ο miBLAST, κατά βάση, χρησιμοποιεί ένα q-gram ευρετήριο που αποθηκεύεται στο δίσκο και μια τεχνική συνένωσης ευρετηρίων, με σκοπό την αποδοτική εύρεση ομοιοτήτων μεταξύ των ακολουθιών του ερωτήματος

και των ακολουθιών της βάσης. Με την τεχνική αυτή της κατασκευής ευρετηρίων για την βάση και το ερώτημα, ο miBLAST επιτυγχάνει σημαντικά καλύτερη απόδοση σε σχέση με άλλες υπάρχουσες μεθόδους.

Ο στόχος μας στα πλαίσια της πτυχιακής είναι η περεταίρω βελτίωση της απόδοσης του miBLAST. Για το επιτύχουμε αυτό, επιλέξαμε να τροποποιήσουμε τον κώδικα του miBLAST έτσι ώστε να δημιουργεί συμπιεσμένο ευρετήριο για την βάση, όπως και να μπορεί να το αναγνώσει επίσης. Ο τρόπος για να μπορέσουμε να έχουμε ένα συμπιεσμένο ευρετήριο είναι εφαρμόζοντας κάποιους αλγόριθμους κωδικοποίησης ακέραιων αριθμών. Λαμβάνοντας υπόψη τα χαρακτηρίστηκα της δομής του q-gram ευρετηρίου που χρησιμοποιεί ο miBLAST, επιλέξαμε τον αλγόριθμο διαφορικής κωδικοποίησης (delta code) σε συνδυασμό με τους αλγόριθμους Elias gamma code και Elias delta code.

Με αυτό τον τρόπο θέλουμε να επιτύχουμε, κατά κύριο λόγο, την μείωση του χώρου του καταλαμβάνει το ευρετήριο στον δίσκο. Επίσης ευελπιστούμε ότι θα έχουμε όφελος και στον χρόνο εκτέλεσης του miBLAST, επωφελούμενοι από το μικρό μέγεθος του ευρετηρίου. Έτσι, μπορούμε να επιταχύνουμε την διαδικασία σύγκρισης πολλών ακολουθιών, έναντι μίας μεγάλης βάσης δεδομένων.

Στο Κεφάλαιο 1, θα παρουσιάσουμε αρχικά κάποιες μεθόδους που μπορούν να χειριστούν βιολογικές βάσεις δεδομένων μεγάλου όγκου. Οι περισσότερες από αυτές χρησιμοποιούν την δομή ευρετηρίου, όπως και ο miBLAST, στον οποίο θα κάνουμε ιδιαίτερη αναφορά. Στο Κεφάλαιο 2, θα περιγράψουμε το πώς εφαρμόζουμε την συμπίεση για τον ευρετήριο του miBLAST και στο Κεφάλαιο 3 θα παρουσιάσουμε τα αποτέλεσμα που επιτυγχάνουμε με την μέθοδο αυτή.

ΚΕΦΑΛΑΙΟ 1

1 Υπάρχοντες Αλγόριθμοι για μεγάλες Βιολογικές Βάσεις Δεδομένων

1.1 BLAST (Basic Local Alignment Search Tool)

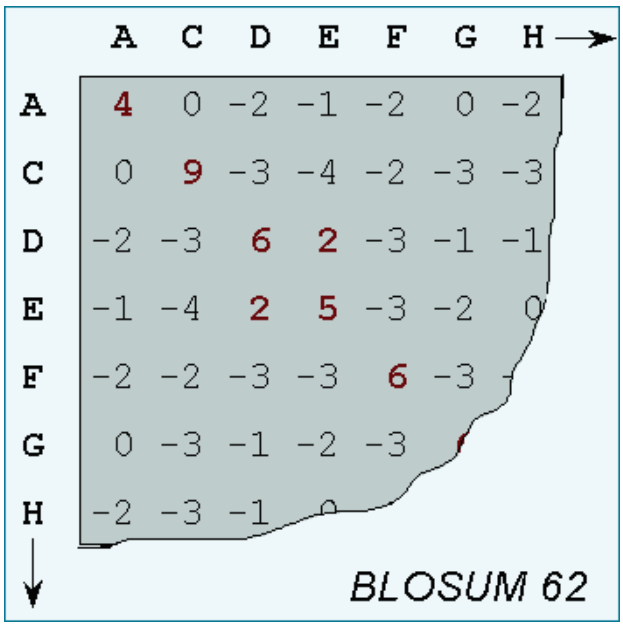
Όταν αναζητούμε ομολογίες, θέλουμε να βρούμε ακολουθίες οι οποίες έχουν παρόμοια λειτουργία, χωρίς να είναι απαραίτητα εξελικτικά συσχετισμένες. Ο αλγόριθμος BLAST [1] βρίσκει περιοχές τοπικής ομοιότητας μεταξύ των βιολογικών ακολουθιών. Η τοπική στοίχιση εντοπίζει μικρά όμοια τμήματα μεταξύ δύο ακολουθιών, τα οποία δεν είναι απαραίτητα στην ίδια θέση και στις δύο ακολουθίες και μπορεί να εμφανίζονται πολλές φορές. Η τοπική στοίχιση είναι χρήσιμη για την εύρεση όμοιων μοτίβων σε δύο ασυσχέτιστες ακολουθίες, ή για την στοίχιση δύο σχετικών ακολουθιών οι οποίες έχουν υποστεί μη-τοπικές αλλαγές. Οι μετρήσεις τοπικής ομοιότητας προτιμούνται ιδιαίτερα για αναζητήσεις σε βάσεις δεδομένων.

Ο αλγόριθμος BLAST χρησιμοποιείται για τη σύγκριση νουκλεοτιδικών ή πρωτεϊνικών ακολουθιών, με μία βάση δεδομένων που περιέχει μεγάλο πλήθος ακολουθιών. Το πλεονέκτημα του BLAST είναι ότι μπορεί να εκτελέσει τέτοιου είδους εργασίες αρκετά γρήγορα, αντίθετα με προγενέστερες μεθόδους οι οποίες ήταν χρονοβόρες. Για αυτό το λόγο αποτελεί ένα από τα πιο δημοφιλή προγράμματα της βιοπληροφορικής. Υπάρχουν φυσικά πιο πρόσφατοι αλγόριθμοι οι οποίοι είναι γρηγορότεροι, ωστόσο όλοι αυτοί βασίζονται στην ευριστική (heuristic) μέθοδο του BLAST.

Όταν κάνουμε στοίχιση δύο ή περισσότερων ακολουθιών, έχουμε σαν σκοπό την αναγνώριση του βιολογικού συσχετισμού μεταξύ των ακολουθιών. Αυτός μπορεί να είναι είτε εξελικτικός ή λειτουργικός. Ειδικά κατά την εξέλιξη, οι ακολουθίες τροποποιούνται με μεταλλάξεις, εισαγωγές και διαγραφές. Οι εξελικτικές σχέσεις μεταξύ δύο ή περισσότερων ακολουθιών μπορούν να ανακαλυφτούν με την στοίχιση αυτών των ακολουθιών. Η σχέση μεταξύ των ακολουθιών μπορεί να μετρηθεί με την χρήση του όρου βαθμός «ομοιότητας» ή με την «απόσταση».

Έχοντας ένα ζευγάρι στοιχισμένων ακολουθιών, θέλουμε να προσδιορίσουμε μια βαθμολογία (score) για την στοίχιση, η οποία θα μετρά την πιθανότητα οι

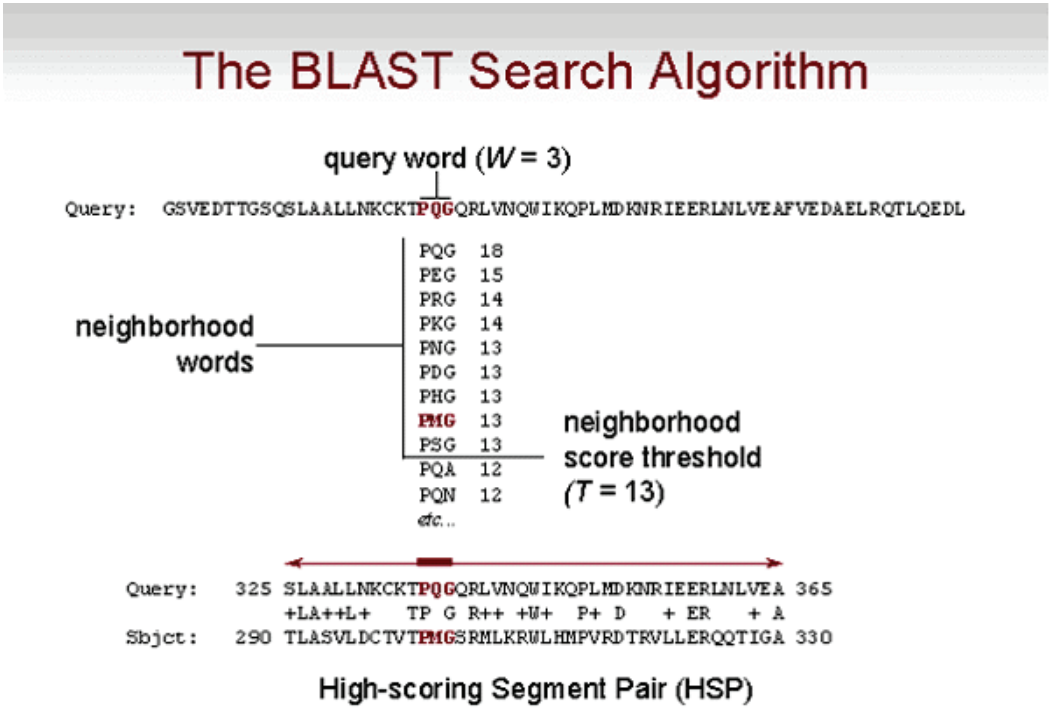
ακολουθίες να συσχετίζονται έναντι της πιθανότητας να είναι άσχετες μεταξύ τους. Αυτό επιτυγχάνεται χρησιμοποιώντας στατιστικά μοντέλα που υπολογίζουν την πιθανότητα για κάθε μία από δύο περιπτώσεις, ώστε να μπορούν έπειτα να συγκριθούν. Η μέτρηση του βαθμού ομοιότητας γίνεται με την βοήθεια κάποιων πινάκων υποκατάστασης (substitution matrix). Για την σύγκριση ακολουθιών με αμινοξέα υπάρχουν οι πίνακες PAM (Point Accepted Mutation), BLOSUM (BLOcks SUbstitution Matrix), οι οποίοι έχουν μέγεθος 20x20 και ορίζουν μια βαθμολογία για κάθε αντιστοίχιση δυο αμινοξέων. Σε γενικές γραμμές εκφράζουν την εξελικτική σχέση των 20 αμινοξέων. Για την σύγκριση των ακολουθιών DNA, όταν υπάρχει όμοιο νουκλεοτίδιο θέτουμε βαθμό +5 και στην αντίθετη περίπτωση -4.



Γράφημα 1.1: Η μορφή του πίνακα υποκατάστασης BLOSUM 62

Ο αλγόριθμος του BLAST αναζητά όμοιες λέξεις μεταξύ του ερωτήματος και των ακολουθιών της βάσης. Έτσι ορίζονται μικρά τμήματα με μήκος W, όπου συνήθως το W είναι 3 έως 5 αμινοξέα για πρωτεϊνικές ακολουθίες και 11 νουκλεοτίδια για ακολουθίες DNA. Αρχικά διαβάζονται οι λέξεις που περιέχει το ερώτημα, οι οποίες και αποθηκεύονται σε ένα πίνακα. Έπειτα ψάχνει στην βάση για όμοιες λέξεις με το ερώτημα, που δίνουν αποτελέσματα μεγαλύτερο ή ίσο με T , όταν συγκρίνονται μεταξύ τους χρησιμοποιώντας έναν δεδομένο πίνακα υποκαταστάσεων. Οι λέξεις της βάσης που έχουν πάρει βαθμολογία μεγαλύτερη ή ίση με το κατώφλι T, αποτελούν τις αρχικές στοιχίσεις, οι οποίες επεκτείνονται και

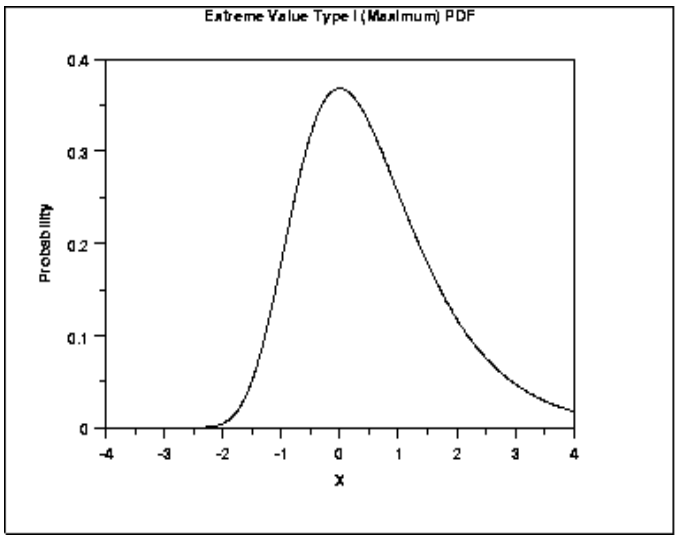
προς τις δύο κατευθύνσεις έως ότου να παραχθούν στοιχίσεις που να υπερβαίνουν την μέγιστη δυνατή βαθμολογία. Κάθε στοίχιση που παράγεται με αυτόν τον τρόπο χαρακτηρίζεται HSP (High-scoring Segment Pair). Η ελάχιστη βαθμολογία (S) για τις αναφερόμενες HSPs, όπως και η μέγιστη τιμή e-value (E) των HSPs, ορίζονται από τον χρήστη. Οι HSPs που πληρούν αυτά τα κριτήρια θα εξαχθούν από το BLAST.



Γράφημα 1.2: Παράδειγμα εφαρμογής του αλγόριθμου του BLAST

Κάνοντας μία αναζήτηση με τον BLAST, λαμβάνουμε τις στοιχίσεις των ακολουθιών μαζί τον βαθμό ομοιότητας τους. Ωστόσο, ένα σημαντικό στοιχείο είναι να γνωρίζουμε την στατιστική σημαντικότητα αυτών των βαθμολογιών. Η στατιστική σημαντικότητα βασίζεται στην πιθανότητα δύο τυχαίες ακολουθίες να έχουν την ίδια βαθμολογία με δύο στοιχισμένες ακολουθίες που έχουν λάβει βαθμό ομοιότητας μεγαλύτερο από την αναμενόμενη τιμή (e-value). Με άλλα λόγια, όταν στοιχίζονται δύο ακολουθίες, ουσιαστικά ελέγχουμε εάν αυτή η στοίχιση προέκυψε τυχαία, ή πραγματικά αυτές οι δύο ακολουθίες είναι ομόλογες. Δηλαδή μας ενδιαφέρει το πως μπορούμε να διαχωρίσουμε «τυχαία» ευρήματα από «σημαντικά».

Όταν κάνουμε ένα είδος σύγκρισης μεταξύ ενός αντικειμένου και ενός συνόλου (βάση δεδομένων) από αντικείμενα, κάθε σύγκριση μας δίνει και μία βαθμολογία. Οι βαθμολογίες του αποτελέσματος ακολουθούν μία κανονική κατανομή (για ολικές στοιχίσεις), όπως συμβαίνει δηλαδή σε ένα σύνολο πολλών τυχαίων τιμών, οι οποίες κατανέμονται ομαλά. Αντίθετα, εάν για κάθε σύγκριση κρατάμε μόνο την μέγιστη βαθμολογία, τα αποτελέσματα θα ακολουθούν ασυμπτωτικά την κατανομή των ακραίων τιμών (Extreme Value Distribution-EVD). Στις στοιχίσεις ακολουθιών, σε κάθε σύγκριση εξάγουμε μόνο την ακριβής στοίχιση, δηλαδή την μέγιστη βαθμολογία, ως εκ τούτου τα αποτελέσματα των συγκρίσεων μεταξύ των ακολουθιών κατανέμονται σύμφωνα με την EVD.



Γράφημα 1.3: Κατανομή των ακραίων τιμών (EVD)

Σε αύτη την κατανομή, ορίζοντας ένα κατώφλι ομοιότητας, μπορεί να υπολογιστεί η τιμή p-value. p-value ορίζεται ως η πιθανότητα εμφάνισης μίας στοίχισης με μεγαλύτερο ή ίσο βαθμό ομοιότητας, δεδομένου ότι ισχύει η μηδενική υπόθεση, δηλαδή πόσο πιθανό είναι να προκύψει τυχαία αυτή η ομοιότητα μεταξύ των ακολουθιών. Η p-value μας προδιαθέτει πως περίπου ο BLAST παράγει μία στοίχιση μεταξύ δύο ακολουθιών που δεν σχετίζονται λειτουργικά. Όσο μεγαλύτερη είναι η τιμή της p-value, τόσο λιγότερο σημαντικό είναι το αποτέλεσμα.

Όταν πραγματοποιείται μία αναζήτηση στην βάση δεδομένων, στην ουσία γίνονται πολλαπλοί έλεγχοι. Η p-value πρέπει λοιπόν να προσαρμόζεται στους πολλαπλούς αυτούς ελέγχους. Έτσι η πιθανότητα πρέπει να καθορίζεται από το

μέγεθος τις βάσης. Αυτό αντιπροσωπεύει η E-value. Η τιμή E-value είναι μία εκδοχή της p-value που είναι προσαρμοσμένη στους πολλαπλούς ελέγχους. Η E-value υπολογίζεται πολλαπλασιάζοντας την p-value με το μέγεθος της βάσης. Η E-value είναι ο αναμενόμενος αριθμός εμφανίσεων περιοχών με ένα συγκεκριμένο βαθμό (score) ομοιότητας, σε μία οποιαδήποτε βάση με δεδομένο μέγεθος. Για παράδειγμα, για p-value ίσο με 0.001 και μία βάση δεδομένων με 1.000.000 ακολουθίες, η αντίστοιχη τιμή E-value είναι 0.001 x 1,000,000 = 1,000.

Η τιμή E-value μπορεί να ερμηνευτεί ως ο πιθανός αριθμός των ξεκάθαρων στοιχίσεων (HSPs) που αναμένεται να εντοπίσουμε στην βάση, οι οποίες θα έχουν βαθμολογία μεγαλύτερη ή ίση με το κατώφλι που έχει οριστεί. Όσο μεγαλύτερη είναι η τιμή της E-value, τόσο λιγότερο στατιστικά σημαντικό χαρακτηρίζεται το αποτέλεσμα της στοίχισης. Για παράδειγμα, η E-value με τιμή 2 σημαίνει ότι αναμένεται να βρούμε 2 τυχαίες ακολουθίες που να έχουν ένα συγκεκριμένο βαθμό ομοιότητας. Έτσι ο BLAST χρησιμοποιεί μόνο την E-value, η οποία γίνεται πιο εύκολα κατανοητή από τους βιολόγους και επιπλέον παίρνει τιμές από 0 έως τον αριθμό των ακολουθιών στη βάση.

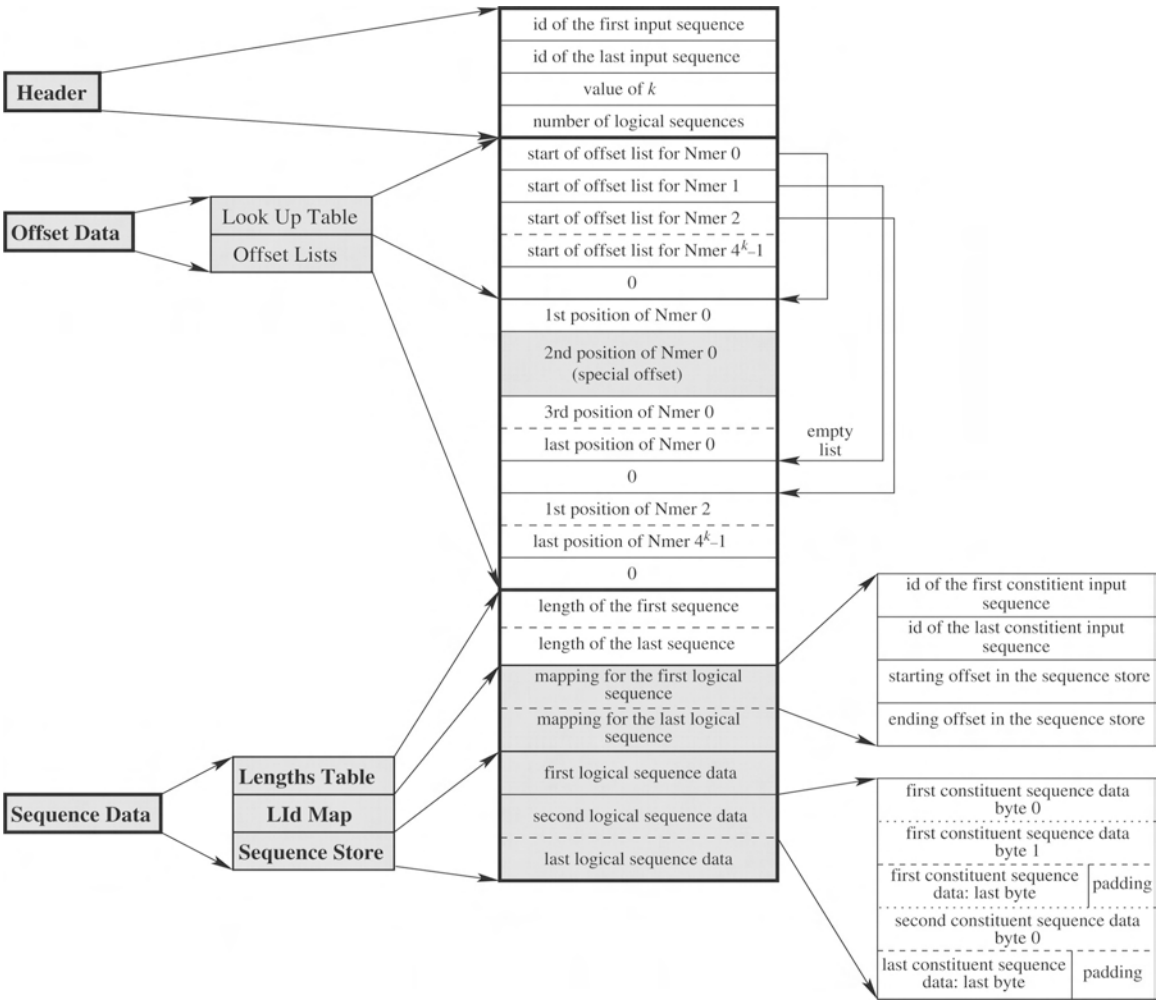
Συνοψίζοντας ο BLAST αποτελεί ένα αρκετά γρήγορο πρόγραμμα για την αναζήτηση μέσα σε βάσεις δεδομένων, το οποίο έχει επίσης το πλεονέκτημα της επιδεκτικότητας στην μαθηματική ανάλυση. Ο αλγόριθμος του BLAST χρησιμοποιείται συχνά σαν τμήμα άλλων αλγορίθμων που εκτελούν προσεγγιστική στοίχιση ακολουθιών.

1.2 MegaBLAST με χρήση ευρετηρίου για την βάση

Ο MegaBLAST [3] είναι ένας αλγόριθμος παρόμοιος με τον BLAST, που χρησιμοποιείται για την στοίχιση νουκλεοτιδικών ακολουθιών οι οποίες διαφέρουν σε μικρό βαθμό. Ο MegaBLAST καταφέρνει να είναι πιο γρήγορος από τον BLAST, ωστόσο επιτυγχάνει μικρότερη ευαισθησία στα αποτελέσματα. Αυτό συμβαίνει διότι χρησιμοποιεί μεγαλύτερο μέγεθος λέξης για την αρχική στοίχιση, σε σχέση με τον BLAST. Το συνηθέστερο μέγεθος λέξης για τον BLAST είναι 11, ενώ για τον MegaBLAST είναι 28. Ο MegaBLAST είναι κατάλληλος για την σύγκριση με ενός μεγάλου πλήθους ακολουθιών, με μία μεγάλη βάση δεδομένων.

Μια καινούργια λειτουργία που έχει προστεθεί στο πρόγραμμα του MegaBLAST, είναι η χρήση ευρετηρίου για την βάση δεδομένων [4], έτσι ώστε να εκτελείται γρηγορότερα η αναζήτηση. Ο κλασσικός BLAST χρησιμοποιεί ένα πίνακα αναζήτησης με τις λέξεις του ερωτήματος, ωστόσο δεν κάνει κάποιου είδους προ-επεξεργασία της βάσης δεδομένων. Με την κατασκευή ευρετηρίου για την βάση, επιτυγχάνεται ο γρήγορος εντοπισμός των όμοιων λέξεων με το ερώτημα.

Το ευρετήριο που εφαρμόζει ο MegaBLAST για την βάση, περιέχει τμήματα ακολουθιών μήκους k νουκλεοτιδίων. Εκτός από το μήκος k, δύο ακόμα σημαντικές παράμετροι είναι το ελάχιστο μήκος των όμοιων λέξεων w ($\geq k$) και το βήμα s που χρησιμοποιείται για την μετακίνηση μέσα στην βάση. Για να εξασφαλιστεί ότι θα βρεθούν όλες οι όμοιες λέξεις μήκους w μεταξύ του ερωτήματος και της βάσης, χρησιμοποιείται η σχέση $s=w-k+1$.



Γράφημα 1.4: Αναπαράσταση της δομής του ευρετηρίου για την βάση δεδομένων

Το ευρετήριο της βάσης δεδομένων αποτελείται από ένα πλήθος αρχείων που ονομάζονται τομείς του ευρετηρίου, όπου το καθένα αντιστοιχεί σε ένα εύρος συνεχόμενων ακολουθιών της βάσης. Κάθε αρχείο του ευρετηρίου περιέχει τρία τμήμα, όπως φαίνεται στο Γράφημα 1.4. Το πλήθος των νουκλεοτιδίων που περιέχονται στην βάση συμβολίζεται με *n*. Ο υπολογισμός του αριθμού των bytes που απαιτούνται για την αποθήκευση του ευρετηρίου, γίνεται με τον εξής τύπο:

$$2 \times 4^{k+1} + \frac{n}{4} + \frac{4n}{\min\left(s, \frac{k-1}{2}\right)}$$

Για παράδειγμα, για μία βάση μεγέθους 1 GB και με τις προεπιλεγμένες τιμές k=12, s=5, το εκτιμώμενο μέγεθος του ευρετηρίου είναι ~1,175 GB.

Η επικεφαλίδα κάθε τομέα του ευρετηρίου πρωτίστως αποθηκεύει το εύρος των ακολουθιών της βάσης, στις οποίες αναφέρεται αυτός ο τομέας του ευρετηρίου. Στο τμήμα της επικεφαλίδας αποθηκεύονται επίσης κάποιες βοηθητικές τιμές για το ευρετήριο.

Ο αλγόριθμος αναζήτησης των αρχικών στοιχίσεων δουλεύει με *λογικές ακολουθίες,* οι οποίες έχουν παρόμοιο μήκος. Έτσι μπορεί να γίνεται πιο αποτελεσματικά η αντιστοίχιση των νουκλεοτιδικών αλληλουχιών μήκους *k,* με αυτές τις ακολουθίες. Στο τμήμα του ευρετηρίου με τα δεδομένα του ακολουθιών, αποθηκεύονται οι αντιστοιχίες των λογικών ακολουθιών με τις ακολουθίες της βάσης δεδομένων. Στο τμήμα με τα δεδομένα θέσης (*Offset data*), περιέχονται οι θέσεις των αλληλουχιών μήκους *k* μέσα στην βάση δεδομένων. Δηλαδή για κάθε διαφορετική αλληλουχία (λέξη), αποθηκεύονται οι θέσεις στην βάση όπου αυτή συναντάται.

Έχει αποδειχτεί ότι ο MegaBLAST που χρησιμοποιεί ευρετήριο, στις περισσότερες περιπτώσεις, είναι αρκετά πιο γρήγορος από τον αρχικό MegaBLAST. Καταλαβαίνουμε λοιπόν ότι η προ-επεξεργασία της βάσης δεδομένων μπορεί να επιταχύνει την εκτέλεση ερωτημάτων σύγκρισης.

1.3 ProbeMatch

Νέες τεχνολογίες επεξεργασίας ακολουθιών DNA παίζουν σημαντικό ρόλο στην σύγχρονη βίο-επιστημονική έρευνα. Αυτές οι μέθοδοι υψηλής απόδοσης παράγουν μεγάλο όγκο δεδομένων, τα οποία μπορούν να χρησιμοποιηθούν σε διάφορες εργασίες όπως η δημιουργία νέων γονιδιακών ακολουθιών. Αυτές οι μέθοδοι παράγουν πολύ μεγάλα σύνολα ακολουθιών μικρού μήκους, τα οποία συχνά χρειάζεται να συσχετιστούν με ένα γονιδίωμα, επιτρέποντας λίγα μόνο σφάλματα (*errors*). Τα παραδοσιακά εργαλεία στοίχισης ακολουθιών, όπως ο BLAST, παρέχουν την απαραίτητη λειτουργικότητα, ωστόσο είναι σε απαγορευτικό βαθμό δαπανηρά υπολογιστικά.

Για να αντιμετωπιστεί αυτό το πρόβλημα, αρκετά προγράμματα έχουν αναπτυχθεί [6,7]. Ωστόσο, παρόλο που είναι αρκετά γρήγορα, δεν επιτυγχάνουν ικανοποιητική ευαισθησία, διότι βρίσκουν στοιχίσεις μόνο χωρίς κενά (*ungapped*) και επιτρέπουν λίγες αστοχίες (*mismatches*). Ο ProbeMatch [5] είναι ένας αποδοτικός και με μεγάλη ευαισθησία αλγόριθμος, οποίος εκτελεί στοίχιση ενός μεγάλου συνόλου με μικρές ακολουθίες. Αντίθετα με τις άλλες μεθόδους, βρίσκει στοιχίσεις με ή χωρίς κενά και μέχρι 3 σφάλματα (αστοχίες, εισαγωγές και διαγραφές). Αυτό του δίνει το πλεονέκτημα να μπορεί να εντοπίζει πολλών ειδών μεταλλάξεις. Ο ProbeMatch επιστρέφει τις στοιχίσεις με την μεγαλύτερη βαθμολογία, ορίζοντας κατάλληλα τις «ποινές» για τα κενά και τις αστοχίες.

Η μέθοδος που ακολουθεί ο ProbeMatch είναι η εξής. Δέχεται σαν είσοδο ένα σύνολο με τις ακολουθίες του ερωτήματος και ένα με τις ακολουθίες της βάσης. Επειδή η βάση δεδομένων συχνά είναι μεγάλη και δεν χωράει στην μνήμη, ο ProbeMatch χωρίζει την βάση σε μικρά τμήματα. Φορτώνει κάθε ακολουθία της βάσης και κατασκευάζει ένα q-gram ευρετήριο. Για κάθε ακολουθία του ερωτήματος, ο ProbeMatch αναζητά στο ευρετήριο να βρει τις πιθανές «επιτυχίες» και έπειτα τις επεκτείνει για να δημιουργήσει μεγαλύτερες στοιχίσεις. Αυτή η διαδικασία επαναλαμβάνεται έως ότου όλα τα τμήματα της βάσης να έχουν επεξεργαστεί.

Για να επιταχύνει την στοίχιση, χρησιμοποιεί μία τεχνική φιλτραρίσματος που βασίζεται στο ακόλουθο λήμμα: Εάν δύο ακολουθίες, Q και T, ταιριάζουν έχοντας k σφάλματα και j μη-επικαλυπτώμενα τμήματα έχουν παρθεί από το Q, τότε η ακολουθία T περιέχει τουλάχιστον ένα από αυτά τα τμήματα με το πολύ $\lfloor k/j \rfloor$

σφάλματα. Βάσει αυτού του λήμματος, ο ProbeMatch για να βρει τις ακολουθίες που ταιριάζουν με το πολύ 3 σφάλματα (k = 3), πρώτα, χωρίζει την ακολουθία του ερωτήματος σε δύο τμήματα (j = 2). Έπειτα, βρίσκει τις ακολουθίες τις βάσης που περιέχουν ένα από αυτά τα τμήματα έχοντας 1 σφάλμα ($\lfloor k/j \rfloor$ = 1). Τέλος, τα τμήματα που ταιριάζουν επεκτείνονται έτσι ώστε να ελεγχτεί εάν ολόκληρη η ακολουθία του ερωτήματος και η ακολουθία της βάσης μπορούν να στοιχίσουν με 3 σφάλματα. Το ευρετήριο του ProbeMatch περιέχει q-gram λέξεις με κενά (*gapped*). Για παράδειγμα, ένα 3-gram με κενό είναι μία μη-συνεχόμενη αλληλουχία με μήκος 4, η οποία έχει 3 δεδομένα σύμβολα (γράμματα) και ένα άγνωστο σε μία καθορισμένη θέση.

Υπάρχουν αρκετά πρότυπα για την δημιουργία q-gram με κενό, τα οποία διαφέρουν στην θέση του κενού. Για παράδειγμα, ένα 3-gram με κενό έχει δύο διαφορετικές εκδοχές, ##_# και #_##, όπου # τα σύμβολα που καταγράφονται και _ το σύμβολο που δεν μας ενδιαφέρει.

Η τεχνική q-gram με κενό έχει το πλεονέκτημα ότι παρέχει πιο αποτελεσματικό φιλτράρισμα σε σχέση με το q-gram χωρίς κενά. Για παράδειγμα, υποθέτουμε ότι θέλουμε να βρούμε όλες τις πιθανές ακολουθίες που ταιριάζουν σε μία ακολουθία του ερωτήματος μήκους 18 νουκλεοτιδίων, επιτρέποντας το πολύ μία αστοχία. Αυτές οι ακολουθίες μπορούν να εντοπιστούν χρησιμοποιώντας ένα q-gram χωρίς κενά με μήκους 9 (#########). Ωστόσο, χρησιμοποιώντας το εξής q-gram με ένα κενό, ######_##### (ή #####_######), μπορούμε επίσης να βρούμε τις υποψήφιες ακολουθίες. Χρησιμοποιώντας αυτό το q-gram με κενό απαιτείται να υπάρχουν δύο επιπλέον ίδια σύμβολα σε σχέση με το q-gram χωρίς κενά, γεγονός που οδηγεί σε ένα πιο επιλεκτικό φιλτράρισμα. Η επιλογή του πρότυπου για το q-gram με κενά είναι ιδιαίτερα σημαντική, και καθορίζεται από το μήκος της ακολουθίας του ερωτήματος και τον αριθμό των αστοχιών που επιτρέπονται. Στο παραπάνω παράδειγμα, χρησιμοποιώντας το πρότυπο, ######_#####, μπορούν να εντοπιστούν οι ακολουθίες που έχουν μία αστοχία, αλλά δεν μπορούν να εντοπιστούν οι υποψήφιες ακολουθίες που έχουν μια εισαγωγή ή μια διαγραφή (*indel*). Για να αντιμετωπιστεί αυτό πρέπει να χρησιμοποιηθούν με περισσότερα πρότυπα. Εκτός από το βασικό πρότυπο, ######_#####, χρησιμοποιούνται ακόμα δύο πρότυπα, ######_ _##### and ##########. Αυτά τα πρότυπα προκύπτουν από το βασικό πρότυπο προσθέτοντας και αφαιρώντας ένα κενό. Ακόμα και αν

υπάρχει μία εισαγωγή η μία διαγραφή, τα ζεύγη των ακολουθιών του ερωτήματος και της βάσης που ταιριάζουν, θα έχουν τουλάχιστον ένα κοινό q-gram.

Ο ProbeMatch έχει υλοποιηθεί έτσι ώστε να βρίσκει ακολουθίες που ταιριάζουν επιτρέποντας 3 σφάλματα, για αυτό χωρίζει την ακολουθία του ερωτήματος σε 2 τμήματα. Για κάθε τμήμα μία ακολουθίας του ερωτήματος, πρέπει να εντοπίσει ακολουθίες τις βάσης που ταιριάζουν έχοντας το πολύ ένα σφάλμα, χρησιμοποιώντας την τεχνική των q-grams με κενά που περιγράψαμε παραπάνω. Έπειτα ελέγχει εάν ολόκληρη η ακολουθία του ερωτήματος μπορεί να στοιχηθεί με τις επιλεγμένες ακολουθίες της βάσης έχοντας 3 σφάλματα.

Ο ProbeMatch αποδεικνύει την αποδοτικότητα και την αποτελεσματικότητα του, παράγοντας περισσότερες στοιχίσεις σε λιγότερο χρόνο, σε σχέση με άλλες μεθόδους.

1.4 miBLAST

Σε πολλές εφαρμογές βιοπληροφορικής απαιτείται αναζήτηση σε μία μεγάλη βάση δεδομένων που περιέχει ακολουθίες νουκλεοτιδίων, χρησιμοποιώντας ως ερώτημα ένα σύνολο από νουκλεοτιδικές ακολουθίες. Για παράδειγμα, η σύγκριση της συλλογής ολιγονουκλεοτιδίων Affymetrix με την βάση Unigene, όπου απαιτείται η αναζήτηση χιλιάδων ακολουθιών μέσα στην τελευταία έκδοση της βάσης Unigene. Άλλο παράδειγμα είναι η σύγκριση μιας μικρό-ακολουθίας ενός ζωικού μοντέλου σε σχέση με ένα διαφορετικό ζωικό είδος , αναζητώντας τμήματα των ακολουθιών του μέσα στην βάση με τα ESTs αυτών των ειδών. Τέλος, παρόμοια διαδικασία εφαρμόζεται κατά τον σχεδιασμό μικρών βιβλιοθηκών siRNA, όπου χρειάζεται να επικυρωθεί ότι τα siRNAs παρεμβαίνουν σε ένα μόνο mRNA. Το κοινό χαρακτηριστικό αυτών των εφαρμογών είναι η σύγκριση μιας μεγάλης συλλογής ερωτημάτων έναντι σε μια μεγάλη βάση. Συχνά οι βάσεις, στις οποίες εφαρμόζονται τέτοιες διεργασίες αναζήτησης, ενημερώνονται σε τακτά χρονικά διαστήματα, άρα απαιτείται περιοδικά η επανάληψη αυτής της διαδικασίας.

Ένας τρόπος για την σύγκριση μιας συλλογής ερωτημάτων -ακολουθιών, είναι η εκτέλεση κάθε ερωτήματος ξεχωριστά χρησιμοποιώντας ένα εργαλείο τοπικής στοίχισης όπως ο BLAST. Ωστόσο στην πράξη, όταν πρέπει να χειριστούμε μια

μεγάλη συλλογή ερωτημάτων, αυτή η μέθοδος είναι πολύ δαπανηρή υπολογιστικά. Σαφέστατα, ένα εργαλείο, το οποίο θα μπορεί να επιταχύνει σημαντικά την διαδικασία στοίχισης σε αυτές τις περιπτώσεις, είναι πολύ χρήσιμο. Ένα τέτοιο εργαλείο είναι ο miBLAST.

Έχουν ήδη αναπτυχθεί κάποιες μέθοδοι για την αποτελεσματική ταυτοποίηση πολλαπλών ερωτημάτων, όπως μπορεί να διαπιστωθεί από σχετική έρευνα. Αυτά τα εργαλεία [2,8,9] είναι σχεδιασμένα για εξειδικευμένες βιολογικές εφαρμογές, όπως η στοίχιση των ESTs με το γονιδίωμα παρόμοιων ζωικών ειδών, και επιτυγχάνουν βελτιωμένη απόδοση με την χρησιμοποίηση ευρετηρίου μη-επικαλυπτόμενων q-grams. Παρόλα αυτά, τέτοιες προσεγγίσεις συχνά θυσιάζουν σε ένα ποσοστό την ευαισθησία, έτσι ώστε να πετύχουν καλή απόδοση. Ο MegaBLAST [3] χρησιμοποιεί έναν προνομιακό αλγόριθμο και επιτυγχάνει σε κάποιο βαθμό μεγαλύτερη ταχύτητα από τον απλό BLAST. Εντούτοις, η χρησιμοποίηση του MegaBLAST περιορίζεται σε περιπτώσεις στοίχισης όπου οι ακολουθίες έχουν μεγάλη ομοιότητα και το μέγεθος λέξης είναι μεγάλο. Ο MPBLAST [10] βελτιώνει δραστικά την ταχύτητα του NCBI BLAST και του WU-BLAST με το να συνενώνει τις ακολουθίες του ερωτήματος, το οποίο έχει ως αποτέλεσμα την μείωση του αριθμού των αναζητήσεων που πραγματοποιούνται στην βάση. Ο MPBLAST ουσιαστικά μετατρέπει το σύνολο των ακολουθιών του ερωτήματος σε μία μεγάλη ακολουθία και την εισάγει σαν απλό ερώτημα στον BLAST.

Στον NCBI BLAST έχει προστεθεί μια επιπλέον λειτουργία, που στην ουσία υλοποιεί την τεχνική συνένωσης του MPBLAST για τα ερωτήματα με πολλαπλές ακολουθίες ˙ και παράγει στοιχίσεις όμοιες με την περίπτωση που θέταμε σαν ερώτημα κάθε ακολουθία ξεχωριστά. Ο BLAST++ [11] εκμεταλλεύεται το γεγονός ότι οι ακολουθίες του ερωτήματος έχουν κάποιες μικρές λέξεις κοινές μεταξύ τους και έτσι παράγει κοινά αποτελέσματα για αυτές τις ακολουθίες. Συνεπώς η απόδοση του BLAST++ εξαρτάται άμεσα το επίπεδο ομοιότητας των ακολουθιών του ερωτήματος. Μία άλλη μέθοδος που επιταχύνει τον χειρισμό πολλαπλού ερωτήματος, είναι η χρησιμοποίηση τεχνικών παράλληλης επεξεργασίας με την βοήθεια μιας συστάδας υπολογιστών (cluster) που τρέχουν το πρόγραμμα mpiBLAST [12]. Αυτή η μέθοδος ουσιαστικά παραλληλοποιεί τις αναζητήσεις που πραγματοποιεί ο BLAST, με το να τμηματοποιεί την βάση και στην συνέχεια να αναθέτει την διαδικασία αναζήτησης του κάθε τμήματος της βάσης, σε διαφορετικό

κόμβο του cluster. Το μειονέκτημα αυτής της προσέγγισης είναι ότι απαιτείται η πρόσβαση σε ένα cluster.

Ο miBLAST [13], τον οποίο θα παρουσιάσουμε, αποσκοπεί στην αποδοτική και ταυτόχρονα πρακτική εκτέλεση των ερωτημάτων που αποτελούνται από ένα μεγάλο αριθμό ακολουθιών. Επιπλέον, έχοντας σαν δεδομένο το πόσο δημοφιλής είναι ο BLAST, και την εξοικείωση που έχουν με αυτόν οι χρήστες, αυτό το εργαλείο επιδιώκει να προσομοιάσει τον τρόπο λειτουργίας, άλλα και την λειτουργικότητα του BLAST. Με άλλα λόγια έχει σαν σκοπό να επιτύχει την ίδια ευαισθησία που έχει ο BLAST στην εξαγωγή των αποτελεσμάτων, εφαρμόζοντας επίσης το ίδιο στατιστικό μοντέλο και την ίδια μορφοποίηση των δεδομένων που δίνονται σαν είσοδοι ή παράγονται στην έξοδο.

Ενώ ο miBLAST είναι ένας αλγόριθμος για την στοίχιση των ακολουθιών που αποτελούν το ερώτημα με τις ακολουθίες της βάσης, είναι ιδιαίτερα χρήσιμος στις περιπτώσεις όπου η βάση και το ερώτημα περιέχουν μεγάλο πλήθος ακολουθιών. Τέτοια παραδείγματα βιολογικών εφαρμογών, που μπορούν να εκτελεστούν με την χρήση του miBLAST, είναι η σύγκριση ενός πλήθους ολιγονουκλεοτιδίων, ακολουθιών cDNA, ή ESTs έναντι μιας ογκώδης βάση με ESTs.

Αναγνωρίζοντας ως στόχο την δημιουργία ενός καταλληλότερου αλγορίθμου για την αποδοτική εξυπηρέτηση τέτοιου είδους ερωτημάτων που απαιτούν υψηλό φόρτο εργασίας, είναι απαραίτητο πρώτα να προσδιοριστούν τα αίτια τα οποία κάνουν τον αλγόριθμο του BLAST να μην είναι αποδοτικός σε αυτές τις περιπτώσεις. Ο αλγόριθμος του BLAST ουσιαστικά για κάθε ακολουθία του ερωτήματος, κάνει και μια σάρωση ολόκληρης της βάσης. Σε κάθε σάρωση της βάσης, ο αλγόριθμος ελέγχει κάθε ακολουθία της βάσης θέλοντας να εντοπίσει μια μικρή λέξη που να είναι κοινή ανάμεσα στην ακολουθία της βάσης και του ερωτήματος. Στην περίπτωση που υπάρχει κοινή λέξη, η στοίχιση επεκτείνεται και προς τις δύο κατευθύνσεις της λέξης, έτσι ώστε να παραχθεί μια ολοκληρωμένη στοίχιση. Συνεπώς, για την εκτέλεση ενός ερωτήματος που αποτελείται από v ακολουθίες, με την μέθοδο του BLAST θα χρειαστεί να πραγματοποιηθούν v σαρώσεις της βάσης και θα γίνει σύγκριση κάθε ακολουθίας του ερωτήματος με κάθε ακολουθία της βάσης.

Αυτή η παρατήρηση ώθησε στην δημιουργία του αλγόριθμου του miBLAST, ο οποίος επιταχύνει την εκτέλεση τέτοιου είδους ερωτημάτων. Συνοπτικά, ο αλγόριθμος αυτός εφαρμόζει μια τεχνική φιλτραρίσματος q-gram και συνένωσης

ευρετηρίων, χρησιμοποιώντας δύο ευρετήρια q-gram [14]. Πραγματοποιώντας μια απλή σάρωση των δυο ευρετηρίων, εφαρμόζει την μέθοδο της συνένωσης και υπολογίζει έτσι για κάθε ακολουθία του ερωτήματος, μια λίστα που περιέχει τις ακολουθίες της βάσης, με τις οποίες έχει τουλάχιστον μια κοινή λέξη (word hit). Στην συνέχεια, ο miBLAST εξετάζει μόνο τις ακολουθίες της βάσης που περιέχονται στην λίστα, για να πραγματοποιήσει τις τελικές στοιχίσεις. Στην πράξη μόνο λίγες ακολουθίες της βάσης ταιριάζουν με το ερώτημα, συνεπώς ο miBLAST δεν χρειάζεται να ελέγξει την πλειονότητα των ακολουθιών στην βάση.

Η χρήση του q-gram ευρετηρίου έχει εξεταστεί και σε άλλες εφαρμογές που ασχολούνται με την αναζήτηση ακολουθιών [2,8,9]. Ωστόσο, αυτές οι εφαρμογές έχουν περιορισμούς από το μέγεθος της βάσης ή από το μέγεθος της κύριας μνήμης, δεδομένου ότι το ευρετήριο πρέπει να βρίσκεται εξ ολοκλήρου στην κύρια μνήμη του συστήματος. Αντίθετα, στην εφαρμογή που μελετάμε εδώ, τα ευρετήρια που χρησιμοποιούνται αποθηκεύονται στον σκληρό δίσκο και έτσι υπάρχει η δυνατότητα να ανταπεξέλθει και σε μεγάλα σύνολα δεδομένων.

Ο miBLAST έχει δοκιμαστεί σε αρκετές περιπτώσεις ερωτημάτων με πολλαπλές ακολουθίες, όπως για παράδειγμα η σύγκριση των χαρακτηριστικών ακολουθιών της συλλογής Affymetrix (Human Genome U133A Set) έναντι της βάσης Unigene. Η συλλογή Affymetrix περιέχει περίπου 250 000 ακολουθίες, με την κάθε μια να έχει κατά μέσο όρο μήκος 25 νουκλεοτίδια. Κάθε ακολουθία της συλλογής πρέπει να συγκριθεί με το σύνολο δεδομένων της UniGene για τον Homo sapiens, που περιέχει 5.064.621 ακολουθίες και περίπου 3,19 gigabases (στην συγκεκριμένη έκδοση). Χρησιμοποιώντας τον NCBI BLAST με τις προκαθορισμένες παραμέτρους (με μέγεθος λέξης 11), για να εκτελεστεί αυτή η εργασία σε ένα κοινό μηχάνημα απαιτούνται 27,27 μέρες (9,50 δευτερόλεπτα για κάθε ακολουθία-ερώτημα). Αντίθετα, ο miBLAST εκτελεί την παραπάνω διεργασία, στο ίδιο μηχάνημα, σε 1,26 μέρες (0,44 δευτερόλεπτα για κάθε ακολουθία-ερώτημα). Το συνολικό κέρδος που επιτυγχάνει ο miBLAST σε αυτήν την περίπτωση, είναι η επιτάχυνση της διαδικασίας κατά 22 φορές. Ωστόσο, ανάλογα με την επιλογή των παραμέτρων, ο miBLAST μπορεί να επιτύχει ακόμα καλύτερα αποτελέσματα σε σχέση με τον BLAST. Για παράδειγμα, χρησιμοποιώντας μέγεθος λέξης 23, ο χρόνος εκτέλεσης μπορεί να γίνει 45 φορές μικρότερος από τον χρόνο που χρειάζεται ο BLAST. Αυτά τα στοιχεία αποδεικνύουν σαφώς την αποδοτικότητα που έχει σαν εργαλείο αναζήτησης.

Τέλος, αναφέρουμε ότι miBLAST έχει δημιουργηθεί σαν μια επέκταση του NCBI BLAST, ενσωματώνοντας δηλαδή τον ήδη υπάρχοντα κώδικα του BLAST. Αυτός ο σχεδιασμός, ουσιαστικά επιτρέπει στον miBLAST την χρησιμοποίηση τμημάτων κώδικα του BLAST, τα οποία αφορούν την λειτουργία της στοίχισης και την μορφοποίηση του αποτελέσματος. Συμπερασματικά, ο miBLAST υιοθετεί το στατιστικό μοντέλο που εφαρμόζει ο BLAST, όπως και την μορφή του αποτελέσματος που παρουσιάζει σαν έξοδο. Λόγω του ότι πολλοί χρήστες είναι ήδη εξοικειωμένοι με το περιβάλλον του BLAST, αναμένεται ότι οι τωρινοί χρήστες του BLAST οι οποίοι χρειάζονται να εκτελέσουν υψηλού φόρτου εργασίας ερωτήματα, μπορούν εύκολα να κάνουν την μετάβαση και να χρησιμοποιήσουν τον miBLAST.

1.4.1 Υλικά και μέθοδοι

Το κεντρικό σημείο της στρατηγικής του miBLAST είναι η χρησιμοποίηση q-gram ευρετηρίων, με σκοπό τον γρήγορο εντοπισμό των ακολουθιών της βάσης οι οποίες περιέχουν «επιτυχία» λέξης, για κάθε ακολουθία του ερωτήματος. Επειδή για κάθε ακολουθία του ερωτήματος, υπάρχουν πιθανότατα λίγες ακολουθίες της βάσης με τις οποίες έχει κάποια κοινή λέξη, ο miBLAST χρειάζεται να εξετάσει μόνο το μικρό αυτό πλήθος ακολουθιών, που προκύπτει μετά από την αναζήτηση στο ευρετήριο. Αφότου γίνει η ανάκτηση αυτού του συνόλου των ακολουθιών, έπειτα πραγματοποιείται η επέκταση της στοίχισης, πέρα από την κοινή λέξη, έτσι ώστε να προκύψει η τελική στοίχιση. Στις επόμενες ενότητες, θα γίνει αρχικά η περιγραφή της δομής του ευρετηρίου q-gram και στην συνέχεια η λεπτομερής περιγραφή των αλγορίθμων που σχετίζονται με την εξαγωγή των επιλεγμένων ακολουθιών (filtering).

1.4.1.1 Σημειώσεις

Ο αλγόριθμος του miBLAST θεωρεί την ύπαρξη δύο συνόλων από ακολουθίες: ένα σύνολο με τις ακολουθίες του ερωτήματος $Q = \{q_1, q_2, \dots, q_i, \dots, q_m\}$ και ένα σύνολο με τις ακολουθίες της βάσης $D = \{d_1, d_2, \dots, d_j, \dots, d_n\}$, όπου τα q_i και d_j αναπαριστούν τις ακολουθίες. Κάθε ακολουθία στα σύνολα Q και D αναπαριστάται με ένα μοναδικό αριθμό (ID). Μία λέξη w ορίζεται σαν μία συμβολοσειρά η οποία

έχει σταθερό μήκος *l*. Η «επιτυχία λέξης» ορίζεται σαν ένα διατεταγμένο ζεύγος (i, j) όπου η ακολουθία του ερωτήματος q_i και η ακολουθία της βάσης d_j έχουν τουλάχιστον μία λέξη κοινή.

1.4.1.2 Η Δομή του Ευρετηρίου και η Κατασκευή του

Η δομή του ευρετηρίου που χρησιμοποιεί ο miBLAST είναι η εξής: Το q-gram ευρετήριο που κατασκευάζεται για ένα σύνολο ακολουθιών, περιέχει μία εγγραφή για κάθε μοναδική *επικαλυπτόμενη* λέξη μήκους *l*, που συναντάται σε αυτό σύνολο. Κάθε εγγραφή του ευρετηρίου αποθηκεύει την λίστα με τα IDs των ακολουθιών που περιέχουν την αντίστοιχη λέξη. Επισημαίνεται ότι κάθε αναφορά σε μία ακολουθία, περιλαμβάνει μόνο τον αριθμό ID της και όχι την θέση που βρίσκεται η λέξη μέσα στην ακολουθία. Συνεπώς, εάν μια λέξη εμφανίζεται παραπάνω από μία φορά σε μία ακολουθία, στην αντίστοιχη εγγραφή του ευρετηρίου η ακολουθία αυτή θα αναφέρεται μόνο μία φορά. Αυτή η επιλογή, να μην καταγράφεται η θέση της λέξης μέσα στην ακολουθία, έγινε με σκοπό την μείωση του μεγέθους των ευρετηρίων που παράγει ο miBLAST. (Ο αλγόριθμος μπορεί να γενικευτεί έτσι ώστε να λειτουργεί με ευρετήριο q-gram, στο οποίο θα αποθηκεύεται και η θέση της λέξης μέσα στην ακολουθία.) Στον Πίνακα 1.1 δίνεται ένα παράδειγμα για την δομή του q-gram ευρετηρίου.

Πίνακας 1.1**:** Ένα παράδειγμα της δομής του q-gram ευρετηρίου.

ID ακολουθίας	ακολουθία
1	ACAAAAA
2	AAAAAAAAC
3	CAACAACAA
4	CAACAACAA

λέξη (*w*)	ID ακολουθίας
AAA	1 2
AAC	2 3 4
ACA	1 3 4
CAA	1 3 4

Η αριστερή στήλη αναπαριστά ένα σύνολο δεδομένων από ακολουθίες και η δεξιά στήλη δείχνει την κατασκευή του ευρετηρίου για αυτό το σύνολο δεδομένων χρησιμοποιώντας μέγεθος λέξης 3.

Το κύριο πλεονέκτημα αυτής της δομής ευρετηρίου είναι η αποδοτικότητα που επιφέρει. Ο χρόνος κατασκευής του ευρετηρίου είναι γραμμικός $[O(n)]$ σε σχέση με το μέγεθος του συνόλου των ακολουθιών που δίνονται σαν είσοδο. Επιπλέον η

αναζήτηση για μία κοινή λέξη είναι πολύ αποδοτική, μιας και απαιτεί σταθερό χρόνο [$O(1)$].

Επειδή στο q-gram ευρετήριο υπάρχει μία εγγραφή για κάθε μοναδική λέξη μήκους q, το μέγεθος του ευρετηρίου μπορεί να προκύψει μεγάλο και να υπερβεί το μέγεθος της κύριας μνήμης που είναι διαθέσιμο. Για την αντιμετώπιση αυτού του προβλήματος, ο miBLAST χρησιμοποιεί την εκδοχή ευρετηρίου που αποθηκεύεται στον σκληρό δίσκο του υπολογιστή. Έτσι οι εγγραφές του ευρετηρίου ανακτούνται από τον δίσκο όταν ζητηθούν. Επιπλέον ο miBLAST είναι σχεδιασμένος έτσι ώστε η προσπέλαση του ευρετηρίου να γίνεται σε μεγάλο βαθμό σειριακά (σε αντίθεση με τις τυχαίες προσπελάσεις (I/O) που είναι πολύ περισσότερο δαπανηρές). Επισημαίνεται ότι η χρησιμοποίηση ευρετηρίου βασισμένου στο δίσκο από τον miBLAST έρχεται σε αντίθεση με κάποιες άλλες προσεγγίσεις που χρησιμοποιούν την τεχνική q-gram [2,8], οι οποίες υλοποιούν ευρετήριο που βρίσκεται στην κύρια μνήμη. Λαμβάνοντας υπόψη ότι το q-gram ευρετήριο είναι συνήθως μεγαλύτερο από την βάση, η χρησιμοποίηση ευρετηρίου ευρισκόμενου στην μνήμη συνεπάγεται ότι οι μέθοδοι δεν έχουν την δυνατότητα να χρησιμοποιηθούν σε περιπτώσεις με ογκώδης βάσεις δεδομένων.

Επίσης πρέπει να σημειωθεί ότι το q-gram ευρετήριο που χρησιμοποιεί ο miBLAST περιλαμβάνει όλες τις επικαλυπτόμενες λέξεις. Αυτός ο σχεδιασμός διασφαλίζει ότι ο αλγόριθμος του miBLAST δεν θα έχει απώλεια στην ευαισθησία σε σχέση με την μέθοδο του BLAST. Αντίθετα η χρησιμοποίηση τεχνικών με μη-επικαλυπτόμενες λέξεις στις άλλες προσεγγίσεις [2,8] έχει σαν αποτέλεσμα την απώλεια στην ευαισθησία, παράγοντας μολαταύτα μικρότερο ευρετήριο. Υπάρχει η πεποίθηση ότι η χρησιμοποίηση δομής μεγαλύτερου ευρετηρίου μπορεί να δικαιολογηθεί, δεδομένου ότι μία από τις επιδιώξεις στον σχεδιασμό του miBLAST είναι να παρέχει την ίδια ευαισθησία με τον BLAST.

Επιπλέον για να διασφαλιστεί ότι η χρησιμοποίηση ευρετηρίων, που αποθηκεύονται δίσκο, δεν προκαλεί μεγάλες επιπτώσεις στην απόδοση, ο miBLAST κάνει προσεκτική χρήση της σειριακής προσπέλασης (ακολουθιακή πρόσβαση). Οι εγγραφές του ευρετηρίου είναι ταξινομημένες με βάση την λογική θέση του αρχείου, που ορίζεται από το λειτουργικό σύστημα και οι προσπελάσεις των λογικών τμημάτων (blocks) του ευρετηρίου από τον miBLAST, γίνονται σειριακά. Η λογική σειριακή προσπέλαση συχνά αντιστοιχεί στην φυσική σειριακή προσπέλαση, δεδομένου ότι τα λειτουργικά συστήματα προσπαθούν να αποθηκεύουν τα δεδομένα

σε ένα αρχείο σε συνεχόμενα blocks του δίσκου στο φυσικό επίπεδο. Αυτή η τεχνική της σειριακή προσπέλασης έχει σημαντικά οφέλη στην απόδοση, μίας και η σάρωση φυσικώς γειτονικών τμημάτων είναι πολύ περισσότερο αποτελεσματική σε σχέση με την τυχαία προσπέλαση τμημάτων, που είναι διασκορπισμένα στον δίσκο. Επιπλέον τα λειτουργικά συστήματα συχνά φέρνουν εκ των προτέρων (prefetch) στην μνήμη τα γειτονικά τμήματα δεδομένων, γεγονός που βελτιώνει ακόμα περισσότερο την απόδοση κατά την σειριακή προσπέλαση.

Το μέγεθος λέξης αποτελεί καθοριστική παράμετρος στον BLAST, η οποία ορίζει το μέγεθος της λέξης που χρησιμοποιείται για την ανίχνευση μίας κοινής λέξης (επιτυχία λέξης). Εφ' όσον σε κάθε ερώτημα μπορεί να ορίζεται διαφορετικό μέγεθος λέξης σαν είσοδο, ο miBLAST πρέπει να εφαρμόζει μια στρατηγική για να ανταπεξέρχεται σε διαφορετικά μεγέθη λέξης. Μια απλοϊκή στρατηγική που μπορεί να εφαρμοστεί, είναι να για κάθε πιθανό μέγεθος λέξης να κατασκευαστεί και ένα q-gram ευρετήριο. Εντούτοις το κόστος κατασκευής τόσων πολλών ευρετηρίων είναι απαγορευτικά μεγάλο. Για να αποφύγει το υψηλό αυτό κόστος, ο miBLAST χρησιμοποιεί μεθόδους που επιτρέπουν την επαναχρησιμοποίηση ενός ευρετηρίου ακόμα και σε ερωτήματα με παράμετρο διαφορετικό μέγεθος λέξης.

Στην ουσία, αυτή η ευελιξία στην χρησιμοποίηση του q-gram ευρετηρίου για διαφορετικά μεγέθη λέξης, αποτελεί τη θεμελιώδη διαφορά μεταξύ της χρήσης του q-gram ευρετηρίου από τον miBLAST σε σχέση με προηγούμενες προσεγγίσεις βασισμένες στην τεχνική q-gram [2,8]. Με τον miBLAST θα μπορούσε στην πράξη να αποθηκεύεται μόνο ένα q-gram ευρετήριο, με το μικρότερο υποστηριζόμενο μήκος λέξης σαν παράμετρο. Διαφορετικά θα μπορούσε να διατηρείται ένα μικρό πλήθος ευρετηρίων ˙ και για κάθε ερώτημα να επιλέγεται το q-gram ευρετήριο με την τιμή q που είναι πλησιέστερη στο μέγεθος λέξης που ορίζεται από το ερώτημα.

1.4.1.3 Ο Αλγόριθμος του miBLAST

Ο αλγόριθμος του miBLAST αποτελείται από τρία κύρια βήματα. Πρώτα κατασκευάζονται τα δύο q-gram ευρετήρια – ένα για τις ακολουθίες του ερωτήματος και ένα για τις ακολουθίες της βάσης. Δεύτερον, χρησιμοποιώντας αυτά τα ευρετήρια, ο αλγόριθμος επιλέγει (filtering), για κάθε ακολουθία του ερωτήματος, τις ακολουθίες της βάσης με τις οποίες υπάρχει κάποια κοινή λέξη (επιτυχία λέξης).

Αλγόριθμος 1 miBLAST(Q,D,l)

ΕΙΣΟΔΟΙ:

• $Q = \{q_1, q_2, \ldots, q_i, \ldots, q_m\}$ είναι ένα σύνολο με τις ακολουθίες του ερωτήματος

• $D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$ είναι ένα σύνολο με τις ακολουθίες της βάσης

• *ID* είναι ένας μοναδικός αριθμός που αντιπροσωπεύει κάθε ακολουθία, q_i και d_j στα σύνολα Q και D

• m είναι το μήκος λέξης που χρησιμοποιείται για την κατασκευή του ευρετηρίου

• l είναι το μήκος λέξης για την αναζήτηση από τον BLAST

ΜΕΤΑΒΛΗΤΕΣ:

• $D_{ID} = \{1,2, \ldots, n\}$ είναι ένα σύνολο με τα IDs των ακολουθιών της βάσης στο D

• $F_i \subseteq D_{ID}$ είναι ένα σύνολο με τα IDs των ακολουθιών που έχουν κοινή λέξη με την q_i

• $Filtered = \{F_1, F_2, \ldots, F_m\}$ είναι ένα σύνολο με τα F για όλες τις q_i μέσα στο Q

• *DIndex* είναι το q-gram ευρετήριο για το D

• *QIndex* είναι το q-gram ευρετήριο για το Q

• *DIndex.L*(w) είναι η λειτουργία που εξάγει από το DIndex την λίστα με τις ακολουθίες που περιέχουν μια λέξη w.

1: Κατασκευή του *DIndex* για το D χρησιμοποιώντας μήκος λέξης m, εάν δεν υπάρχει ήδη.
2:
3: **if** $l \leq m$ **then**
4: *Filtered*=INDEX-JOIN FILTER (*DIndex*, Q, l, m)
5: **else if** $l > m$ **then**
6: *Filtered*=SLIDING-WINDOW FILTER (*DIndex*, Q, l, m)
7: **end if**
8:
9: **for all** sequence q_i in Q **do**
10: **for all** sequenceID j in F_i of *Filtered* **do**
11: Εκτελεί στοίχιση μεταξύ της q_i από το Q και της d_j από το D
12: **end for**

Τέλος, καλείται το τμήμα του BLAST που εκτελεί την στοίχιση, δίνοντας σαν είσοδο τις επιλεγμένες ακολουθίες της βάσης ˙ έτσι ώστε να παραχθεί η τελική στοίχιση.

Η μέθοδος του miBLAST περιγράφεται στο «Αλγόριθμος 1». Ο miBLAST δέχεται σαν είσοδο ένα σύνολο με τις ακολουθίες του ερωτήματος, ένα σύνολο με τις ακολουθίες της βάσης καθώς και το μέγεθος λέξης που θα χρησιμοποιηθεί για την στοίχιση από τον BLAST. Βάσει των εισόδων αυτών, ο miBLAST ξεκινά δημιουργώντας το ευρετήριο της βάσης. Στην πράξη, το ευρετήριο της βάσης είναι συχνά εκ των προτέρων κατασκευασμένο και βρίσκεται αποθηκευμένο στον σκληρό δίσκο ˙ έτσι το ίδιο ευρετήριο μπορεί να επαναχρησιμοποιηθεί σε πολλές αναζητήσεις.

Επισημαίνεται ότι στις γραμμές 3-7 του Αλγορίθμου 1, ο miBLAST εκτελεί δύο διαφορετικούς αλγόριθμους φιλτραρίσματος των ακολουθιών (filtering), η επιλογή των οποίων εξαρτάται από το μήκος λέξης *l*. Εάν το μήκος λέξης *l* που ορίζεται στο ερώτημα, είναι μικρότερο ή ίσο με το μήκος λέξης του ευρετηρίου *m*, τότε ο miBLAST χρησιμοποιεί τον αλγόριθμο συνένωσης ευρετηρίων (index join) για το φιλτράρισμα των ακολουθιών (παρουσιάζεται στον Αλγόριθμο 2). Εντούτοις, εάν $l > m$, ο miBLAST χρησιμοποιεί την μέθοδο κυλιόμενου παραθύρου για την εκτέλεση της διαδικασίας φιλτραρίσματος (επιλογής), η οποία και παρουσιάζεται στον Αλγόριθμο 3.

Αναζήτηση στο Ευρετήριο

Η διαδικασία αναζήτησης στο ευρετήριο (index lookup), $L(w)$, επιστρέφει μία λίστα με τα IDs των ακολουθιών που περιέχουν την λέξη *w* μήκους *l*. Όταν το μήκος λέξης του ερωτήματος είναι ίσο με το μήκος λέξης του ευρετηρίου, $l = m$, η λίστα αυτή μπορεί να ανακτηθεί πραγματοποιώντας μια απλή αναζήτηση στο ευρετήριο. Ωστόσο, όταν $l < m$, κατά τη διαδικασία αναζήτησης στο ευρετήριο πραγματοποιούνται πολλαπλές αναζητήσεις. Σε αυτές τις διαπεράσεις του ευρετηρίου αναζητούνται όλες οι λέξεις των οποίων το πρόθεμα ταυτίζεται με κάποια λέξη του ερωτήματος. Η τελική λίστα του αποτελέσματος κατασκευάζεται υπολογίζοντας απλώς την ένωση όλων των λιστών με τις ακολουθίες οι οποίες έχουν προκύψει από κάθε αναζήτηση. Επειδή το q-gram ευρετήριο που χρησιμοποιείται είναι ταξινομημένο αλφαβητικά, αυτές οι πολλαπλές αναζητήσεις ουσιαστικά

πραγματοποιούνται με σειριακή αναζήτηση στο ευρετήριο, καθιστώντας την διαδικασία των αυτή αρκετά αποτελεσματική.

Αλγόριθμος 2 INDEX-JOIN FILTER (*DIndex*, *Q*, *l*, *m*)

```
1: Κατασκευή του QIndex για το Q χρησιμοποιώντας μήκος λέξης m l
2: Κατασκευή του πίνακα bitmap[|Q|, |D|]
3:
4: for all word w in QIndex do
5:     If QIndex.L(w) ≠ NULL & DIndex.L(w) ≠ NULL then
6:         L_q ← QIndex.L(w)
7:         L_d ← DIndex.L(w)
8:         for i ← 1 to |L_q| do
9:             for j ← 1 to |L_d| do
10:              bitmap[L_q[i], L_d[j]] = TRUE
11:            end for
12:        end for
13:    end if
14: end for
15:
16: for i ← 1 to |Q| do
17:    for j ← 1 to |D| do
18:        if bitmap[i, j] = TRUE then
19:            F_i = F_i ∪ {j}
20:        end if
21:    end for
22: end for
23:
```

Index Join Filtering

Με σκοπό να εντοπίσει όλες τις κοινές λέξεις (word hits) μεταξύ του συνόλου των ακολουθιών του ερωτήματος *Q* και του συνόλου των ακολουθιών της βάσης *D*, ο miBLAST χρησιμοποιεί μια τεχνική συνένωσης ευρετηρίων. Αρχικά κατασκευάζονται δύο ευρετήρια, ένα για το *D* και ένα για το *Q*. Αφού ολοκληρωθεί η κατασκευή των ευρετηρίων, ο miBLAST συνενώνει τα ευρετήρια αυτά με βάση τις λέξεις. Σε αυτήν την υλοποίηση του q-gram ευρετηρίου, χρησιμοποιούνται απλά τα λιγότερο σημαντικά (least-significant) bits από κάθε χαρακτήρα της λέξης, πράγμα που ουσιαστικά έχει ως αποτέλεσμα οι εγγραφές του ευρετηρίου να είναι ταξινομημένες αλφαβητικά.

Ξεκινώντας με την πρώτη λέξη του ευρετηρίου του ερωτήματος, για κάθε λέξη *w* γίνεται ανάγνωση του ευρετηρίου με σκοπό την εξαγωγή της λίστας $L_q(w) = (w, y, z, \ldots)$ όπου οι ακολουθίες q_x, q_y και q_z περιέχουν την λέξη *w*. Επόμενο βήμα είναι η ανάγνωση του ευρετηρίου της βάσης αναζητώντας την ίδια λέξη, έτσι ώστε να γίνει εξαγωγή της λίστας $L_d(w) = (p, q, r, \ldots)$. Έπειτα υπολογίζεται το καρτεσιανό γινόμενο των στοιχείων που περιέχονται σε αυτές τις δύο λίστες-σύνολα, έτσι ώστε να παραχθούν τα ζεύγη των ακολουθιών που έχουν μεταξύ τους κάποια κοινή λέξη, (x, p), (x, q), (x, r), (y, p), (y, q), (y, r), … Δηλαδή κάθε ζεύγος υποδηλώνει μία ακολουθία του ερωτήματος που έχει κάποια κοινή λέξη με μια ακολουθία της βάσης. Αυτές οι ακολουθίες, στην συνέχεια, εξετάζονται για την δημιουργία της τελικής στοίχισης, χρησιμοποιώντας τον αλγόριθμο στοίχισης του BLAST. Αυτή η διαδικασία συνεχίζεται μέχρις ότου εξεταστούν όλα τα ζεύγη ακολουθιών από το *Q* και το *D*.

Σημειώνεται ότι με την συνένωση των δύο ευρετηρίων παράγεται η συνολική λίστα με τα ζεύγη των ακολουθιών από το ερώτημα και την βάση που περιέχουν κοινή λέξη. Εμφανίζονται δύο ενδιαφέροντα χαρακτηριστικά όσον αφορά τα περιεχόμενα αυτής της λίστας. Πρώτον, η λίστα αυτή θα μπορούσε να περιέχει ένα πλήθος διπλών εγγραφών, αφού κάποια ακολουθία της βάσης και κάποια ακολουθία, υπάρχει η πιθανότητα να έχουν παραπάνω από μία κοινή λέξη. Δεύτερον, η λίστα αυτή δεν είναι ταξινομημένη με βάση τους αύξοντες αριθμούς των ακολουθιών του ερωτήματος. Ένας απλός τρόπος για την παραγωγή της τελικής στοίχισης θα ήταν η επέκταση της στοίχισης πέρα από την κοινή λέξη κάθε ζεύγους ακολουθιών. Ωστόσο, εξαιτίας του πρώτου χαρακτηριστικού της λίστας, μία ακολουθία της βάσης

ενδέχεται να χρειάζεται να εξεταστεί περισσότερες από μία φορές για την δημιουργία της στοίχισης με το ίδιο ερώτημα, πράγμα που δεν είναι τόσο αποδοτικό. Επίσης, το αποτέλεσμα που εξάγεται θα περιέχει στοιχίσεις που δεν θα είναι ομαδοποιημένες με βάση τους αριθμούς των ακολουθιών του ερωτήματος. Συνεπώς το αποτέλεσμα θα πρέπει να ταξινομηθεί πριν παρουσιαστεί στον χρήστη. Αυτή η διαδικασία ταξινόμησης μπορεί να είναι αρκετά δαπανηρή, ιδιαίτερα σε περιπτώσεις ερωτημάτων μεγάλου φόρτου εργασίας.

Για να ανταπεξέλθει ο miBLAST σε τέτοιες περιπτώσεις, εφαρμόζει μια αποδοτική εκδοχή διάδικου πίνακα (bitmap). Χρησιμοποιείται μια δομή ενός διάδικου πίνακα δύο διαστάσεων, ο οποίος περιέχει $|Q| \times |D|$ θέσεις, όπου τα $|Q|$ και $|D|$ αντιπροσωπεύουν το πλήθος των ακολουθιών του ερωτήματος και της βάσης αντίστοιχα. Όλες οι θέσεις του πίνακα (bits) αρχικοποιούνται στο 0 –FALSE. Όταν εντοπιστεί ένα ζεύγος ακολουθιών με κοινή λέξη (i, j), η αντίστοιχη θέση του πίνακα (i, j) παίρνει τιμή 1 – TRUE. Μετά την ολοκλήρωση της διαδικασίας συνένωσης των ευρετηρίων, οι θέσεις του πίνακα με τιμή TRUE αντιπροσωπεύουν τα ζεύγη των ακολουθιών της βάσης και του ερωτήματος που έχουν τουλάχιστον μία κοινή λέξη.

Σαρώνοντας διαδοχικά τις γραμμές του πίνακα, για κάθε ακολουθία του ερωτήματος, προκύπτει μία ταξινομημένη λίστα με τα IDs των ακολουθιών της βάσης, με τις οποίες θα πραγματοποιηθεί η λειτουργία της στοίχισης. Τα αποτελέσματα που παράγονται είναι φυσικά ταξινομημένα, με βάση την σειρά των ακολουθιών του ερωτήματος και επίσης κάθε ακολουθία της βάσης εξετάζεται το πολύ μια φορά για κάθε ακολουθία του ερωτήματος.

Sliding-window Filtering

Επειδή το κόστος κατασκευής ενός q-gram ευρετηρίου για μία ογκώδης βάση μπορεί να είναι υψηλό, είναι σκόπιμο να εξεταστούν μέθοδοι που επιτρέπουν την επαναχρησιμοποίηση ενός υπάρχοντος ευρετηρίου, όταν αυτό είναι δυνατό. Παρόλα αυτά, επειδή το μήκος λέξης για την αναζήτηση είναι μια παράμετρος που ορίζεται από τον χρήστη, η επαναχρησιμοποίηση ενός ευρετηρίου με μήκος λέξης διαφορετικό από το μήκος λέξης που έχει δοθεί σαν παράμετρος, αποτελεί πρόκληση. Εδώ γίνεται αναφορά στην τεχνική κυλιόμενου παραθύρου (sliding-window), η οποία επιτρέπει στον miBLAST να επαναχρησιμοποιήσει ένα ευρετήριο

όταν το μήκος λέξης του ερωτήματος είναι μεγαλύτερο από το μήκος λέξης του ευρετηρίου.

Αλγόριθμος 3 SLIDING-WINDOW FILTER(*D*Index, *Q*, *l*, *m*)

```
1: Κατασκευή ενός διάδικου πίνακα bitmap[|Q|, |D|]
2: Κατασκευή ενός μετρητή counter[|D|]
3:
4: for all sequences q_i in Q do
5:       for all words w of length l in q_i do
6:                initialize the counter
7:                for all all substrings w′ of length m in w do
8:                        if DIndex.L(w′) ≠ NULL then
9:                                L_d ← DIndex.L(w′)
10:                               for j ← 1 to |L_d| do
11:                                       counter[L_d[j]] ++
12:                               end for
13:                       end if
14:               end for
15:               for j ← 1 to |D| do
16:                   if counter [j] = l − m + 1 then
17:                       bitmap [i,j] = TRUE
18:                   end if
19:               end for
20:     end for
21: end for
22:
23: for i ← 1 to |Q| do
24:         for  j ← 1 to |D| do
25:                if bitmap [i, j] = TRUE then
26:                       F_i = F_i ∪ {j}
27:                end if
28:         end for
29: end for
```

Για την επεξήγηση της μεθόδου κυλιόμενου παραθύρου, θα χρησιμοποιηθούν οι ακόλουθες σημειώσεις: το $s[1..k]$ δηλώνει μία λέξη μήκους k μέσα σε μια ακολουθία s. Τα m και l δηλώνουν το μήκος λέξης του ευρετηρίου και του ερωτήματος αντίστοιχα.

Η μέθοδος κυλιόμενου παραθύρου βασίζεται στην παρατήρηση πως εάν μια λέξη του ερωτήματος $s[1..l]$ περιέχεται σε μια ακολουθία της βάσης di, τότε όλες οι υπό-ακολουθίες της μήκους m, $s[1..m]$, $s[2..m+1]$, $s[3..m+2]$, … , $s[l - m+1..l]$, περιέχονται επίσης στην ακολουθία di. Συνεπώς, η εγγραφή του ευρετηρίου για κάθε μία από αυτές τις υπό-ακολουθίες, πρέπει να περιέχει ID ακολουθίας με τιμή i. Αυτή η ιδιότητα παρέχει την αναγκαία συνθήκη έτσι ώστε η λέξη $s[1.l]$ να βρίσκεται σε μια ακολουθία di. Επισημαίνεται ότι αυτή η συνθήκη δεν αποτελεί επαρκή συνθήκη, το οποίο συνεπάγεται ότι αυτή η τεχνική ενδέχεται να παράγει ψευδώς θετικά. Εντούτοις, τα ψευδώς θετικά μπορούν εύκολα να αφαιρεθούν ελέγχοντας απλώς τις ακριβής επιτυχίες λέξης (word hits) κατά την ανάκτηση μιας ακολουθίας από την βάση.

Σημειώνεται ότι μια παρόμοια τεχνική χρησιμοποιείται επίσης από τον A/GBLAST για να εντοπίζει τις ταυτίσεις λέξεων μήκους l μεταξύ του ερωτήματος και της βάσης χρησιμοποιώντας ένα πίνακα αναζήτησης για λέξεις μήκους m που βρίσκονται στην ακολουθία του ερωτήματος. Ο αλγόριθμος για την μέθοδο κυλιόμενου παραθύρου παρουσιάζεται στο Αλγόριθμος 3.

1.4.1.4 Υλοποίηση του miBLAST

Η διαδικασία υλοποίησης του miBLAST αποτελείται από τρία κύρια μέρη: την κατασκευή των q-gram ευρετηρίων, το φιλτράρισμα των ακολουθιών και την δημιουργία της στοίχισης. Η διαδικασία κατασκευής ευρετηρίου παίρνει σαν είσοδο μια μορφοποιημένη (formatted) βάση δεδομένων, η οποία παράγεται με το εργαλείο formatdb της NCBI, και έπειτα κατασκευάζει ένα q-gram ευρετήριο για αυτήν τα βάση. Η διαδικασία φιλτραρίσματος –επιλογής παίρνει σαν είσοδο το ευρετήριο της βάσης και το σύνολο των ακολουθιών της βάσης και εξάγει ένα σύνολο με τις ακολουθίες της βάσης που περιέχουν κοινή λέξη με τις ακολουθίες του ερωτήματος. Τέλος, το εργαλείο δημιουργίας των στοιχίσεων υπολογίζει τις τελικές στοιχίσεις των ακολουθιών. Το εργαλείο δημιουργίας στοιχίσεων της εφαρμογής, χρησιμοποιεί το καθιερωμένο εργαλείο στοιχίσεων του NCBI BLAST, το οποίο έχει τροποποιηθεί

έτσι ώστε να απαιτείται η εξέταση μόνο ενός υποσυνόλου των ακολουθιών της βάσης, το οποίο υποσύνολο έχει προκύψει από την διαδικασία φιλτραρίσματος.

Ο miBLAST έχει γραφτεί σε γλώσσα C χρησιμοποιώντας το NCBI toolkit. Η τρέχουσα υλοποίηση του miBLAST χρησιμοποιεί τον NCBI BLAST v.2.2.8 και υποστηρίζει ερωτήματα για νουκλεοτιδικά σύνολα δεδομένων. Αυτές οι τροποποιήσεις προσθέτουν λιγότερες από χίλιες γραμμές στον κώδικα.

1.4.2 Αξιολόγηση του miBLAST

Εδώ παρουσιάζονται τα αποτελέσματα της εμπειρικής αξιολόγησης του miBLAST που περιλαμβάνονται στην επίσημη δημοσίευση του. Η βάση δεδομένων που χρησιμοποιήθηκε για την εμπειρική αξιολόγηση είναι η NCBI Human UniGene build #177. Αυτή η βάση δεδομένων περιέχει 5 064 621 ακολουθίες και περίπου 3.19 gigabases.

Κατά την αξιολόγηση χρησιμοποιήθηκε ένα πλήθος ερωτημάτων έτσι ώστε να εξεταστεί η επίδραση των εξής παραμέτρων: το πλήθος των ακολουθιών του ερωτήματος, το μήκος λέξης και το μήκος των ακολουθιών του ερωτήματος. Τα δύο πρώτα σύνολα που χρησιμοποιούνται ως ερώτημα για τις δοκιμές, προέρχονται από την συλλογή Human Genome U133A, η οποία περιέχει ολιγονουκλεοτιδικές ακολουθίες που διατηρεί η Affymetrix και από την συλλογή RefSet Oligos για το ανθρώπινο γονιδίωμα της Illumina Inc. Η συλλογή της Affymetrix περιέχει 247.965 ακολουθίες με μέσο μήκος 25 βάσεις. Η συλλογή της Illumina περιέχει 22,740 ακολουθίες με μέσο μήκος 70 βάσεις. Αυτά τα σύνολα ερωτημάτων χρησιμοποιούνται για την εξακρίβωση εάν τα περιεχόμενα των συλλογών αυτών συμβαδίζουν με τις τρέχοντες συστάδες (clusters) της Unigene, λόγω του ότι οι συστάδες της τροποποιούνται σε τακτά χρονικά διαστήματα. Επιπλέον, χρησιμοποιήθηκαν ως ερώτημα, σύνολα ακολουθιών ποικίλου μήκους (από 16 bp μέχρι 512 bp), οι οποίες εξήχθησαν με τυχαία επιλογή μέσα από την βάση δεδομένων που περιέχει τα ESTs του ανθρώπου. Τα συγκεκριμένα ερωτήματα είχαν σκοπό την μέτρηση της επίδρασης που έχει το μήκους των ακολουθιών του ερωτήματος, στην απόδοση διαφόρων αλγορίθμων.

Για την αξιολόγηση της απόδοσης του miBLAST, θεωρείται σκόπιμο να γίνει σύγκριση του miBLAST σε σχέση με άλλα εργαλεία όπως ο MegaBLAST, BLAST++, NCBI-BLAST και MPBLAST. Ωστόσο, όπως αναφέρθηκε παραπάνω,

κάποιες από αυτές τις μεθόδους δεν μπορούν να είναι άμεσα συγκρίσιμες με τον miBLAST.

Έτσι δεν έγινε σύγκριση του miBLAST με τον MegaBLAST, λόγω του ότι, όπως είναι γνωστό, η ευαισθησία του MegaBLAST είναι συνήθως μικρότερη από ευαισθησία που έχει ο κλασικός BLAST.

Όσον αφορά τον BLAST++, η έκδοση του την δεδομένη στιγμή που πραγματοποιήθηκαν τα πειράματα, δεν μπορούσε να χειριστεί την βάση δεδομένων UniGene build #177 εξαιτίας του μέγεθος της. Έτσι χρησιμοποιήθηκε μόνο το πρώτο μισό αυτής της βάσης για την σύγκριση του miBLAST με τον BLAST++.

Σε αυτή την ενότητα παρουσιάζεται η εκτεταμένη σύγκριση του miBLAST με τον NCBI BLAST v.2.2.8. Για τα ερωτήματα που περιέχουν πολλαπλές ακολουθίες, ο NCBI BLAST μπορεί να χρησιμοποιηθεί με δύο τρόπους. Η πρώτη εκδοχή είναι να εκτελεστεί ο BLAST επαναληπτικά, για κάθε ακολουθία του ερωτήματος. Η δεύτερη εκδοχή είναι η χρησιμοποίηση της επιλογής «-B», που υπήρχε στην συγκεκριμένη έκδοση του BLAST. Αυτή η εκδοχή, που αποκαλείται εδώ ως BLAST-B, ουσιαστικά εφαρμόζει την μέθοδο πολυπλεξίας του χρησιμοποιείται στον MPBLAST [56]. Σε αυτή την προσέγγιση, ένας ορισμένος αριθμός ακολουθιών του ερωτήματος συνδέεται αλυσιδωτά ώστε να παραχθεί μία μεγάλη αλληλουχία. Έπειτα καλείται ο παραδοσιακός BLAST για την εκτέλεση του ερωτήματος με την συνδεδεμένη αλληλουχία.

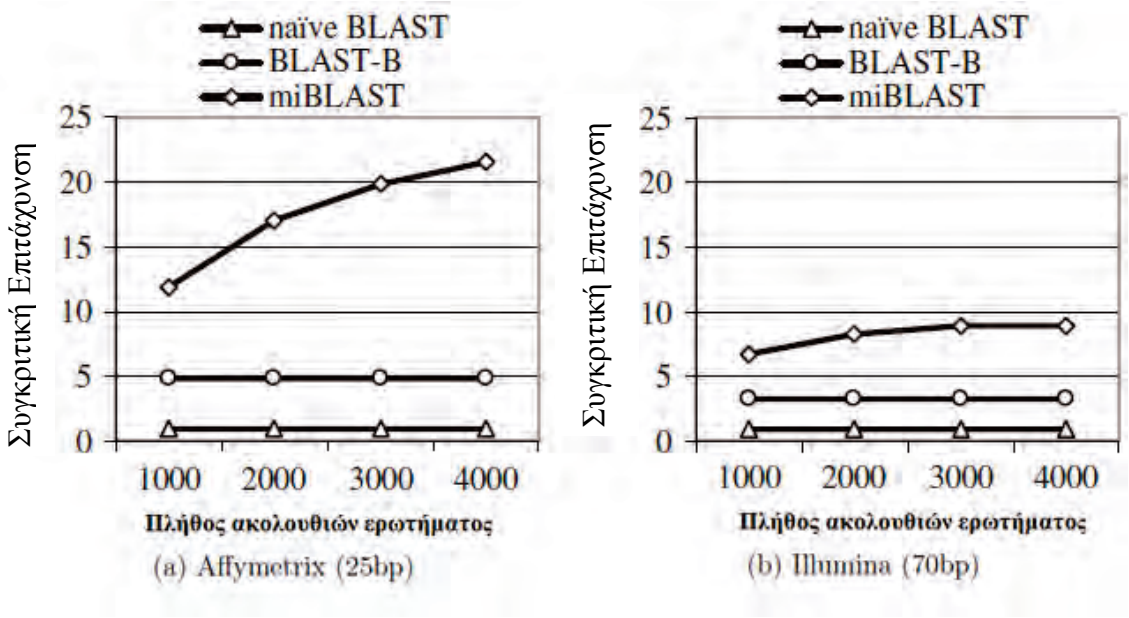
Για την εκτέλεση του BLAST-B, ο χρήστης πρέπει να ορίσει το πλήθος των ακολουθιών του ερωτήματος που θα συνδεθούν. Σε αυτή την έκδοση του BLAST, το ανώτερο όριο στον αριθμό των ακολουθιών που μπορούν να συνδεθούν είναι 255. Εντούτοις, παρόλο που ο χρήστης μπορεί να θέσει μια τιμή μεταξύ 2 και 255, παρατηρείται ότι υπάρχει συχνά ένα βέλτιστο πλήθος ακολουθιών. Γενικά η απόδοση βελτιώνεται στην αρχή, καθώς η έκταση της αλληλουχίας αυξάνεται, αλλά ξεκινά να πέφτει σταδιακά μετά από ένα σημείο. Το βέλτιστο σημείο μπορεί να μεταβάλλεται ανάλογα με το μήκος των ακολουθιών του ερωτήματος, το σύνολο δεδομένων της βάσης, καθώς και ανάλογα με τις παραμέτρους αναζήτησης του BLAST. Για τον BLAST-B, πραγματοποιήθηκαν κάποιες δοκιμές έτσι ώστε να βρεθεί το βέλτιστο πλήθος ακολουθιών για κάθε ερώτημα και να χρησιμοποιηθεί έπειτα κατά τις μετρήσεις.

Η εκτέλεση των τριών αυτών μεθόδων έγινε με την χρήση των προεπιλεγμένων παραμέτρων του BLAST. Οι παρακάτω δοκιμές πραγματοποιήθηκαν σε ένα

μηχάνημα με επεξεργαστή 2.2GHz AMD Opteron και 4GB RAM που τρέχει τον πυρήνα των Linux, 2.6.9. Όλες οι μετρήσεις αφορούν την περίπτωση όπου η μνήμη cache είναι άδεια πριν την έναρξη της εκτέλεσης ενός ερωτήματος.

1.4.2.1 Επίδραση του πλήθους των ακολουθιών του ερωτήματος

Για να εξεταστεί η επίδραση του πλήθους των ακολουθιών του ερωτήματος στην απόδοση των miBLAST, BLAST και BLAST-B, δοκιμάστηκαν ερωτήματα με πλήθος ακολουθιών από 1000 έως 4000. Τα γραφήματα 2.1 (a) και 2.1 (b) παρουσιάζουν την επιτάχυνση σε σχέση με τον BLAST, χρησιμοποιώντας τις συλλογές Affymetrix (25bp) and Illumina (70bp) αντίστοιχα.



Γράφημα 1.5: Η συγκριτική επιτάχυνση κάθε μεθόδου σε σχέση με τον απλό BLAST για ερωτήματα με διαφορετικό πλήθος ακολουθιών, χρησιμοποιώντας μέγεθος λέξης 11

Όπως φαίνεται στο Γράφημα 1.5, ο miBLAST είναι σημαντικά ταχύτερος από τις άλλες δυο μεθόδους. Η απόδοση του miBLAST βελτιώνεται καθώς το πλήθος των ακολουθιών αυξάνεται, και στην περίπτωση με τις 4000 ακολουθίες, ο miBLAST είναι 21.6 και 9.9 φορές ταχύτερος από τον BLAST για τα ερωτήματα με 25 bp και 70 bp αντίστοιχα, και 4.5 και 2.7 φορές ταχύτερος από τον BLAST-B για τα ερωτήματα με 25 bp και 70 bp αντίστοιχα.

Πίνακας 1.2: Επίδραση του πλήθους των ακολουθιών του ερωτήματος

Πλήθος Ακολουθιών Ερωτήματος	Κόστος Φιλτραρίσματος ανά Ακολουθία (25 bp)	Κόστος Στοίχισης ανά Ακολουθία (25 bp)	Κόστος Φιλτραρίσματος ανά Ακολουθία (70 bp)	Κόστος Στοίχισης ανά Ακολουθία (70 bp)
1000	0.44 (54%)	0.37 (46%)	0.46 (31%)	1.02 (69%)
2000	0.23 (42%)	0.33 (58%)	0.25 (21%)	0.96 (79%)
3000	0.17 (36%)	0.31 (64%)	0.18 (16%)	0.94 (84%)
4000	0.14 (31%)	0.30 (69%)	0.17 (15%)	0.94 (85%)

Ο αναλυτικός χρόνος εκτέλεσης του miBLAST για τις δοκιμές που φαίνονται στο Γράφημα 2.1. Όλοι οι χρόνοι αναφέρονται σε δευτερόλεπτα (sec)

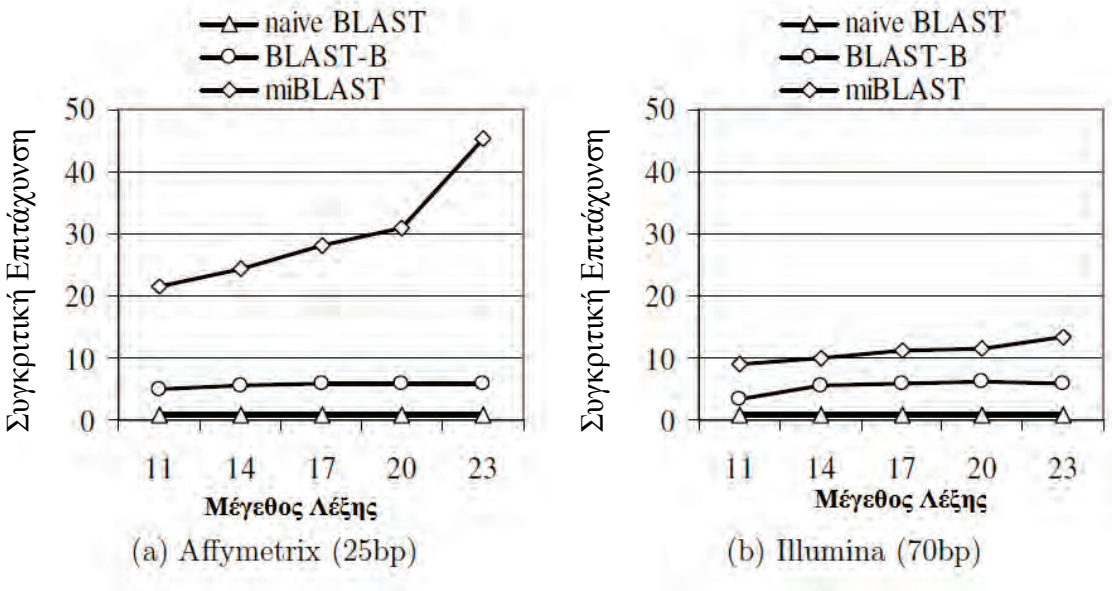
Για την εξήγηση, του γιατί η απόδοση του miBLAST βελτιώνεται καθώς το πλήθος των ακολουθιών αυξάνεται, παρουσιάζεται ο Πίνακας 1.2, ο οποίος δείχνει την δραστική μείωση του κόστους φιλτραρίσματος (λειτουργία συνένωσης ευρετηρίων) και του κόστους κλήσης της μεθόδου στοίχισης του BLAST για το κάθε ερώτημα. Όπως μπορεί να φανεί από αυτόν τον πίνακα, τόσο το κόστος του φιλτραρίσματος όσο και το κόστος της στοίχισης ανά ακολουθία, μειώνονται καθώς το πλήθος των ακολουθιών αυξάνεται. Όσον αφορά το κόστος φιλτραρίσματος, με μεγαλύτερο πλήθος ακολουθιών στο ερώτημα, το κόστος συνένωσης των ευρετηρίων μοιράζεται, έχοντας σαν αποτέλεσμα την μείωση του κόστους φιλτραρίσματος ανά ακολουθία ερωτήματος. Όσον αφορά το κόστος στοίχισης, η μείωση του κόστους ανά ακολουθία οφείλεται στα πλεονεκτήματα από την χρήση της μνήμης cache του λειτουργικού συστήματος. Για τις αρχικές ακολουθίες του ερωτήματος, η προσκόμιση μιας ακολουθίας της βάσης συχνά απαιτεί μια λειτουργία Ε/Ε (*I/O*) του δίσκου. Όμως, όσο το πλήθος των ακολουθιών του ερωτήματος αυξάνεται, αυξάνεται και η πιθανότητα μια ακολουθία της βάσης να σχετίζεται με παραπάνω από μία ακολουθίες του ερωτήματος. Επειδή απαιτούνται επαναλαμβανόμενες προσβάσεις σε μια ακολουθία του ερωτήματος, είναι αρκετά πιθανόν αυτή η ακολουθία να βρίσκεται στην μνήμη cache του λειτουργικού συστήματος και έτσι να μην χρειάζεται να γίνει κάποιο Ε/Ε δίσκου. Συνεπώς, όσο αυξάνεται το μέγεθος του ερωτήματος, τόσο το κόστος της στοίχισης μειώνεται.

Πρέπει να σημειωθεί επίσης ότι η αποθήκευση τμημάτων της βάσης δεδομένων στην μνήμη cache έχει ως αποτέλεσμα την περαιτέρω βελτίωση της απόδοσης του

miBLAST. Για παράδειγμα, όταν εκτελούνται διαδοχικά πέντε ερωτήματα με 4000 ακολουθίες, η σχετική επιτάχυνση της επεξεργασία του τελευταίου ερωτήματος φτάνει τις 25.4, ενώ η σχετική επιτάχυνση του πρώτου ερωτήματος είναι 21.6 φορές. Ο λόγος αυτής της βελτίωσης είναι ότι οι εκτελέσεις 2-4 επωφελούνται από το γεγονός της ανάγνωσης των δεδομένων από την μνήμη cache, όπου είχαν μεταφερθεί μετά από την εκτέλεση των προηγούμενων ερωτημάτων. Ο miBLAST επωφελείται επιπλέον από αυτήν την εναποθήκευση (caching), λόγω του ότι απαιτούνται περισσότερες διαδικασίες Ε/Ε στον δίσκο, όσο μεγαλύτερο είναι το ευρετήριο. Επίσης αναμένεται, στην περίπτωση που υπήρχε ο αναγκαίος χώρος για να αποθηκευτεί ολόκληρο το ευρετήριο του miBLAST στην κύρια μνήμη, ότι η απόδοση του θα ήταν ακόμα καλύτερη.

1.4.2.2 Επίδραση του μεγέθους λέξης ως παράμετρος του BLAST

Εδώ θα εξεταστεί η επίδραση του μεγέθους (μήκους) λέξης ως παράμετρος του ερωτήματος, η οποία αποτελεί μία από τις κύριες παραμέτρους του BLAST.



Γράφημα 1.6**:** Η επίδραση του μεγέθους λέξης, ως παράμετρος του BLAST, στην απόδοση κάθε μεθόδου, για ερώτημα με 4000 ακολουθίες

Για το πείραμα αυτό χρησιμοποιήθηκε το ίδιο σύνολο δεδομένων και το ίδιο ερώτημα που χρησιμοποιήθηκαν και στο προηγούμενο, με την διαφορά ότι εδώ θα δοκιμάζονται μεγέθη λέξης από 11 έως 23. Σε κάθε περίπτωση χρησιμοποιήθηκε από τον miBLAST ένα ευρετήριο κατασκευασμένο με μέγεθος λέξης 11. Τα αποτελέσματα αυτών των δοκιμών παρουσιάζονται στο Γράφημα 1.6. Όπως φαίνεται στο γράφημα, η απόδοση του miBLAST βελτιώνεται σημαντικά όσο το μέγεθος λέξης αυξάνεται. Για μέγεθος λέξης 23, ο miBLAST είναι 45.2 (25 bp) και 13.6 (70 bp) φορές ταχύτερος από τον BLAST-B.

Για την κατανόηση του γιατί ένα μεγάλο μέγεθος λέξης ωφελεί τον miBLAST, μετρήθηκε μία τιμή που ονομάζεται λόγος φιλτραρίσματος (filtration ratio). Ο λόγος φιλτραρίσματος ορίζεται σαν τον λόγο του πλήθους των ακολουθιών που αναγνωρίζονται ως ενδεχόμενα αποτελέσματα με βάση την επιτυχία λέξης, προς το πλήθος των ακολουθιών της βάσης [9]. Ο λόγος φιλτραρίσματος μετρά το ποσοστό της βάσης που πρέπει να εξεταστεί έτσι ώστε να παραχθούν οι τελικές στοιχίσεις. Με την χρήση του miBLAST, μια μικρή τιμή του λόγου φιλτραρίσματος οδηγεί σε καλύτερη απόδοση καθώς ένα μικρότερο τμήμα της βάσης χρειάζεται να εξεταστεί κατά την φάση της στοίχισης. Καθώς το μέγεθος λέξης αυξάνεται, η πιθανότητα να υπάρχει επιτυχία λέξης (κοινή λέξη) μειώνεται εκθετικά, έχοντας σαν αποτέλεσμα την εκθετική μείωση της τιμή του λόγου φιλτραρίσματος. Αυτή η επίδραση στην λειτουργία του φιλτραρίσματος που σχετίζεται με το μέγεθος λέξης, καθώς και το αποτέλεσμα της στη απόδοση, αποτυπώνονται στον Πίνακα 1.3.

Πίνακας 1.3: Μέγεθος λέξης και λόγος φιλτραρίσματος

Μέγεθος Λέξης	Μέσος λόγος φιλτραρίσματος	Κόστος Στοίχισης ανά Ακολουθία (sec)
11	0.54964 %	0.30
14	0.03924 %	0.13
17	0.02231 %	0.11
20	0.02115 %	0.09
23	0.02009 %	0.08

Η επίδραση του μεγέθους λέξης ως παράμετρος του BLAST, στον λόγο φιλτραρίσματος και στο κόστος στοίχισης για τον miBLAST. Το ερώτημα που χρησιμοποιήθηκε για την εξαγωγή αυτών των αποτελεσμάτων, είναι 4000 ακολουθίες από το σύνολο δεδομένων της Affymetrix.

Όπως μπορούμε να δούμε στο Γράφημα 1.6, με τον miBLAST ένα μεγαλύτερο μέγεθος λέξης οδηγεί στην μείωση του πλήθους των ακολουθιών που θα ανακτηθούν για να εξεταστούν. Αντίθετα ο κλασικός BLAST αλλά και ο BLAST-B πρέπει να εξετάσουν ολόκληρη την βάση δεδομένων (στην διάρκεια της αναζήτησης κοινών λέξεων), έτσι ο μικρός λόγος φιλτραρίσματος δεν συνεπάγεται με την μείωση του πλήθους των ακολουθιών της βάσης που θα ανακτηθούν. Παρουσιάζεται βέβαια μία μείωση στο πλήθος των στοιχίσεων που υπολογίζονται, ωστόσο ο BLAST και ο BLAST-B καταναλώνουν τον περισσότερο χρόνο εκτέλεσης τους, στην λειτουργία εύρεσης κοινών λέξεων (word hit generation). Κατά συνέπεια, η συνολική μείωση του χρόνου εκτέλεσης κατά την αύξηση του μέγεθος λέξης, είναι πολύ μικρή για αυτές τις δύο μεθόδους.

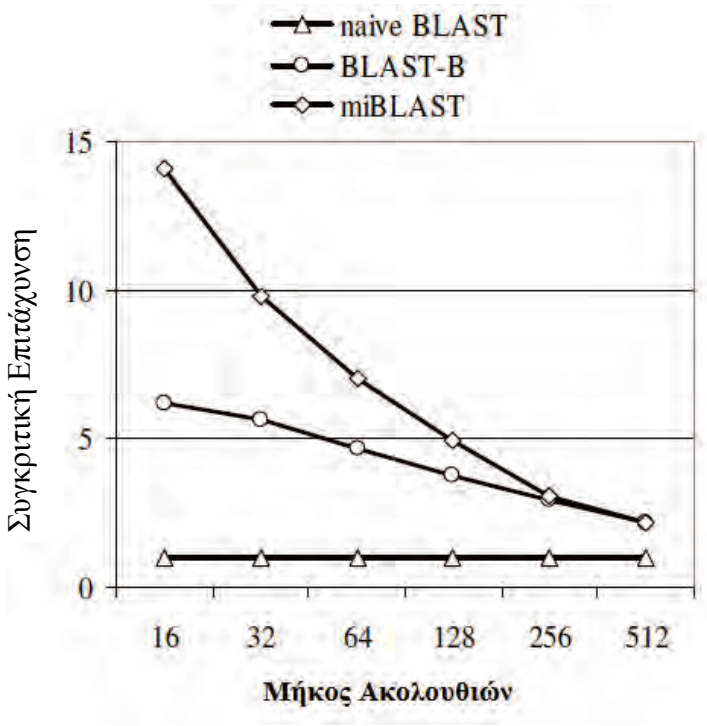
Πίνακας 1.4: Χρόνος εκτέλεσης (sec) ανά ακολουθία για διάφορα μεγέθη λέξης

Μέγεθος Λέξης	naive BLAST	BLAST-B	miBLAST
11	9.50	1.95	0.44
14	9.26	1.61	0.38
17	9.46	1.58	0.37
20	9.45	1.58	0.30
23	9.44	1.56	0.21

Ο μέσος χρόνος εκτέλεσης ανά ακολουθία για τα αποτελέσματα που παρουσιάζονται στο Γράφημα 1.6 (α), για ερώτημα με 4000 ακολουθίες.

Θα μπορούσε κάποιος να παρατηρήσει στον Πίνακα 1.4 ότι η αύξηση του μεγέθους λέξης δεν έχει σημαντική επίδραση στην απόδοση του κλασικού BLAST και του BLAST-B. Ενώ αυτό το αποτέλεσμα θα μπορούσε να φανεί αντίθετο στην προδιάθεση για βελτίωση της απόδοσης όσο μεγαλύτερο είναι το μέγεθος λέξης, αποδεικνύεται ότι στην περίπτωση της ανάγνωσης νουκλεοτιδικών ακολουθιών, αυξάνοντας το μέγεθος λέξης δεν έχει κάποια σημαντική επίδραση στην απόδοση των BLAST και BLAST-B εξαιτίας του τρόπου αναπαράστασης των ακολουθιών της βάσης σε bytes και των αλληλεπιδράσεων του τρόπου αυτού με την επεξεργασία βάσει του μήκους λέξης. Αυξάνοντας την παράμετρο του BLAST, μέγεθος λέξης, μπορεί να έχουμε αμελητέα επίδραση στην απόδοση, καθώς είναι πιθανόν ο επεξεργαστής να εκτελεί ακόμα και ισοδύναμη εργασία, επειδή ανακτά και επεξεργάζεται δεδομένα, τα οποία είναι δομημένα σε blocks με μεγέθη καθορισμένα

από την θεμελιώδη αρχιτεκτονική του υπολογιστή. Στην πραγματικότητα, σε κάποιες περιπτώσεις η αύξηση του μεγέθους μπορεί πράγματι να επιφέρει μια μικρή μείωση στην απόδοση (για παράδειγμα όταν ένα μεγαλύτερο μέγεθος λέξης για τον BLAST, απαιτεί από τον επεξεργαστή να χειρίζεται ένα μεγαλύτερο πλήθος τμημάτων (blocks) της μνήμης). Αυτό το αποτέλεσμα που έχει η αύξηση του μεγέθους λέξης έχει επίσης αναφερθεί και για τον WU-BLAST.



Γράφημα 1.7: Συγκριτική επιτάχυνση σε σχέση με τον BLAST για διάφορα μήκη ακολουθιών

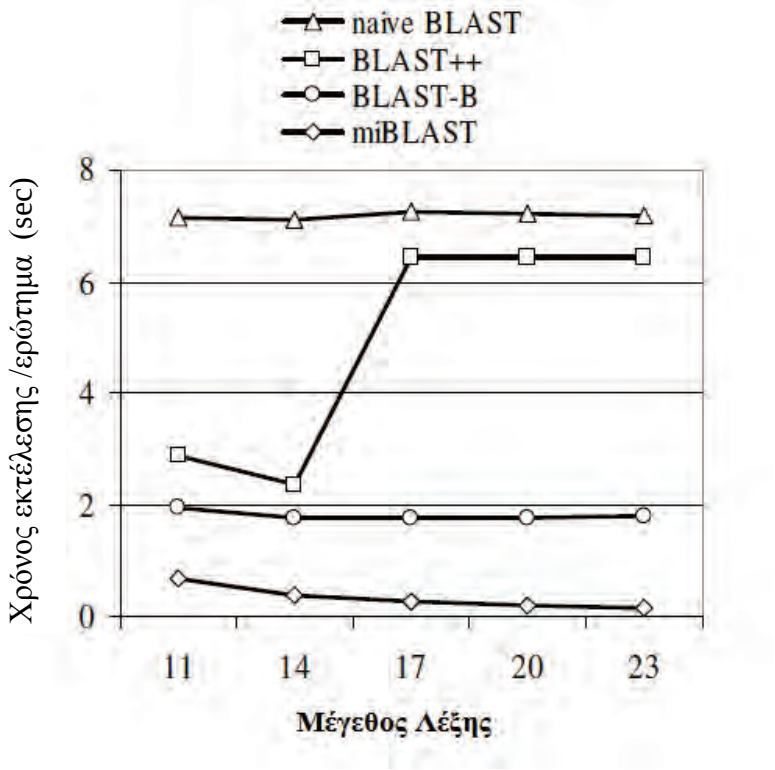
1.4.2.3 Επίδραση του μήκους των ακολουθιών του ερωτήματος

Για την μέτρηση της επίδραση του μήκους των ακολουθιών του ερωτήματος στην απόδοση του miBLAST, χρησιμοποιήθηκαν σύνολα με ακολουθίες από την βάση των EST του ανθρώπου, τα οποία τέθηκαν σαν ερώτημα πάνω στην βάση UniGene. Κάθε σύνολο περιείχε διαφορετικού μήκους ακολουθίες από 16 έως 512 bp. Τα σύνολα αποτελούνταν από 1000 ακολουθίες σταθερού μήκους το καθένα. Τα αποτελέσματα του προέκυψαν (βλέπε Γράφημα 1.7) δείχνουν ότι το κέρδος στην απόδοσης του miBLAST μειώνεται όσο το μήκος των ακολουθιών αυξάνεται. Το

αίτιο αυτής της συμπεριφοράς είναι ότι η απόδοση του miBLAST εξαρτάται άμεσα από τον λόγο φιλτραρίσματος, καθώς πρωτίστως αυτός επιδρά στην επιτάχυνση της λειτουργίας φιλτραρίσματος. Γενικά, ένας μικρός λόγος φιλτραρίσματος επιφέρει καλύτερη σχετική απόδοση για τον miBLAST.

1.4.2.4 Σύγκριση απόδοσης με τον BLAST++

Εδώ παρουσιάζονται τα στοιχεία της σύγκρισης του miBLAST με τον BLAST++. Όπως επισημάνθηκε πιο πριν, η έκδοση του BLAST++ που χρησιμοποιήθηκε δεν μπορούσε να χειριστεί το μέγεθος της βάσης δεδομένων Human Unigene. Έτσι για τις δοκιμές χρησιμοποιήθηκε μόνο το πρώτο μισό της



Γράφημα 1.8: Ο χρόνος εκτέλεσης κάθε μεθόδου, για διάφορα μεγέθη λέξης

Human Unigene. Το σύνολο του ερωτήματος που χρησιμοποιήθηκε είναι οι ολιγονουκλεοτιδικές ακολουθίες (25bp) της Affymetrix. Η διαμόρφωση του BLAST++ είναι παρόμοια με αυτή του BLAST-B. Τα αποτελέσματα της σύγκρισης παρουσιάζονται στο Γράφημα 1.8. Όπως μπορεί να φανεί σε αυτό το γράφημα, ο BLAST++ ξεπερνά σε απόδοση μόνο τον κλασικό BLAST και είναι χειρότερος από

τον miBLAST και τον BLAST-B σε όλες τις περιπτώσεις. Ο miBLAST είναι 5 και 48 φορές γρηγορότερος από τον BLAST++ με μέγεθος λέξης 11 και 23 αντίστοιχα. Η βελτίωση της απόδοσης για τον BLAST++ προέρχεται από την καταγραφή πληροφοριών των ακολουθιών της βάσης για τις κοινές λέξεις που έχουν με το ερώτημα. Ωστόσο, όταν το μέγεθος λέξης είναι μεγάλο, το πλήθος των κοινών λέξεων μειώνεται, και ο BLAST++ καταλήγει να εξετάζει περισσότερες ακολουθίες τις βάσης για κάθε ακολουθία του ερωτήματος.

1.4.3 Παρατηρήσεις

1.4.3.1 Το κόστος κατασκευής και αποθήκευσης του ευρετηρίου.

Το ευρετήριο που χρησιμοποιεί ο miBLAST απαιτεί ($8 \times S^m + 4 \times N$) bytes, όπου S είναι το μέγεθος του αλφαβήτου για τα σύμβολα (4 για τις νουκλεοτιδικές ακολουθίες), m είναι το μήκος λέξης για το ευρετήριο και N είναι ο συνολικός αριθμός των συμβόλων μέσα στη βάση. $8 \times S^m$ bytes χρησιμοποιούνται για την κεφαλίδα του ευρετηρίου και $4 \times N$ bytes χρησιμοποιούνται για την αποθήκευση των δεδομένων με τα ID των ακολουθιών. Ωστόσο, επειδή δεν αποθηκεύονται διπλές τιμές με τα ID των ακολουθιών όταν η ίδια λέξη συναντάται πάνω από μία φορά σε μία ακολουθία, το πραγματικό μέγεθος του ευρετηρίου είναι σημαντικά μικρότερο από αυτό που προκύπτει από τον παραπάνω τύπο. Για την βάση δεδομένων human UniGene build #177, η οποία περιέχει 3.19 gigabases, χρησιμοποιώντας μέγεθος λέξης 11, το ευρετήριο που παράγεται για αυτή την βάση είναι 11.94 GB. Μόνο 32MB αυτού του χώρου απαιτούνται για την κεφαλίδα του ευρετηρίου, το υπόλοιπο τμήμα του χώρου χρησιμοποιείται για την αποθήκευση των ID των ακολουθιών.

Η κατασκευή ενός q-gram ευρετηρίου είναι δαπανηρή για ογκώδεις βάσεις δεδομένων. Ωστόσο, το κόστος κατασκευής του ευρετηρίου είναι ένα εφάπαξ κόστος και επειδή το ίδιο ευρετήριο μπορεί να χρησιμοποιηθεί για πολλά ερωτήματα, το κόστος αυτό μοιράζεται (amortized) σε όλα τα ερωτήματα. Για παράδειγμα, όπως αναφέρεται στην δημοσίευση του miBLAST, ο χρόνος κατασκευής του ευρετηρίου για την Human UniGene ήταν 38 λεπτά στο μηχάνημα

που χρησιμοποιήθηκε. Λαμβάνοντας υπόψη ότι η συλλογή Affymetrix που χειρίστηκε ως ερώτημα περιέχει 247.965 ακολουθίες, το κόστος κατασκευής ευρετηρίου ανά ακολουθία ερωτήματος είναι 0.0092 δευτερόλεπτα (sec), και το κόστος αυτό μπορεί να μειωθεί επιπλέον όταν έχουμε ως ερώτημα μεγαλύτερο σύνολο ακολουθιών ή όταν το ίδιο ευρετήριο χρησιμοποιείται για την εκτέλεση πολλών ερωτημάτων.

1.4.3.2 Βιολογικές εφαρμογές του miBLAST

Σε αυτό το κομμάτι θα γίνει ένας προσδιορισμός των βιολογικών εφαρμογών που μπορούν να επωφεληθούν από τον miBLAST. Τα χαρακτηρίστηκα του miBLAST τον καθιστούν άμεσα εφαρμόσιμο στις περιπτώσεις που γίνεται σύγκριση ενός μεγάλου πλήθους ολιγονουκλεοτιδικών ακολουθιών, ακολουθιών cDNA, ή ESTs έναντι σε μία βάση δεδομένων με ESTs. Παρακάτω επικεντρωνόμαστε σε κάποιες εφαρμογές με αυτά τα χαρακτηριστικά.

Μια σημαντική εφαρμογή είναι η σύγκριση μιας συλλογής ολιγονουκλεοτιδίων, όπως η συλλογή της Affymetrix, έναντι στην βάση UniGene. Οι συλλογές της Affymetrix συγκρίνονται με την πιο πρόσφατη έκδοση της UniGene και η σύγκριση-αναζήτηση αυτή συχνά επαναλαμβάνεται περιοδικά εξ αιτίας των ανανεώσεων που γίνονται στο σύνολο δεδομένων της UniGene. Αυτή η διαδικασία σύγκρισης απαιτείται να έχει υψηλή ευαισθησία, δηλαδή θα πρέπει να αναγνωρίζονται και οι περιπτώσεις που δεν υπάρχει απόλυτη ταύτιση των ακολουθιών, αλλά διαφέρουν σε μερικές βάσεις (mismatches). Έτσι πρέπει να χρησιμοποιηθεί ένα εργαλείο στοίχισης που επιτυγχάνει υψηλή ευαισθησία. Ο BLAST, οποίος χειρίζεται επικαλυπτόμενες λέξεις στην τεχνική αναζήτησης, όταν εκτελείται για μικρές τιμές της παραμέτρου μέγεθος λέξης, φέρνει αποτελέσματα με υψηλή ευαισθησία που είναι κατάλληλα για αυτήν την εφαρμογή. Επειδή αυτή η εργασία πρέπει να επαναλαμβάνεται περιοδικά για ένα μεγάλο πλήθος ολιγονουκλεοτιδίων έναντι σε μια ογκώδης βάση δεδομένων με EST, ο miBLAST είναι μια καλή εναλλακτική του BLAST.

Μια παρόμοια εφαρμογή είναι ο σχεδιασμός νέων συνόλων probe sets για μικρό-ακολουθίες (microarrays) DNA. Ο miBLAST μπορεί να χρησιμοποιηθεί για την σύγκριση –αναζήτηση των νέων συνόλων σε μια βιβλιοθήκη με EST ενός ζωικού είδους σαν ένα πρώτο έλεγχο της ευαισθησίας. Τα αποτελέσματα τις

σύγκρισης μπορούν να χρησιμοποιηθούν για τον σχεδιασμό νέων μικρό-ακολουθιών. Αυτή η εργασία αποτελεί το κύριο βήμα σε προγράμματα που έχουν αναπτυχθεί για τον σχεδιασμό τέτοιου είδους συνόλων [15,16] και ο miBLAST μπορεί να συμβάλλει στην επιτάχυνση του υπολογιστικού μέρους.

Τέλος πρέπει να επισημανθεί ότι ο miBLAST βελτιώνει την αποδοτικότητα των συγκρίσεων μικρών νουκλεοτιδικών ακολουθιών με τα ESTs. Εάν κάποιος ενδιαφέρεται για στοιχίσεις του τύπου, EST με γονιδίωμα ή γονιδίωμα με γονιδίωμα, υπάρχουν άλλες μέθοδοι [2,17,18] που είναι πιθανότατα καταλληλότερες για αυτές τις εργασίες.

1.4.4 Συμπεράσματα για τον miBLAST

Σε αυτήν την ενότητα, έγινε μία παρουσίαση του miBLAST, ενός γρήγορου αλγόριθμου παρόμοιου με τον BLAST, ο οποίος ασχολείται με την αποδοτική στοίχιση ερωτημάτων με πολλές ακολουθίες νουκλεοτιδίων, σε σχέση με μία βάση νουκλεοτιδικών ακολουθιών. Οι υπάρχουσες μέθοδοι για την εκτέλεση τέτοιων ερωτημάτων ουσιαστικά εφαρμόζουν ένα μοντέλο εμφωλευμένων βρόγχων (nested-loops), στο οποίο κάθε ακολουθία του ερωτήματος στοιχίζεται ξεχωριστά με την χρήση του εργαλείου στοίχισης BLAST.

Αυτή η προσέγγιση μπορεί να αποδειχτεί πολύ δαπανηρή, ειδικά στις περιπτώσεις ερωτημάτων που περιέχουν πολλές ακολουθίες. Χρησιμοποιώντας ένα συνδυασμό q-gram ευρετηρίων και ενός αλγορίθμου συνένωσης ευρετηρίων, ο miBLAST καταφέρνει να επιταχύνει δραματικά την εκτέλεση τέτοιων ερωτημάτων. Ο miBLAST είναι ιδιαίτερα αποδοτικός για ερωτήματα που αποτελούνται από μικρές ακολουθίες, όπως συλλογές ολιγονουκλεοτιδίων.

Το εργαλείο miBLAST έχει υλοποιηθεί με την χρήση του NCBI toolkit και εφαρμόζει το ίδιο στατιστικό μοντέλο με τον BLAST και την ίδια μορφοποίηση του αποτελέσματος στην έξοδο, έξη ώστε να είναι οικείος στους χρήστες.

ΚΕΦΑΛΑΙΟ 2

2 Εφαρμογή συμπιεσμένου ευρετηρίου για τον miBLAST

Σε αυτό το κεφάλαιο θα αναφερθούμε στο πως εφαρμόσαμε την συμπίεση για το ευρετήριο που χειρίζεται ο miBLAST. Αρχικά θα παρουσιάσουμε τους αλγορίθμους κωδικοποίησης ακεραίων που χρησιμοποιήσαμε για την συμπίεση. Έπειτα θα αναλύσουμε τους λόγους για τους οποίους επιλέξαμε να κάνουμε συμπίεση στο ευρετήριο και θα περιγράψουμε την μέθοδο που εφαρμόζουμε για την κατασκευή και την ανάγνωση του. Τέλος θα κάνουμε εκτενή αναφορά στο πώς υλοποιήσαμε αυτήν την μέθοδο παρεμβαίνοντας στο κώδικα του miBLAST.

2.1 Αλγόριθμοι κωδικοποίηση ακεραίων

2.1.1 Διαφορική κωδικοποίηση (Delta code)

Στην επιστήμη, στην μηχανική και στα μαθηματικά, το ελληνικό γράμμα δέλτα (Δ) χρησιμοποιείται για να δηλώσει τις μεταβολή της τιμής μίας μεταβλητής. Ο όρος διαφορική κωδικοποίηση αναφέρεται σε διάφορες τεχνικές που αποθηκεύουν τα δεδομένα ως την διαφορά μεταξύ των διαδοχικών τιμών (ή χαρακτήρων), αντί να αποθηκεύουν τις κάθε αυτές τιμές τους. Το αποτέλεσμα αυτής της διαδικασίας είναι η μείωση του χώρου που καταλαμβάνουν τα δεδομένα. Η διαφορική κωδικοποίηση είναι ιδιαίτερα αποτελεσματική όταν τα δεδομένα είναι κάπως «ομαλά», δηλαδή οι γειτονικές τιμές έχουν μικρές διαφορές μεταξύ τους. Η διαφορική κωδικοποίηση έχει εφαρμογές στην κωδικοποίηση ακολουθιών αριθμών, κειμένου [19] και ήχου και χρησιμοποιείται ιδιαίτερα από τα πρωτόκολλα διαδικτύου και από υπηρεσίες δημιουργίας αντιγράφων ασφαλείας (backup).

Πιο συγκεκριμένα για την περίπτωση κωδικοποίησης μιας ακολουθίας ακεραίων αριθμών οι οποίοι βρίσκονται σε αύξουσα σειρά, παρουσιάζουμε το εξής παράδειγμα: Έχουμε την παρακάτω ακολουθία ακέραιων αριθμών

12, 13, 14, 17, 19, 21, 25, 26, 34

Χρησιμοποιώντας την διαφορική κωδικοποίηση για τη παραπάνω ακολουθία προκύπτει το εξής αποτέλεσμα:

12, 1, 1, 3, 2, 2, 4, 1, 8

Παρατηρούμε ότι η πρώτη τιμή της ακολουθίας παραμένει η ίδια όπως ήταν στην αρχική ακολουθία, ενώ οι υπόλοιπες τιμές που κωδικοποιούνται είναι ίσες με την διαφορά της αντίστοιχης τιμής από την προηγούμενη.

Ακολουθεί ένα παράδειγμα εφαρμογής της διαφορικής κωδικοποίησης σε ένα αντίστροφο ευρετήριο για την ανάκτηση κειμένου [20]. Για κάθε λέξη που εμφανίζεται στην συλλογή των δεδομένων, κατασκευάζεται μία λίστα με τους αύξοντες αριθμούς των εγγράφων που την περιέχουν. Για μία λέξη *t*, η λίστα στο ευρετήριο είναι

$$\{f_t : d_1, d_2, d_3, \ldots, d_{ft}\}$$

Όπου f_t είναι ο αριθμός των αρχείων που περιέχουν την λέξη *t* και d_i είναι ένας ακέραιος που αντιπροσωπεύει ένα αρχείο.

Χρησιμοποιώντας την διαφορική κωδικοποίηση, η παραπάνω λίστα παίρνει την εξής μορφή:

$$\{f_t : d_1, d_2 - d_1, d_3 - d_2, \ldots, d_{ft} - d_{ft-1}\}$$

2.1.2 Κώδικες Elias [21]

2.1.2.1 Elias gamma code

Ο Elias gamma code είναι ένας κώδικας γενικής χρήσης για την κωδικοποίηση θετικών ακεραίων αριθμών, ο οποίος λαμβάνει ως δεδομένο ότι οι ακέραιοι με μικρότερη τιμή είναι πιο πιθανό να εμφανίζονται στα δεδομένα. Στην ουσία επωφελείται από την αναλογικά πτωτική κατανομή. Οι τιμές που χρησιμοποιούν *ν* bits πρέπει να είναι δύο φορές πιο πιθανές από τις τιμές που χρησιμοποιούν *ν*+1 bits.

Ο Elias gamma code είναι ο απλούστερος από τους κώδικες του Elias και ορίζεται ως εξής. Για την κωδικοποίηση ενός φυσικού αριθμού $x \in \mathrm{N} = \{1, 2, 3, \ldots\}$, στην αρχή του κώδικα θέτουμε $\lfloor \log_2 x \rfloor$ μηδενικά bits ακολουθούμενα από τη

δυαδική αναπαράσταση του x. Επισημαίνεται ότι χρησιμοποιούνται $\lfloor \log_2 x \rfloor + 1$ bits για την δυαδική αναπαράσταση του x. Ουσιαστικά στην μέση του κώδικα πάντα υπάρχει ένα bit με τιμή ένα-1. Το συνολικό μέγεθος του κώδικα σε bits προκύπτει από τον τύπο $2\lfloor \log_2 x \rfloor + 1$.

Πίνακας 2.1: Μερικά παραδείγματα του κώδικα Elias gamma code

x	$C_\gamma(x)$
1	1
2	010
3	011
4	00100
5	00101
6	00110
7	00111
8	0001000
9	0001001
10	0001010
⋮	⋮
19	000010011
⋮	⋮
147	000000010010011

Για παράδειγμα, για $x = 9$, η δυαδική αναπαράσταση του είναι 1001 και τα μηδενικά πριν από αυτή είναι $\lfloor \log_2 9 \rfloor = 3$, έτσι ο κώδικας Elias gamma code που προκύπτει είναι $C_\gamma(9) = 0001001$.

Ο κώδικας Elias gamma code μπορεί να αποκωδικοποιηθεί άμεσα χωρίς την χρήση πινάκων ή πολύπλοκων συναρτήσεων. Η διαδικασία αποκωδικοποίησης έχει ως εξής, αρχικά μετριέται το πλήθος v των μηδενικών bits στην αρχή του κώδικα. Εάν το v είναι μηδέν, η τιμή που αποκωδικοποιείται είναι το ένα, αλλιώς ο αποκωδικοποιητής διαβάζει τα υπόλοιπα v +1 ψηφία και αποκωδικοποιεί τον αντίστοιχο δυαδικό αριθμό.

Ο κώδικας Elias gamma code είναι ιδιαίτερα αποδοτικός για μικρούς ακεραίους, με το βασικό του πλεονέκτημα να είναι ότι κωδικοποιεί την τιμή ένα (1) χρησιμοποιώντας μόνο ένα bit. Σημειώνεται ότι δεν μπορεί να κωδικοποιήσει το μηδέν. Στην περίπτωση που χρειάζεται να κωδικοποιήσουμε μεγαλύτερες τιμές, ο άλλος κώδικας του Elias, ο Elias delta code, είναι πιο κατάλληλος.

2.1.2.2 Elias delta code

Ο Elias delta code είναι κάπως πιο περίπλοκος και ουσιαστικά αποτελεί μια επέκταση του Elias gamma code, τον οποίον χρησιμοποιεί σαν αρχική δομή. Επίσης ο κώδικας που παράγει είναι λίγο μεγαλύτερος για τους μικρούς ακέραιους, αλλά για μεγαλύτερος τιμές ακέραιων η κατάσταση αντιστρέφεται.

Ξεκινάμε παρουσιάζοντας μια τροποποιημένη μορφή του, που την καλούμε $\hat{C}_\delta$. Για την κωδικοποίηση ενός φυσικού αριθμού $x \in N = \{1, 2, 3,...\}$, ο κώδικας $\hat{C}_\delta(x)$ υπολογίζεται ως εξής: γράφουμε τον κώδικα gamma code για την τιμή $\lfloor \log_2 x \rfloor + 1$ δηλ. $C_\gamma(\lfloor \log_2 x \rfloor + 1)$, ακολουθούμενο από την δυαδική αναπαράσταση του x. Υπενθυμίζουμε ότι χρησιμοποιούνται $\lfloor \log_2 x \rfloor + 1$ bits για την αναπαράσταση του x σε δυαδική μορφή.

Για παράδειγμα, για $x = 13$, ο δυαδικός αριθμός είναι 1101˙ υπολογίζοντας $\lfloor \log_2 13 \rfloor = 3$, έχουμε $C_\gamma(4) = 00100$ ˙ τελικά, $\hat{C}_\delta(13) = 001001101$.

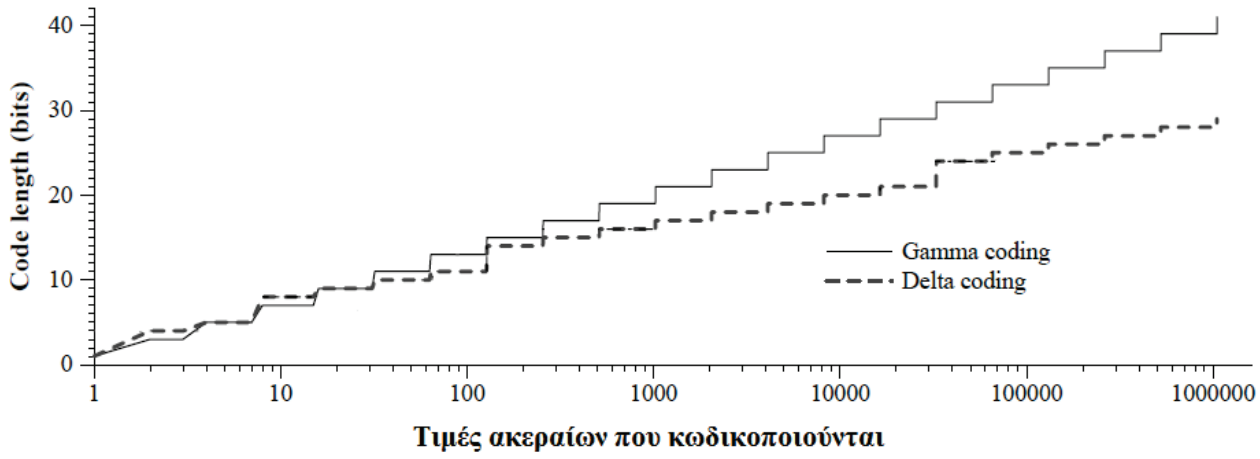
Η τελική έκδοση του Elias delta code, που συμβολίζεται εδώ ως C_δ, προκύπτει από την παρατήρηση ότι το πρώτο ‘1’ στη δυαδική αναπαράσταση του x δεν χρειάζεται, μιας και όλες οι δυαδικές αναπαραστάσεις ξεκινούν με ‘1’. Έτσι στο παράδειγμα από πάνω έχουμε $C_\delta(13) = 00100101$. Το συνολικό μέγεθος του κώδικα σε bits προκύπτει από τον τύπο $\lfloor \log_2 x \rfloor + 2\lfloor \log_2(\lfloor \log_2 x \rfloor + 1) \rfloor + 1$. Σαν ένα άλλο παράδειγμα, θεωρούμε $x = 7$ ˙ ο δυαδικός αριθμός είναι 111˙ έπειτα έχουμε $\lfloor \log_2 3 \rfloor + 1 = 3$ και $C_\gamma(7) = 011$ ˙ τελικά, $C_\delta(7) = 01111$. Στον Πίνακα 2.2 παρουσιάζονται κάποια παραδείγματα της κωδικοποίησης με Elias delta code.

Πίνακας 2.2: Μερικά παραδείγματα του κώδικα Elias delta code

x	$C_\delta(x)$
1	1
2	0100
3	0101
4	01100
5	01101
6	01110
7	01111
8	00100000
9	00100001
10	00100010
⋮	⋮
19	001010011
⋮	⋮
147	00010000010011

Για την αποκωδικοποίηση ˙ αρχικά, αποκωδικοποιούμε τον κώδικα Elias gamma code που βρίσκεται στα πρώτα bits˙ το αποτέλεσμα της αποκωδικοποίησης αυτής παρέχει στον αποκωδικοποιητή την πληροφορία για τον αριθμό των ψηφίων b της δυαδικής αναπαράστασης του κωδικοποιημένου αριθμού. Τελικά ο αποκωδικοποιητής διαβάζει τα υπόλοιπα b-1 ψηφία, εισάγει ένα ‘1’ στην αρχή και αποκωδικοποιεί τον δυαδικό αριθμό που προκύπτει. Σαν παράδειγμα, περιγράφουμε την αποκωδικοποίηση του κώδικα 001010001: πρώτα, για την αποκωδικοποίηση του Elias gamma code στην αρχή, μετράμε το πλήθος των μηδενικών bits, τα οποία είναι δύο ˙ αυτό σημαίνει ότι χρειάζεται να διαβάσουμε τα επόμενα τρία ψηφία, 101˙ μετατρέποντας το 101 σε δεκαδικό παίρνουμε το 5, που σημαίνει ότι ο κωδικοποιημένος αριθμός στην δυαδική του μορφή έχει έξι ψηφία, το πρώτο από τα οποία είναι 1 (έτσι είναι πάντα) και τα υπόλοιπα περιέχονται στο κώδικα, έτσι έχουμε το 10001˙ τελικά μετατρέποντας αυτήν την δυαδική αναπαράσταση σε δεκαδικό προκύπτει το 17.

Συγκρίνοντας τους δύο κώδικες του Elias [22], παρατηρούμε ότι για τις πιο μικρές τιμές ακέραιων (1-15) ο Elias gamma code χρησιμοποιεί λιγότερα bits για την κωδικοποίηση τους. Αντίθετα για μεγαλύτερες τιμές (>30) ο Elias delta code είναι αυτός που χρησιμοποιεί μικρότερο κώδικα. Επίσης κάνοντας του απαραίτητους υπολογισμούς βλέπουμε ότι ο Elias delta code για πολύ μεγάλες τιμές προσεγγίζει την εντροπία, το οποίο σημαίνει ότι είναι ασυμπτωτικά βέλτιστος. Στο Γράφημα 2.1 παρουσιάζεται μία σύγκριση του μήκος του κώδικα που χρησιμοποιούν οι δύο αλγόριθμοι για την κωδικοποίηση ακέραιων με τιμές από 1 έως ένα εκατομμύριο.



Γράφημα 2.1: Το μήκος του κώδικα των Elias gamma code και Elias delta code σε bits για ακεραίους με εύρος τιμών από 1 έως 1 εκατομμύριο.

2.2 Ο λόγος που επιλέξαμε τη συμπίεση

Όπως αναφέρθηκε και κατά την περιγραφή του miBLAST, όταν ο όγκος της βάσης δεδομένων που χρησιμοποιείται είναι αρκετά μεγάλος, τότε το ευρετήριο που παράγει ο miBLAST για αυτήν καταλαμβάνει πολύ χώρο στον δίσκο. Συγκεκριμένα για την έκδοση της βάσης που δοκιμάστηκε, το ευρετήριο που παράχθηκε είχε μέγεθος πάνω από 11 GB. Λαμβάνοντας ως δεδομένο ότι οι βάσεις δεδομένων όπως η UniGene αυξάνουν συνεχώς το μέγεθος τους, συμπεραίνουμε ότι το ευρετήριο που δημιουργεί ο miBLAST για την εκτέλεση ερωτημάτων πάνω σε αυτές τις βάσεις, θα έχει όλο και μεγαλύτερο μέγεθος. Το πρώτο πρόβλημα που δημιουργείται είναι ότι απαιτείται πολύς χώρος στον δίσκο για την αποθήκευση του ευρετηρίου αυτού. Επίσης λόγω του ότι το ευρετήριο βρίσκεται με την μορφή αρχείου στο σκληρό δίσκο, εκτελούνται κάποιες διαδικασίες I/O για την ανάγνωση του, που όσο πιο μεγάλο είναι το ευρετήριο τόσο πιο χρονοβόρα γίνεται αυτή η διαδικασία.

Για να επιτύχουμε την ελάττωση του μεγέθους του ευρετηρίου, ο μόνος τρόπος είναι εφαρμόζοντας κάποιους αλγόριθμους συμπίεσης. Παρόλο που η διαδικασία κωδικοποίησης και αποκωδικοποίησης απαιτεί επιπλέον υπολογιστικό χρόνο, υπολογίζουμε ότι εκτός από το προφανές κέρδος σε χώρο, θα έχουμε επίσης κέρδος και στον χρόνο κατασκευής και ανάγνωσης του ευρετηρίου, επωφελούμενοι από το μειωμένο μέγεθος του. Για την επιλογή των κατάλληλων αλγορίθμων πρέπει να λάβουμε υπόψη την μορφή των δεδομένων που περιέχονται στο q-gram ευρετήριο του miBLAST.

Συνοπτικά, το ευρετήριο αποθηκεύει πληροφορία για το ποίες ακολουθίες της βάσης περιέχουν κάθε λέξη q-gram. Έτσι το ευρετήριο περιέχει μια εγγραφή για κάθε μοναδική *επικαλυπτόμενη* λέξη μήκους *q*. Κάθε εγγραφή του ευρετηρίου αποθηκεύει την λίστα με τα ID των ακολουθιών που περιέχουν την αντίστοιχη λέξη. Αφού οι τιμές των ID είναι ακέραιοι θετικοί αριθμοί, προσανατολιζόμαστε έτσι στη χρησιμοποίηση αλγορίθμων κωδικοποίησης ακεραίων. Επίσης λόγω του ότι η σάρωση της βάσης κατά την δημιουργία του ευρετηρίου γίνεται σειριακά, καταλαβαίνουμε ότι οι τιμές των ID μέσα στην λίστα κάθε εγγραφής θα βρίσκονται σε αύξουσα σειρά. Αυτή η μορφή που έχουν τα δεδομένα ταιριάζει στην χρησιμοποίηση της διαφορικής κωδικοποίησης, όπου αντί να αποθηκεύονται οι τιμές ID κάθε αυτές, θα αποθηκεύεται μόνο η διαφορά κάθε τιμής ID από την προηγούμενη τιμή στην λίστα. Οι τιμές (της διαφοράς) που θα προκύπτουν θα είναι

μικρότερες από τις αρχικές τιμές, έτσι μπορούμε να επωφεληθούμε από την εφαρμογή του Elias gamma code ή του Elias delta code, οι οποίοι χρησιμοποιούν μικρότερο κώδικα όσο πιο μικρές είναι οι τιμές που πρέπει να κωδικοποιήσουν.

Όπως καταλαβαίνουμε, το όφελος από την κωδικοποίηση γίνεται μεγαλύτερο, όσο μικρότερες είναι οι τιμές της διαφοράς μεταξύ των γειτονικών ID στην λίστα. Το γεγονός ότι στην βάση UniGene, οι ακολουθίες που περιέχονται είναι χωρισμένες σε συστάδες βάσει ομοιότητας, μας προδιαθέτει ότι οι γειτονικές ακολουθίες θα έχουν πολλές ίδιες λέξεις. Έτσι και οι τιμές ID που αποθηκεύονται στην λίστα κάθε λέξης, έχουν μεγάλες πιθανότητες να είναι κοντινές. Το ίδιο αποτέλεσμα έχουμε επίσης όσο μικρότερο είναι το μέγεθος λέξης για το οποίο κατασκευάζεται το ευρετήριο, διότι τότε κάθε λέξη περιέχεται σε περισσότερες ακολουθίες.

Για να αποδείξουμε ότι υπάρχει όφελος από την εφαρμογή της κωδικοποίησης, θα παραθέσουμε το ακόλουθο παράδειγμα: Ας υποθέσουμε ότι έχουμε την παρακάτω ακολουθία με IDs.

14, 17, 25, 29, 30, 36, 42

Έχοντας ως δεδομένο ότι η γλώσσα C για την αναπαράσταση ενός ακεραίου, χρησιμοποιεί τύπο δεδομένων μεγέθους 4 bytes (32 bits), θα υπολογίσουμε πόσο χώρο χρειάζεται για να αποθηκεύσει την παραπάνω ακολουθία. Οι ακέραιοι που περιέχονται στην ακολουθία είναι 7, άρα απαιτούνται 28 bytes, δηλαδή (224 bits).
Εφαρμόζοντας την διαφορική κωδικοποίηση προκύπτει η εξής ακολουθία:

14, 3, 8, 4, 1, 6, 12

Έπειτα κωδικοποιούμε τις τιμές αυτές χρησιμοποιώντας τον Elias gamma code.
Για να υπολογίσουμε το συνολικό χώρο που απαιτείται για την αποθήκευση της παραπάνω ακολουθίας, προσθέτουμε τα bit που χρειάζονται για την κωδικοποίηση κάθε ακέραιου: 7 + 3 + 7 + 5 + 1 + 5 + 7 = 40 bits. Το κέρδος που αποκομίζουμε είναι 184 bits.

Συνοψίζοντας, αναφέρουμε ότι έχοντας ως σκοπό να ελαττώσουμε το μέγεθος του ευρετηρίου που παράγει ο miBLAST, εφαρμόσαμε τον αλγόριθμο διαφορικής κωδικοποίησης (Delta code) διαδοχικά με τον Elias gamma code ως πρώτη εκδοχή, αλλά και την διαφορική κωδικοποίηση σε συνδυασμό με τον Elias delta code. Παρόλο που δεν μπορούμε να γνωρίζουμε εξ αρχής το κέρδος που θα επιτύχουμε, υπολογίζουμε ότι χρησιμοποιώντας συμπιεσμένο ευρετήριο για τον miBLAST θα έχουμε όφελος σε χώρο, αλλά και στον χρόνο εκτέλεσης.

2.3 Μέθοδος κατασκευής και ανάγνωσης του Συμπιεσμένου Ευρετηρίου

Σε αυτό το σημείο θα περιγράψουμε την διαδικασία κατασκευής και ανάγνωσης του συμπιεσμένου ευρετηρίου της βάσης δεδομένων, την οποία εφαρμόσαμε στο miBLAST. Θα αναφέρουμε με απλά λόγια τα βήματα που ακολουθούνται. Αναλυτικά η υλοποίηση της μεθόδου αυτής παρουσιάζεται στην επόμενη ενότητα.

Κατά την κατασκευή του ευρετηρίου, διαβάζουμε κάθε ακολουθία της βάσης δεδομένων, διαδοχικά. Μόλις διαβάσουμε κάποια λέξη από μία ακολουθία, τότε πρέπει να προσθέσουμε τον αριθμό ID της ακολουθίας αυτής, στην εγγραφή του ευρετηρίου όπου αποθηκεύονται οι ακολουθίες που περιέχουν την συγκεκριμένη λέξη. Εμείς πριν το κάνουμε αυτό, κωδικοποιούμε πρώτα την τιμή ID της ακολουθίας με την χρήση της διαφορικής κωδικοποίησης. Δηλαδή υπολογίζουμε την διαφορά της συγκεκριμένης τιμής ID, με το ID της προηγούμενης ακολουθίας που περιείχε αυτή την λέξη. Έπειτα κωδικοποιούμε την τιμή της διαφοράς που προκύπτει με τον Elias gamma code, στην πρώτη εκδοχή της μεθόδου και με τον Elias delta code στην δεύτερη. Αυτόν τον κώδικα αποθηκεύουμε τελικά στο ευρετήριο.

Για να ανακτήσουμε τις ακολουθίες της βάσης που περιέχουν μια συγκεκριμένη λέξη, πρέπει να διαβάσουμε την αντίστοιχη εγγραφή του ευρετηρίου. Η αποκωδικοποίηση των τιμών ID που είναι αποθηκευμένες εκεί, πραγματοποιείται με την εξής διαδικασία. Αρχικά αποκωδικοποιούμε την πρώτη τιμή χρησιμοποιώντας τον αλγόριθμο αποκωδικοποίησης του Elias gamma code (ή Elias delta code). Η πρώτη τιμή καταγράφεται όπως έχει στο αποτέλεσμα. Για κάθε επόμενη τιμή που αποκωδικοποιείται με τον αλγόριθμο του Elias, κάνουμε έπειτα την διαφορική αποκωδικοποίηση. Δηλαδή προθέτουμε την τιμή αυτή με την προηγούμενη τιμή ID που εξάγαμε, ώστε να προκύψει η πραγματική τιμή ID της ακολουθίας.

Αυτή είναι σε γενικές γραμμές η λογική της μεθόδου συμπίεσης που εφαρμόσαμε για το ευρετήριο που χρησιμοποιεί ο miBLAST. Αυτήν την μέθοδο την υλοποιήσαμε πάνω στον ήδη υπάρχοντα κώδικα του miBLAST, προθέτοντας τα τμήματα του κώδικα που εκτελούν τις παραπάνω λειτουργίες. Ακολουθεί αναλυτική περιγραφή του τρόπου υλοποίησης.

2.4 Υλοποίηση

Κατά την διαδικασία τροποποίησης του κώδικα του miBLAST, έτσι ώστε να μπορεί να κατασκευάσει και να διαβάσει κωδικοποιημένο ευρετήριο, έπρεπε να αντιμετωπιστούν κάποιες συγκεκριμένες δυσκολίες. Αυτές γίνονται εύκολα κατανοητές μελετώντας τις διαφορές που έχουν τα μη κωδικοποιημένα δεδομένα από τα δεδομένα που έχουν υποστεί κωδικοποίηση με τις μεθόδους που αναλύσαμε παραπάνω. Η βασική τους διαφορά είναι ότι οι κωδικοποιημένοι αριθμοί αναπαριστούνται με κώδικα μεταβλητού μήκους, άρα δεν έχουν σταθερό μέγεθος σε Bits. Αντίθετα, οι αριθμοί που βρίσκονται στην κλασική μορφή ASCII, καταλαμβάνουν συγκεκριμένο χώρο κατά την αποθήκευση τους από το λειτουργικό σύστημα. Έτσι, προκύπτει το πρόβλημα πως όταν θέλουμε να κωδικοποιήσουμε μία ακολουθία ακέραιων, δεν μπορούμε να υπολογίσουμε εξ αρχής τον χώρο που χρειάζεται για να αποθηκευτεί στο δίσκο, γνωρίζοντας απλώς το πλήθος των ακέραιων που περιέχονται στην ακολουθία.

Επισημαίνουμε ότι η διαδικασία συμπίεσης του ευρετηρίου γίνεται κατά την διάρκεια της κατασκευής του και όχι αφότου έχει κατασκευαστεί ήδη από τον miBLAST. Για αυτόν τον λόγο η λειτουργία της συμπίεσης εντάχθηκε μέσα στον κώδικα του miBLAST, ο οποίος είναι γραμμένος σε γλώσσα C. Στο εξής θα παρουσιάσουμε αναλυτικά τις επεμβάσεις που κάναμε στον κώδικα του miBLAST, έτσι ώστε να χειρίζεται συμπιεσμένο ευρετήριο για την βάση δεδομένων.

Το λειτουργικό σύστημα που χρησιμοποιήσαμε είναι η διανομή Ubuntu 11.10 με πύρινα Linux 3.0.0-19, διότι ο υπάρχον κώδικας χρησιμοποιεί βιβλιοθήκες των Linux και επειδή ο miBLAST γενικά έχει σχεδιαστεί για να τρέχει σε περιβάλλον Linux. Η μεταγλώττιση του προγράμματος έγινε με τον μεταγλωττιστή gcc-4.6 μαζί με την εντολή *make*. Για την από-σφαλματοποίηση χρησιμοποιήθηκε το εργαλείο gnu gdb.

Αρχικά πρέπει να αναφέρουμε ότι το ευρετήριο που χρησιμοποιεί ο miBLAST αποθηκεύεται στον δίσκο με την μορφή δύο αρχείων. Το πρώτο αρχείο ονομάζεται αρχείο κεφαλής (head file) και λειτουργεί σαν δείκτης προς το δεύτερο αρχείο, το αρχείο «σώματος» (body file), το οποίο περιέχει όλα το δεδομένα του ευρετηρίου. Πιο συγκεκριμένα, στο αρχείο κεφαλής αποθηκεύεται, για κάθε λέξη, η θέση μέσα στο body file όπου βρίσκεται η λίστα με τα ID των ακολουθιών που την περιέχουν.

Πίνακας 2.3: Παράδειγμα της κατασκευής των δύο αρχείων του ευρετηρίου

- Δεδομένα εισόδου

Ακολουθία 1: AAAC

Ακολουθία 2: AACA

- Λογική μορφή ευρετηρίου (μέγεθος λέξης 3)

AAA	1
AAC	1 2
AAG	
AAT	
ACA	2
:	
:	

- Φυσική αναπαράσταση ευρετηρίου

.head file

	Δείκτης θέσης (σε byte)
AAA	0
AAC	4
AAG	-1
AAT	-1
ACA	12
:	-1
:	-1

.body file

1 1 2 2 .....

Σημείωση: Κάθε ID ακολουθίας καταλαμβάνει χώρο 4 byte

Στον Πίνακα 2.3 υπάρχει ένα παράδειγμα για το πώς κατασκευάζονται τα δύο αρχεία του ευρετήριο και την δομή που έχουν. Έχοντας δύο ακολουθίες σαν είσοδο, αρχικά παρουσιάζεται η λογική μορφή του ευρετηρίου για μέγεθος λέξης 3 και έπειτα η φυσική του αναπαράσταση με την μορφή των δύο αρχείων.

Εμείς για να έχουμε συμπιεσμένο ευρετήριο, πρέπει να κωδικοποιήσουμε τις τιμές ID που αποθηκεύονται στο body file. Για την αναπαράσταση αυτών των τιμών, το πρόγραμμα του miBLAST χρησιμοποιεί τύπο δεδομένων για ακέραιους (int), οποίος έχει σταθερό μέγεθος 4 bytes. Εφαρμόζοντας κωδικοποίηση, ο κώδικας που θα παράγεται για κάθε τιμή ID θα έχει διαφορετικό μέγεθος, άρα δεν μπορούμε να τον αναθέσουμε κατευθείαν σε ένα τύπο δεδομένων. Για αυτόν τον λόγο θα πρέπει να γράφουμε κάθε bit του κώδικα ξεχωριστά, δημιουργώντας έτσι έναν μονοδιάστατο πίνακα από bits. Επειδή στην γλώσσα C δεν υπάρχει η δυνατότητα να κατασκευαστεί με άμεσο τρόπο ένας πίνακας από bit, αυτό το επιτυγχάνουμε χρησιμοποιώντας ένα πίνακα τύπου char (χαρακτήρων) όπου κάθε μεταβλητή char έχει μέγεθος 1 byte (8 bits). Έτσι πρέπει να δουλεύουμε σε επίπεδο bit κάνοντας χρήση των αντίστοιχων τελεστών.

Η διαδικασία κατασκευής του ευρετηρίου ξεκινά με την δημιουργία του αρχείου κεφαλής (head file). Αρχικά ο miBLAST σαρώνει όλη η βάση μία φορά, με σκοπό να καταγράψει, για κάθε λέξη, το πλήθος των ακολουθιών που την περιέχουν. Δηλαδή μετράει το πλήθος των IDs που αποθηκεύονται στη λίστα κάθε λέξης. Αυτή η διαδικασία γίνεται για τον υπολογισμό της θέσης μέσα στο body file, όπου ξεκινά η λίστα με τα IDs για κάθε λέξη. Όπως είδαμε στο παράδειγμα, οι λίστες με τα IDs αποθηκεύονται διαδοχικά, βάσει αλφαβητικής σειράς των λέξεων που αντιστοιχούν, χωρίς να χωρίζονται μεταξύ τους. Ο miBLAST γνωρίζοντας απλώς το πλήθος των IDs που αντιστοιχούν σε κάθε λέξη, μπορεί εύκολα να υπολογίσει τις θέσεις αυτές, αφού κάθε τιμή ID καταλαμβάνει σταθερό χώρο. Αντίθετα εμείς που κωδικοποιούμε τα IDs, πρέπει να μετρήσουμε πρώτα το χώρο που απαιτείται για την αποθήκευση τους, ώστε να μπορούμε υπολογίσουμε τη θέση κάθε λίστας. Για αυτόν τον λόγο χρησιμοποιούμε έναν πίνακα με τόσες θέσεις όσες είναι οι διαφορικές λέξεις που περιέχονται στο ευρετήριο, όπου σε κάθε θέση του πίνακα γράφουμε πόσα bits καταλαμβάνει η λίστα τις αντίστοιχης λέξης.

Στο head file αποθηκεύονται επίσης κάποιες πληροφορίες για το ευρετήριο, όπως το συνολικό του μέγεθος και το πλήθος των διαφορετικών λέξεων που περιέχει ως εγγραφές. Το πλήθος των διαφορετικών λέξεων υπολογίζεται με την πράξη 4^m,

όπου *m* είναι το μέγεθος λέξης για το οποίο κατασκευάζεται το ευρετήριο και το 4 είναι το πλήθος των διαφορετικών συμβόλων (νουκλεοτιδικών βάσεων). Έπειτα ο miBLAST προχωρά στην κατασκευή του body file του ευρετηρίου.

Επειδή συνήθως το ευρετήριο της βάσης έχει μέγεθος αρκετά GB, είναι προφανές πως δεν μπορεί να φορτωθεί ολόκληρο στην κύρια μνήμη του συστήματος. Για αυτόν το λόγο η κατασκευή και η ανάγνωση του ευρετηρίου του πραγματοποιείται με την χρήση ενός buffer, στον οποίο φορτώνεται κάθε φορά ένα τμήμα (block) του ευρετηρίου. Ο miBLAST χρησιμοποιεί ως buffer έναν πίνακα ακεραίων, μιας και εκεί αποθηκεύονται τιμές ID. Εμείς όπως αναφέραμε παραπάνω χρησιμοποιούμε έναν πίνακα χαρακτήρων, στον οποίο γράφουμε ένα-ένα τα bits κάνοντας χρήση των τελεστών μετακίνησης bit (shift) και των τελεστών λογικής σύγκρισης σε επίπεδο bit (and και or). Με την δική μας προσέγγιση μια ακόμα διαφορά είναι ότι σε κάθε block του ευρετηρίου δεν περιέχεται σταθερός αριθμός IDs, λόγω μεταβλητού κώδικα. Επίσης στο τέλος ενός block μπορεί να βρίσκεται ο μίσος κώδικας μίας τιμής ID και ο υπόλοιπος να είναι στην αρχή του επόμενου block. Αυτό το γεγονός δημιουργεί κάποιες δυσκολίες οι οποίες αντιμετωπίζονται προσθέτοντας κάποιες λειτουργίες ελέγχου.

Περιγράφοντας πιο αναλυτικά την διαδικασία κατασκευής του body file του ευρετηρίου, αναφέρουμε ότι ο miBLAST πρέπει να εκτελέσει τόσες σαρώσεις τις βάσεις όσο το πλήθος των blocks στα οποία χωρίζεται το ευρετηρίου. Το μέγεθος του block για την διαδικασία κατασκευής έχει οριστεί στα 32 MB. Στην πρώτη σάρωση της βάσης κατασκευάζει το πρώτο block του ευρετηρίου, στην δεύτερη το δεύτερο block και ούτω κάθε εξής. Αυτό συμβαίνει έτσι ώστε τα IDs των ακολουθιών που αντιστοιχούν σε κάθε λέξη να αποθηκεύονται διαδοχικά. Επίσης είναι προφανές ότι σε κάθε block περιέχονται οι εγγραφές συγκριμένων λέξεων. Για αυτό τον λόγο, πριν την έναρξη της κατασκευής ενός block, υπολογίζεται ποία είναι η πρώτη και ποία η τελευταία λέξη από αυτές των οποίων οι εγγραφές ID θα περιέχονται στο συγκεκριμένο block του ευρετηρίου. Υπενθυμίζουμε ότι οι λέξεις αντιπροσωπεύονται με αλφαβητική σειρά μέσα στο ευρετήριο, όπως φαίνεται και στο παράδειγμα.

Σε κάθε σάρωση της βάσης, μόλις αναγνωστεί μια λέξη, ελέγχεται αρχικά εάν οι εγγραφές της θα περιέχονται στο παρών Block. Εάν ναι, τότε το ID της ακολουθίας που περιέχει την λέξη προστίθεται στην θέση μέσα στο body file, όπου καταγράφονται τα IDs που αντιστοιχούν στην συγκεκριμένη λέξη.

Εμείς δεν αποθηκεύουμε αυτούσιο το ID της ακολουθίας, αλλά το κωδικοποιούμε πρώτα χρησιμοποιώντας διαφορική κωδικοποίηση. Για να εφαρμόσουμε την διαφορική κωδικοποίηση, πρέπει να γνωρίζουμε την τιμή ID της προηγούμενης ακολουθίας που περιείχε την συγκεκριμένη λέξη. Αυτό το πετυχαίνουμε καταγράφοντας, για κάθε λέξη, την τιμή ID της τελευταίας ακολουθίας που την συναντήσαμε. Έτσι αφαιρούμε την τρέχουσα τιμή ID από την προηγούμενη (τρέχον τελευταία), εκτός βέβαια εάν είναι η πρώτη τιμή της λίστας. Έπειτα κωδικοποιούμε την διαφορά τους με την χρήση του Elias gamma code στην μία εκδοχή του προγράμματος, και με Elias delta code στην άλλη. Η αρίθμηση των ακολουθιών ξεκινά από το 0, αλλά οι κώδικες του Elias δεν μπορούν να κωδικοποιήσουν την τιμή 0. Για αυτό τον λόγο πριν κωδικοποιήσουμε μία τιμή προσθέτουμε +1. Αντίστοιχα στην αποκωδικοποίηση αφαιρούμε -1.

Ο miBLAST υπολογίζει την θέση (σε bytes) μέσα στο block, όπου θα αποθηκευτεί η τιμή ID, γνωρίζοντας την θέση που ξεκινούν οι εγγραφές για την λέξη που εντόπισε, καθώς και το πλήθος των εγγραφών ID που έχει αποθηκεύσει μέχρι τώρα για την συγκεκριμένη λέξη. Επειδή εμείς δεν έχουμε σταθερό κώδικα για κάθε τιμή ID, αντί να μετράμε το πλήθος των ID, προσθέτουμε τα bit που έχουμε γράψει ως τώρα για κάθε λέξη, έτσι ώστε να μπορούμε να υπολογίσουμε την θέση (σε bits) όπου θα γράψουμε μία τιμή ID. Επίσης πρέπει να ελέγχουμε εάν ο κώδικας για μία τιμή ID χωράει ολόκληρος στο παρόν block, ή εάν πρέπει να συνεχίζεται στο επόμενο block, καθώς το σημείο στο οποίο χωρίζεται.

Παραθέτουμε κάποιες γραμμές του κώδικα που σχετίζονται με την κωδικοποίηση των τιμών ID και την καταγραφή τους στο ευρετήριο.

```
//πίνακας που λειτουργεί ως buffer για την κατασκευή των block του ευρετηρίου
ptrWordPosition = (char *)malloc(indexInfo->blockSize);

//βοηθητικός πίνακας
//[3*i] αποθηκεύει πόσα bits έχουν χρησιμοποιηθεί μέχρι τώρα για κάθε λέξη
//[3*i+1] αποθηκεύεται η τελευταία τιμή ID
//[3*i+2] αποθηκεύει την αρχική θέση της λίστας
WordAddressType* tmpWordIndexInfo = (int *)malloc(3*indexInfo->wordIndexCounterSize
*sizeof(WordAddressType));
.
.
.
//υπολογίζουμε την θέση στο ευρετήριο (σε bits) που θα αποθηκεύσουμε το ID.
calculatedAddress= tmpWordIndexInfo[3*encodedWord+2] + tmpWordIndexInfo[3*encodedWord];
```

```
//διαφορική κωδικοποίηση ενός ID
if ( tmpWordIndexInfo[3*encodedWord+1] == NO_SUCH_WORD)
    last = 0; //όταν συναντάμε πρώτη φορά την λέξη
else
    last = tmpWordIndexInfo[3*encodedWord+1];//ID της προηγούμενης ακολουθίας που
περιείχε την ίδια λέξη

delta = seqReadInfo.curSeqNo - last;

//ενημερώνουμε την τελευταία τιμή
tmpWordIndexInfo[3*encodedWord+1] = seqReadInfo.curSeqNo;

delta++;

//πόσα bits χρειάζονται για την κωδικοποίηση της διαφοράς με Elias gamma code
l=log2(delta);
elg=2*l+1;

//θέση στο block
int t = calculatedAddress - startingBlockAddr;

//εάν χωράει ολόκληρος ο κώδικας του id στο Block
if ( t + elg <= indexInfo->blockSize*8)
{
   int x=delta;

    ///κωδικοποίηση με Elias gamma code///
    l = log2(x); //πληθος 0 στην αρχή του κώδικα

    for (a=0; a<l; a++)
    {
       ptrWordPosition[t/8] &= ~(0x01<<(t%8)); //θέτουμε τα αρχικά bit 0
       t++;
    }

    ptrWordPosition[t/8] |= 0x01 << (t%8); //ο 1 μετα τα 0
    t++;

    for (a=l-1; a >= 0; a--)
    {
       if (x & 1 << a)
          ptrWordPosition[t/8] |= 0x01 << (t%8);
       else
          ptrWordPosition[t/8] &= ~(0x01<<(t%8));
       t++;
    }
    ///
    curBitsum+=elg; // πόσα bit έχουν γραφτεί στο block
    }

else //εάν δεν χωράει ολόκληρος ο κώδικας του id στο Block
{
    u = t + elg - indexInfo->blockSize*8; //Ποσά bit περισσεύουν
    v = ceil ( (double)u/8); //θέσεις πίνακα
    upolArray = (char *)malloc(v); //πίνακας που αποθηκεύει το υπόλοιπο του κώδικα,
το οποίο θα μεταφερθεί στο επόμενο Block

// αντίστοιχα η κωδικοποίηση με Elias gamma code
    l = log2(x); s=0;
    for (a=0 ; a<l; a++)
    {
       //εάν το τρέχον bit χωράει στο Block τότε το αποθηκεύουμε
       if ( t < indexInfo->blockSize*8 )
```

```
         {
            ptrWordPosition[t/8] &= ~(0x01<<(t%8));
            t++;
         }
         else //αλλιώς γράφουμε το bit στον πίνακα του υπολοίπου
         {
            upolArray[s/8] &= ~(0x01<<(s%8));
            s++;
         }
      }
      .
      .
      .
}
curBitsum+=elg-u;
.

//ποσά bits έχουμε χρησιμοποιήσει ως τώρα στην εγγραφή (λίστα) της λέξης
tmpWordIndexInfo[3*encodedWord] +=elg;
```

Ανάγνωση του ευρετηρίου γίνεται όταν θέλουμε να ανακτήσουμε την λίστα με τα IDs των ακολουθιών που περιέχουν μία λέξη. Οι λέξεις αναζητούνται στο ευρετήριο, κατά την εκτέλεση ενός ερωτήματος, βάσει αλφαβητικής σειράς. Όπως έχουμε αναφέρει και νωρίτερα, οι εγγραφές (λίστες) των λέξεων είναι αποθηκευμένες μέσα στο ευρετήριο επίσης βάσει αλφαβητικής σειράς. Άρα η ανάγνωση του ευρετηρίου γίνεται σχεδόν σειριακά, πράγμα που είναι πιο αποδοτικό σε σχέση με τις τυχαίες προσπελάσεις. Η ανάγνωση του ευρετηρίου γίνεται και αυτή με την χρήση ενός buffer, στο οποίο φορτώνεται από τον δίσκο κάθε block του ευρετηρίου σειριακά. Το μέγεθος του block για την ανάγνωση έχει οριστεί στα 64 MB.

Όταν φορτώνεται ένα block στην μνήμη, αρχικά αναγνωρίζεται η πρώτη και η τελευταία λέξη των οποίων δεδομένα βρίσκονται στο συγκεκριμένο block. Έτσι όταν θέλουμε να ανακτήσουμε την λίστα με τα IDs για μία λέξη, μπορούμε άμεσα να εξακριβώσουμε εάν πρέπει να τα αναζητήσουμε στο παρόν block ή αν πρέπει να φορτώσουμε το επόμενο block.

Για την εύρεση της ακριβής θέσης μέσα στο body file όπου ξεκινά η λίστα με τα IDs που περιέχουν την λέξη, η πληροφορία δίνεται από το head file του ευρετηρίου. Ο miBLAST γνωρίζοντας τον συνολικό χώρο που καταλαμβάνουν αυτά τα IDs, μπορεί εύκολα να υπολογίσει και το πλήθος τους, μιας και κάθε τιμή έχει σταθερό μέγεθος. Αντίθετα εμείς που χρησιμοποιούμε κωδικοποιημένες τιμές ID, δεν μπορούμε να ξέρουμε εξ αρχής το πλήθος των IDs που αντιστοιχούν στην λέξη, λόγω του ότι κάθε τιμή αναπαριστάται με κώδικα μεταβλητού μήκους. Για να

αντιμετωπιστεί αυτή η δυσκολία, χρησιμοποιούμε ελέγχους για το πόσα Bits έχουμε αποκωδικοποιήσει μέχρι μία δεδομένη χρονική στιγμή. Επίσης πραγματοποιούμε έλεγχο αν οι εγγραφές της λέξης συνεχίζονται και στο επόμενο block, αλλά και για το εάν ο κώδικας μίας τιμής ID δεν είναι ολόκληρος στο παρόν block, δηλαδή ένα τμήμα του βρίσκεται στην αρχή του επόμενου block.

Η αποκωδικοποίηση του τιμών ID γίνεται χρησιμοποιώντας αρχικά τον αλγόριθμο αποκωδικοποίησης του Elias gamma code στην πρώτη εκδοχή του προγράμματος, ή του Elias delta code στην δεύτερη. Η τιμή που παίρνουμε προστίθεται έπειτα με την προηγούμενο ID που αποκωδικοποιήσαμε, έτσι ώστε να γίνει η διαφορική αποκωδικοποίηση και να προκύψει το ID της ακολουθίας σε μορφή ακεραίου. Όλα αυτά τα IDs αποθηκεύονται σε ένα πίνακα ακέραιων, οποίος επιστρέφεται στην αρχική συνάρτηση για την εκτέλεση της συνένωσης με το ευρετήριο του ερωτήματος.

Θα παρουσιάσουμε τώρα κάποια χαρακτηριστικά τμήματα του κώδικα που εκτελεί την ανάκτηση και αποκωδικοποίηση των εγγραφών του ευρετηρίου.

```
//από ποια θέση (σε bits) του block, ξεκινούν οι εγγραφές της λέξης που ψάχνουμε
if ( theWord != indexInfo->curStartingWordInIndex )
   begReadingOffset=indexInfo->wordIndexAddress[theWord] - startBlockAddr;
else
   begReadingOffset = 0;

//που τελειώνουν μέσα στο block
endReadingOffset = readEndingAddr - startBlockAddr;
t=begReadingOffset;

//εάν είναι η τελευταία λέξη που περιέχει εγγραφές σε αυτό το block και δεν βρισκόμαστε στο τελευταίο block υπάρχει πιθανότητα στο τέλος να υπάρχει ανολοκλήρωτος κώδικας(δηλαδή να συνεχίζεται στο επόμενο block)
if ( nextValidEntryIdx == NO_SUCH_WORD && indexInfo->currentNumOfReading < indexInfo->numOfReading-1)
{
   while (t <= endReadingOffset)
   {
  ///ξεκινάει η αποκωδικοποίηση ενός ID///
      ts=t;
      //αποκωδικοποίηση Elias gamma code//
      numberBits = 0;
      while ( !(indexInfo->sWordIndexBit[t/8] & 0x01 << (t%8)) && t-1 <= endReadingOffset)
      {
          numberBits++;
          t++;
      }
      int current = 0;

      t++;
```

```
   for (a=numberBits-1; a >= 0; a--)
   {
       if( indexInfo->sWordIndexBit[t/8] & 0x01 << (t%8) )
          current |= 1 << a;
       t++;
   }
   current |= 1 << numberBits;
   //
   current--;
   //εάν έχουμε διαβάσει πέρα από τα όρια σημαίνει ότι δεν είναι ολόκληρος ο κώδικας του
   Elias gamma
   if (t-1 > endReadingOffset)
   {
       //αντιγράφουμε την αρχή του ανολοκλήρωτου κώδικα σε έναν πίνακα ώστε να μπορούμε
   έπειτα να τον συνδέσουμε με τον υπόλοιπο που βρίσκεται στο επόμενο block
       indexInfo->elBits= endReadingOffset - ts +1;
       indexInfo->upolElias = (char *)malloc( ceil( (double)indexInfo->elBits/8 ) );
       int c;
       for (c=0; c<indexInfo->elBits; c++,ts++)
       {
           if( indexInfo->sWordIndexBit[ts/8] & 0x01 << (ts%8) ) //εαν είναι 1
              indexInfo->upolElias[c/8] |= 0x01 << (c%8); //θέτουμε 1
           else
              indexInfo->upolElias[c/8] &= ~(0x01<<(c%8)); //θέτουμε 0
       }
       break;
   }
   //αποκωδικοποίηση διαφορικής κωδικοποίησης (delta code)
   tempPos += current;

   *((*wordList)+j) = tempPos;//εγγραφή του id στον πίνακα ακεραίων
   j++;
   numWordPrinting++;
}
else //εάν είναι η τελευταία λέξη που περιέχει εγγραφές σε αυτό το block
{
   //εάν είναι η αρχική λέξη στο block
   if ( theWord == indexInfo->curStartingWordInIndex )
   {
   .
   .
   .
```

Συνοψίζοντας, αναφέρουμε ότι διαμορφώσαμε τον κώδικα του miBLAST έτσι ώστε να κατασκευάζει και να διαβάζει συμπιεσμένο ευρετήριο για την βάση δεδομένων. Χρειάστηκε λοιπόν να υλοποιήσουμε σε γλώσσα C τους αλγόριθμους κωδικοποίησης ακεραίων Elias gamma code, Elias delta code και την διαφορική κωδικοποίηση. Κατά την εφαρμογή της κωδικοποίηση στον miBLAST, αντιμετωπίστηκαν κάποιες δυσκολίες οι οποίες προέρχονταν κυρίως από το γεγονός ότι είχαμε να χειριστούμε μεταβλητού μήκους κώδικα αναπαράστασης για τα δεδομένα του ευρετηρίου.

ΚΕΦΑΛΑΙΟ 3

3 Αποτελέσματα Δοκιμών

Για να αποδείξουμε ότι με την εφαρμογή του συμπιεσμένου ευρετηρίου επιτυγχάνουμε βελτίωση της απόδοσης του miBLAST, εκτελέσαμε κάποιες χαρακτηριστικές δοκιμές. Ο σκοπός μας είναι να επιβεβαιώσουμε ότι μειώνουμε τον χώρο που καταλαμβάνει το ευρετήριο στον δίσκο, αλλά και να εξακριβώσουμε εάν επιτυγχάνουμε επιτάχυνση του χρόνο εκτέλεσης, και σε ποιες περιπτώσεις.

Για όλες τις δοκιμές χρησιμοποιήθηκε η τελευταία έκδοση της βάσης δεδομένων UniGene, η NCBI Human UniGene Build #231, η οποία περιέχει τα ESTs του ανθρώπου ταξινομημένα σε συστάδες. Η συγκεκριμένη βάση δεδομένων περιέχει 6.877.387 ακολουθίες. Πληροφορικά, ένα EST (expressed sequence tag) είναι μία μικρή υπό-ακολουθία μιας ακολουθίας cDNA. Τα ESTs παίζουν καθοριστικό ρόλο στην μελέτη του γονιδιώματος και στον προσδιορισμό των γονιδιωματικών ακολουθιών.

Πιο συγκεκριμένα, θα συγκρίνουμε το μέγεθος του συμπιεσμένου ευρετηρίου που δημιουργούμε, σε σχέση με το ευρετήριο που παράγει ο miBLAST, το χρόνο κατασκευής του, καθώς και τον χρόνο της λειτουργίας φιλτραρίσματος. Επίσης θα κάνουμε σύγκριση του συνολικού χρόνου που απαιτεί ο τροποποιημένος miBLAST για την εκτέλεση ερωτημάτων στοίχισης, σε σχέση με τον αρχικό miBLAST και τον κλασικό BLAST. Οι ακολουθίες που χρησιμοποιήσαμε ως ερώτημα προήλθαν από την συλλογή της Affymetrix. Εκτελέσαμε δοκιμές για διάφορα μεγέθη λέξης και για ερωτήματα που αποτελούνται από ποικίλου πλήθους ακολουθίες.

Οι δοκιμές έγιναν στον προσωπικό μου υπολογιστή που έχει επεξεργαστή Intel Core2 Duo E7500 στα 2.93 GHz και μνήμη ram 3 GB. Όλα τα προγράμματα τα έτρεξα σε λειτουργικό Ubuntu 11.10 με πύρινα Linux 3.0.0-19. Σίγουρα εάν είχαμε στην διάθεση μας έναν πιο ισχυρό υπολογιστή, οι χρόνοι εκτέλεσης που καταγράφουμε στις δοκιμές θα ήταν αρκετά μικρότεροι.

3.1 Μέγεθος Ευρετηρίου

Εδώ εξετάζουμε το όφελος που έχουμε σε χώρο, χρησιμοποιώντας τον τροποποιημένο miBLAST, που χειρίζεται συμπιεσμένο ευρετήριο. Για αυτό τον λόγο θα συγκρίνουμε το μέγεθος του ευρετηρίου που δημιουργούν οι δύο εκδοχές του τροποποιημένου miBLAST που υλοποιήσαμε, σε σχέση με το μέγεθος του ευρετηρίου του αρχικού miBLAST. Οι δοκιμές έγιναν για την κατασκευή ευρετηρίου της βάσης UniGene με μέγεθος λέξης 7, 9 και 11. Τα αποτελέσματα της σύγκρισης παρουσιάζονται στον Πίνακα 3.1. Επειδή η κατασκευή ευρετηρίου με μέγεθος λέξης μεγαλύτερο από 11 είναι πολύ δαπανηρή, για αυτό τον λόγο και αποφεύγεται. Όπως αναφέραμε και στην περιγραφή του miBLAST, για την εκτέλεση ερωτημάτων σύγκρισης με παράμετρο μεγάλο μέγεθος λέξης (>11), χρησιμοποιείται η τεχνική κυλιόμενου παραθύρου (sliding-window), η οποία επιτρέπει την επαναχρησιμοποίηση ευρετηρίου με μέγεθος λέξης μικρότερο από αυτό του ερωτήματος.

Πίνακας 3.1: Μέγεθος ευρετηρίου για διάφορα μεγέθη λέξης (σε GB)

Μέγεθος Λέξης	miBLAST	miBLAST με Συμπιεσμένο Ευρετήριο	
		1η Εκδοχή Delta code + Elias gamma code	**2η Εκδοχή** Delta code + Elias delta code
7	**15,3**	**2,6**	**-**
9	**16**	**3,6**	**3,57**
11	**16,1**	**4,4**	**4,1**

Πρέπει να σημειώσουμε ότι στην διανομή Ubuntu 11.10 που χρησιμοποιούμε, δεν εφαρμόζεται το κλασικό δυαδικό σύστημα για την αναπαράσταση του χώρου στο σκληρό δίσκο (π.χ. 1 KB = 1024 Bytes). Αντίθετα εφαρμόζεται το δεκαδικό σύστημα, το οποίο ανήκει στο διεθνές σύστημα μονάδων, όπου για παράδειγμα 1 KB = 1000 Bytes και ούτω κάθε εξής.

Όπως φαίνεται στον Πίνακα 3.1, με την εφαρμογή των αλγόριθμων κωδικοποίησης στον miBLAST, παράγουμε σε κάθε περίπτωση πολύ μικρότερο ευρετήριο από αυτό που παράγει ο αρχικός miBLAST. Αυτό συμβαίνει απλούστατα διότι όλες οι τιμές ID των ακολουθιών που περιέχονται στο ευρετήριο, αναπαριστούνται με λιγότερα Bits όντας κωδικοποιημένες με τους αλγορίθμους αυτούς. Πιο συγκεκριμένα, κάθε τιμή ID καταλάμβανε αρχικά χώρο 32 Bits, όπως ένας ακέραιος (int), ενώ με τη δική μας μέθοδο χρειαζόμαστε, κατά μέσο όρο, λιγότερα από 9 Bits για την αναπαράσταση των τιμών αυτών. Για την περίπτωση ευρετηρίου με μέγεθος λέξης 7 ειδικότερα, ο μέσος όρος πέφτει στα 5 Bits ανά ID.

Επίσης παρατηρούμε ότι όσο πιο μικρό είναι το μέγεθος λέξης για το οποίο κατασκευάζεται το ευρετήριο, τόσο μικρότερο είναι και το μέγεθος του συμπιεσμένου ευρετηρίου που παράγεται. Αυτό μπορεί να εξηγηθεί λαμβάνοντας υπόψη το γεγονός ότι όσο μικρότερη είναι μια λέξη, τόσο περισσότερες βάσει πιθανοτήτων είναι οι ακολουθίες που την περιέχουν. Έτσι στην λίστα της θα περιέχονται περισσότερες τιμές ID, άρα και η διαφορά μεταξύ των τιμών αυτών, που προκύπτει με την διαφορική κωδικοποίηση, θα είναι πιθανόν μικρότερη. Γνωρίζοντας επίσης ότι όσο πιο μικρές είναι οι διαφορές αυτές που κωδικοποιούνται με τους αλγόριθμους του Elias, τόσο μικρότερος είναι ο κώδικας αναπαράστασης τους ˙ εύκολα καταλαβαίνουμε γιατί με μικρό μέγεθος λέξης η συμπίεση είναι πιο αποτελεσματική. Ένα χαρακτηριστικό που ευνοεί επιπλέον την συμπίεση, είναι ότι στην βάση δεδομένων UniGene, οι ακολουθίες που περιέχει είναι ομαδοποιημένες σε συστάδες (clusters). Αυτό συνεπάγεται ότι οι γειτονικές ακολουθίες είναι αρκετά όμοιες, άρα έχουν αρκετές κοινές λέξεις. Έτσι οι τιμές ID που γράφονται στο ευρετήριο για κάθε λέξη, θα έχουν μικρή διαφορά μεταξύ τους.

Συγκρίνοντας τώρα τις δύο εκδοχές της συμπίεσης ευρετηρίου, βλέπουμε ότι ο συνδυασμός διαφορικής κωδικοποίησης (Delta code) μαζί με Elias delta code, έχει λίγο καλύτερο αποτέλεσμα για το μεγάλο μέγεθος λέξης, σε σχέση με τον συνδυασμό διαφορικής κωδικοποίησης με Elias gamma code. Το αντίθετο συμβαίνει όσο μικραίνει το μέγεθος λέξης. Υπενθυμίζουμε ότι ο Elias delta code για μεγάλους ακεραίους (>30) χρησιμοποιεί λιγότερα Bits κώδικα από τον Elias gamma code. Όπως εξηγήσαμε και στην προηγούμενη παράγραφο, όσο μεγαλύτερο είναι το μέγεθος λέξης για το οποίο κατασκευάζεται το ευρετήριο, τόσο πιο μεγάλες συνολικά είναι και οι τιμές τις διαφοράς μεταξύ των ID. Έτσι μπορούμε να καταλάβουμε γιατί ο Elias delta code επιτυγχάνει πιο αποδοτική συμπίεση για

μέγεθος λέξης 11. Για τον αντίστοιχο λόγο δεν χρησιμοποιήσαμε την $2^{η}$ εκδοχή της συμπίεσης για την κατασκευή ευρετηρίου με μέγεθος λέξης 7.

Το βαθμό του οφέλους που έχουμε σε χώρο, εφαρμόζοντας τους αλγόριθμους κωδικοποίησης ακεραίων για το ευρετήριο του miBLAST, μπορούμε να το κατανοήσουμε καλύτερα παρατηρώντας τον Πίνακα 3.2 και το Γράφημα 3.1.

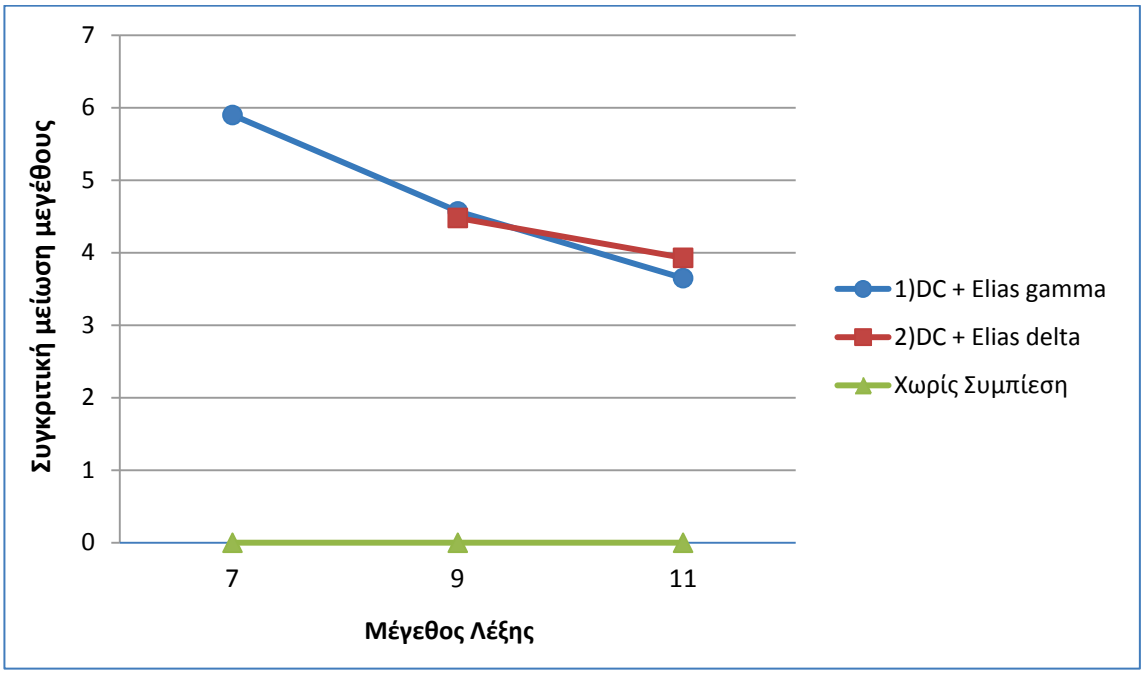
Πίνακας 3.2: Ποσοστό μείωσης μεγέθους ευρετηρίου εφαρμόζοντας συμπίεση

Μέγεθος Λέξης	**$1^{η}$ Εκδοχή** Delta code + Elias gamma code	**$2^{η}$ Εκδοχή** Delta code + Elias delta code
7	83%	-
9	77,5%	77,7%
11	73%	75%

Στον Πίνακα 3.2 καταγράφεται το ποσοστό της μείωσης του μεγέθους του ευρετηρίου χρησιμοποιώντας συμπίεση, σε σχέση με το ευρετήριο που παράγει ο αρχικός miBLAST. Για μέγεθος λέξης 7, το ευρετήριο που δημιουργούμε με την χρήση της διαφορικής κωδικοποίησης μαζί με τον Elias gamma code, είναι 83% μικρότερο από το αρχικό. Για μέγεθος λέξης 9, εφαρμόζοντας πάλι αυτούς τους αλγόριθμους κωδικοποίησης, επιτυγχάνουμε μείωση κατά 77,5%. Την ίδια σχεδόν απόδοση έχουμε και με την δεύτερη εκδοχή συμπίεσης (διαφορική κωδικοποίηση με Elias delta code). Για μέγεθος λέξης 11 ωστόσο, η δεύτερη εκδοχή φαίνεται πιο αποτελεσματική αφού συμπιέζει το ευρετήριο κατά 75%.

Στο Γράφημα 3.1 παρουσιάζεται η συγκριτική μείωση του μεγέθους του ευρετηρίου εφαρμόζοντας τις δύο εκδοχές συμπίεσης. Ουσιαστικά εκφράζεται η πληροφορία που μας δίνουν οι παραπάνω πίνακες, σε γραφική μορφή. Για μέγεθος λέξης 7, χρησιμοποιώντας την διαφορική κωδικοποίηση διαδοχικά με τον Elias gamma code, το ευρετήριο που προκύπτει είναι σχεδόν 6 φορές μικρότερο από το ευρετήριο που δεν έχει υποστεί συμπίεση. Όσο αυξάνεται το μέγεθος λέξης βλέπουμε ότι το συγκριτικό όφελος σε χώρο ελαττώνεται κάπως, αλλά παρόλα αυτά μένει σε υψηλό επίπεδο. Για μέγεθος λέξης 9, και με τις δύο εκδοχές συμπίεσης έχουμε κοντά στις 4,5 φορές μικρότερο ευρετήριο, ενώ για μέγεθος λέξης 11 ο

συνδυασμός διαφορικής κωδικοποίησης με Elias gamma code delta έχει λίγο καλύτερο αποτέλεσμα σε σχέση με την πρώτη εκδοχή, φτάνοντας στις 4 φορές.



Γράφημα 3.1: Συγκριτική μείωση του μεγέθους του ευρετηρίου εφαρμόζοντας συμπίεση

3.2 Χρόνος Κατασκευής Ευρετηρίου

Εκτός βέβαια από το χώρο στο δίσκο που καταλαμβάνει το ευρετήριο της βάσης, πρέπει να εξετάσουμε επίσης και τον χρόνο που απαιτείται για την κατασκευή του, έτσι ώστε να έχουμε μια πιο συνολική εικόνα για αυτό το κομμάτι της λειτουργίας του miBLAST. Για αυτό τον λόγο μετρήσαμε τον χρόνο κατασκευής των ευρετηρίων που παραθέσαμε στην προηγούμενη ενότητα. Η μέτρηση του χρόνο έγινε με την χρήση αντίστοιχης συνάρτησης που διαθέτει η γλώσσα C.

Γνωρίζουμε φυσικά ότι η εφαρμογή της συμπίεσης έχει κάποιο υπολογιστικό κόστος. Αυτό συμβαίνει λόγω της χρησιμοποίησης των προαναφερόμενων αλγορίθμων για την κωδικοποίηση των τιμών που περιέχονται στο ευρετήριο. Ωστόσο μελετώντας τον τρόπο υλοποίησης του miBLAST, συμπεράναμε ότι με ένα μικρότερου μεγέθους ευρετήριο μπορούμε να έχουμε συνολικό όφελος και στον χρόνο κατασκευής.

Πίνακας 3.3: Χρόνος κατασκευής ευρετηρίου (σε sec)

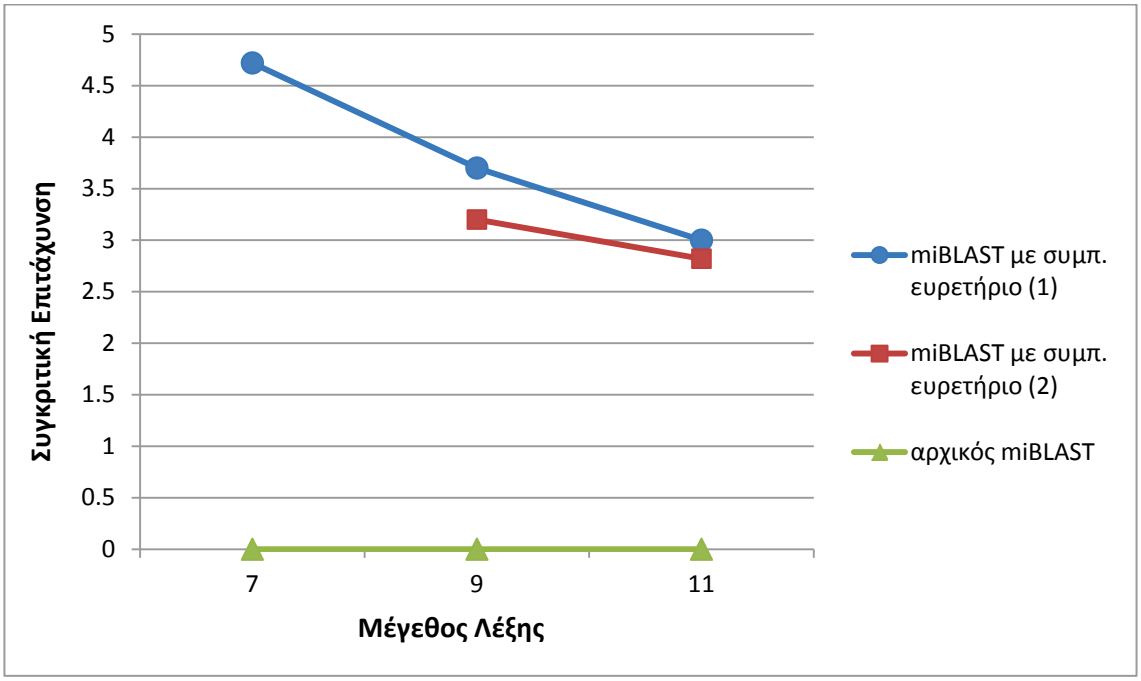
Μέγεθος Λέξης	miBLAST	miBLAST με Συμπιεσμένο Ευρετήριο	
		1η Εκδοχή Delta code + Elias gamma code	**2η Εκδοχή** Delta code + Elias delta code
7	**20481**	**4342**	**-**
9	**21563**	**5848**	**6780**
11	**21560**	**7181**	**7649**

Στον Πίνακα 3.3 καταγράφεται ο χρόνος που απαιτείται για την κατασκευή ενός ευρετηρίου για την βάση UniGene, χρησιμοποιώντας διάφορα μεγέθη λέξης. Βλέπουμε λοιπόν πως σε όλες τις περιπτώσεις, ο χρόνος κατασκευής ενός συμπιεσμένου ευρετηρίου είναι πολύ μικρότερος από αυτόν που χρειάζονταν ο αρχικός miBLAST για την δημιουργία του αντίστοιχου ευρετηρίου**.** Η επιτάχυνση της διαδικασίας αυτής, μπορεί να εξηγηθεί θυμίζοντας ότι ο miBLAST κατά την κατασκευή του ευρετηρίου, κάνει τόσες διαπεράσεις της βάσης όσο το πλήθος των blocks που αποτελούν το ευρετήριο. Έτσι για ένα συμπιεσμένο ευρετήριο, το οποίο χωρίζεται εκ των πραγμάτων σε λιγότερα blocks, απαιτούνται να γίνουν λιγότερες διαπεράσεις της βάσης στην διάρκεια της κατασκευής του. Χαρακτηριστικά, για την δημιουργία ευρετηρίου με μέγεθος λέξης 11, για την ίδια βάση δεδομένων, ο αρχικός miBLAST εκτελούσε 480 διαπεράσεις της βάσης, ενώ ο τροποποιημένος miBLAST που υλοποιήσαμε, εκτελεί μονό 122 (στην εκδοχή με την διαφορική κωδικοποίηση (Delta code) μαζί με Elias delta code).

Επίσης παρατηρούμε πως, όσο πιο μικρό είναι το μέγεθος λέξης για το οποίο κατασκευάζεται το ευρετήριο, τόσο μεγαλύτερο όφελος έχουμε στο χρόνο κατασκευής του ευρετηρίου. Όπως αναφέραμε στην προηγούμενη ενότητα, με μικρό μέγεθος λέξης π.χ. 7, επιτυγχάνουμε μεγαλύτερο ποσοστό συμπίεσης σε σχέση με την περίπτωση συμπίεσης ευρετηρίου για μεγαλύτερο μέγεθος λέξης. Καταλαβαίνουμε λοιπόν πως αυτή η διαφορά στο χρόνο κατασκευής, εμφανίζεται εξαιτίας του διαφορετικού μεγέθους των ευρετηρίων. Επίσης η κωδικοποίηση μικρότερων τιμών διαφοράς, με του κώδικες του Elias, απαιτεί σχετικά λιγότερο

χρόνο. Άρα και αυτός ο παράγοντας συμβάλει στην επιτάχυνση της διαδικασίας όταν έχουμε μικρό μέγεθος λέξης.

Στο Γράφημα 3.2 παρουσιάζεται η συγκριτική επιτάχυνση της διαδικασίας κατασκευής ευρετηρίου, που επιτυγχάνουμε εφαρμόζοντας συμπίεση, σε σχέση με τον χρόνο που απαιτεί ο miBLAST για την ίδια διαδικασία.



Γράφημα 3.2: Συγκριτική επιτάχυνση της κατασκευής ευρετηρίου εφαρμόζοντας συμπίεση

Για μέγεθος λέξης 7, ο τροποποιημένος miBLAST που χρησιμοποιεί τον αλγόριθμο διαφορικής κωδικοποίησης μαζί με τον Elias gamma code (1η εκδοχή), χρειάζεται 4,7 φορές λιγότερο χρόνο για την κατασκευής του ευρετηρίου, από ότι ο αρχικός miBLAST. Για μέγεθος λέξης 9, με την ίδια μέθοδο, επιτυγχάνουμε επιτάχυνση της διαδικασίας κατά 3,7 φορές, ενώ για μέγεθος λέξης 11, η κατασκευή του ευρετηρίου γίνεται 3 φορές πιο γρήγορα σε σύγκριση με τον αρχικό miBLAST. Με την δεύτερη εκδοχή της συμπίεσης (διαφορική κωδικοποίηση μαζί με Elias delta code), το κέρδος σε χρόνο είναι σχετικά μικρότερο. Αυτό συμβαίνει επειδή ο αλγόριθμος κωδικοποίησης ακεραίων Elias delta code έχει μεγαλύτερη πολυπλοκότητα από τον Elias gamma code. Ωστόσο ,όπως είδαμε παραπάνω, με τον Elias delta code, για μέγεθος λέξης 11 το ευρετήριο που παράγεται είναι κατά 300 MB μικρότερο σε σχέση με την πρώτη εκδοχή που χρησιμοποιούμε τον gamma code.

3.3 Χρόνος Φιλτραρίσματος Βάσης

Ο miBLAST για την εκτέλεση ενός ερωτήματος σύγκρισης πάνω σε μία βάση δεδομένων, ακολουθεί τρία βασικά βήματα. Πρώτα κατασκευάζει το ευρετήριο για την βάση, εφόσον αυτό δεν υπάρχει. Έπειτα εκτελεί την λειτουργία φιλτραρίσματος της βάσης και τέλος χρησιμοποιεί το εργαλείο στοίχισης για τον υπολογισμό της ομοιότητας των ακολουθιών.

Σε αυτό το σημείο θα παρουσιάσουμε τα αποτελέσματα των δοκιμών για την μέτρηση του χρόνου φιλτραρίσματος που επιτυγχάνει η δική μας εκδοχή του miBLAST. Θα τα αντιπαραθέσουμε με τους χρόνους του αρχικού miBLAST, έτσι ώστε να μπορούμε να τους συγκρίνουμε.

Όπως αναφέραμε κατά την περιγραφή του miBLAST, λειτουργία φιλτραρίσματος (filtering) ονομάζεται η διαδικασία όπου για κάθε ακολουθία του ερωτήματος, επιλέγονται οι ακολουθίες της βάσης με τις οποίες έχει κάποια κοινή λέξη. Με τις συγκεκριμένες ακολουθίες θα γίνει έπειτα ο υπολογισμός της ομοιότητας για την ακολουθία του ερωτήματος. Το q-gram ευρετήριο χρησιμοποιείται ουσιαστικά για την επιτάχυνση της διαδικασίας φιλτραρίσματος. Καταλαβαίνουμε λοιπόν ότι ο χρόνος που απαιτείται για την αναζήτηση και ανάγνωση του ευρετηρίου παίζει καθοριστικό ρόλο στην απόδοση της λειτουργίας αυτής. Κατά την εκτέλεση της λειτουργίας φιλτραρίσματος σαρώνεται όλο το ευρετήριο, ωστόσο αναζητούνται μόνο οι λέξεις που περιέχονται στο ερώτημα. Έτσι λοιπόν ανακτούνται μόνα τα δεδομένα για τις λέξεις αυτές.

Γνωρίζουμε φυσικά πως με την εφαρμογή της συμπίεσης έχουμε κάποιο επιπλέον κόστος, λόγω της αποκωδικοποίησης των δεδομένων κατά την ανάγνωση του ευρετηρίου. Παρόλα αυτά πρέπει να ελέγξουμε εάν το μειωμένο μέγεθος του ευρετηρίου μπορεί να επιδράσει θετικά, έτσι ώστε να εκτός από τον χρόνο κατασκευής να έχουμε όφελος ακόμα και στον χρόνο ανάγνωσης.

Σαν ερώτημα για τις δοκιμές χρησιμοποιήσαμε 1000 ακολουθίες μήκους 25 νουκλεοτιδίων, τις οποίες πήραμε μέσα από την συλλογή GeneChip Human Genome U133A 2.0 Array της εταιρίας Affymetrix. Οι δοκιμές έγιναν για διάφορα μεγέθη λέξης. Το μέγεθος λέξης που δίνουμε σαν είσοδο στην διαδικασία φιλτραρίσματος, πρέπει να είναι ίσο με το μέγεθος λέξης που θα ορίσουμε έπειτα σαν παράμετρο του BLAST για την εξαγωγή της τελικής στοίχισης. Για τα μεγέθη λέξης 9 και 11 χρησιμοποιήσαμε ευρετήριο αντίστοιχου με μέγεθος λέξης, ενώ για φιλτράρισμα με

παράμετρο 13 και 15 επαναχρησιμοποιήσαμε το ευρετήριο με μέγεθος λέξης 11, εφαρμόζοντας την τεχνική sliding-window του miBLAST.

Στον Πίνακας 3.4 καταγράφεται ο χρόνος που χρειάζεται ο τροποποιημένος miBLAST για την εκτέλεση του φιλτραρίσματος της βάσης UniGene, χρησιμοποιώντας συμπιεσμένο ευρετήριο. Γίνεται επίσης σύγκριση με τον χρόνο που απαιτεί ο αρχικός miBLAST για αυτήν την διαδικασία.

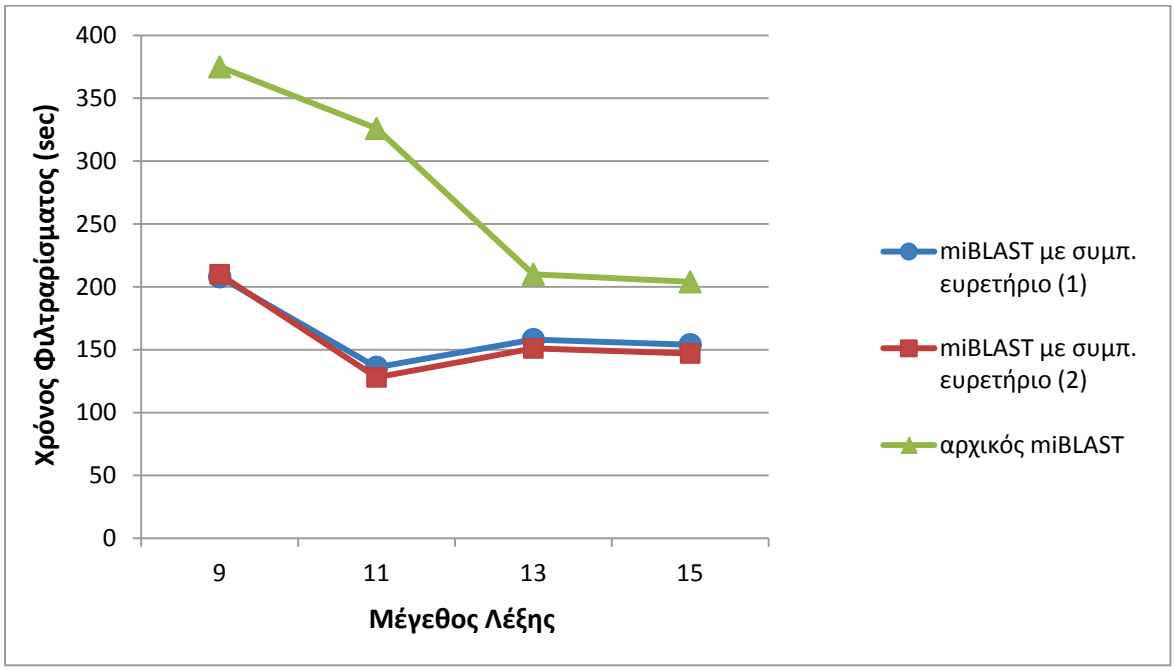
Πίνακας 3.4: Χρόνος φιλτραρίσματος της βάσης για διάφορα μεγέθη λέξης (σε sec)

Μέγεθος Λέξης	miBLAST	miBLAST με Συμπιεσμένο Ευρετήριο	
		1η Εκδοχή Delta code + Elias gamma code	**2η Εκδοχή** Delta code + Elias delta code
9	**375**	**208**	**210**
11	**326**	**136**	**128**
13	**210**	**158**	**151**
15	**204**	**154**	**147**

Μπορούμε να δούμε πως, για όλα τα μεγέθη λέξης, η δική μας μέθοδος επιτυγχάνει εμφανώς καλύτερο χρόνο. Αυτό το γεγονός εξηγείται επειδή το συμπιεσμένο ευρετήριο φορτώνεται πιο γρήγορα στην μνήμη, κατά την ανάγνωση του. Απαιτούνται δηλαδή λιγότερες διαδικασίες E/E (I/O) με τον σκληρό δίσκο. Υπενθυμίζουμε ότι το ευρετήριο φορτώνεται ανά τμήματα (blocks) σταθερού μεγέθους, με την χρήση ενός buffer. Καταλαβαίνουμε λοιπόν ότι επωφελούμαστε το μικρό μέγεθος του ευρετηρίου.

Τα αποτελέσματα αυτά τα αποτυπώσαμε στο Γράφημα 3.3, έτσι ώστε να γίνεται πιο εύκολη η σύγκριση. Όπως φαίνεται, το μεγαλύτερο κέρδος σε χρόνο το έχουμε για τα μεγέθη λέξης 9 και 11. Πιο συγκεκριμένα, για μέγεθος λέξης 11, η μέθοδος μας (και στης δύο εκδοχές της) είναι παραπάνω από δύο φορές πιο γρήγορη από τον αρχικό miBLAST. Για τα μεγέθη λέξης 13 και 15, το κέρδος που επιτυγχάνουμε είναι μικρότερο. Αυτό συμβαίνει επειδή ο περισσότερος χρόνος καταναλώνεται για

τους υπολογισμούς που απαιτούνται για την επαναχρησιμοποίηση του ευρετηρίου με μικρότερο μέγεθος και όχι για την ανάγνωση του ευρετηρίου.



Γράφημα 3.3: Σύγκριση του χρόνου φιλτραρίσματος που επιτυγχάνουμε

Συγκρίνοντας τις δύο εκδοχές συμπίεσης που εφαρμόσαμε, βλέπουμε ότι ο συνδυασμός διαφορικής κωδικοποίησης μαζί με Elias delta code έχει λίγο καλύτερο αποτέλεσμα. Αυτό εξηγείται διότι σε αυτή την περίπτωση το συμπιεσμένο ευρετήριο είναι μικρότερο, άρα φορτώνεται και πιο γρήγορα στην μνήμη. Το γεγονός ότι ο Elias delta code έχει μεγαλύτερη πολυπλοκότητα από τον Elias gamma code, δεν επιβαρύνει σε χρόνο επειδή, όπως αναφέραμε, κατά την αναζήτηση στο ευρετήριο απαιτείται να αποκωδικοποιηθεί μόνο ένα μέρος από τα δεδομένα που περιέχει. Άρα απαιτείται περισσότερος χρόνος για την φόρτωση του ευρετηρίου, σε σχέση τον χρόνο ανάκτησης των δεδομένων που αναζητούνται.

3.4 Συνολικός χρόνος εκτέλεσης ερωτημάτων σύγκρισης

Εδώ θα εξετάσουμε τον συνολικό χρόνο που χρειάζεται η μέθοδος μας για την εκτέλεση ερωτημάτων σύγκρισης πάνω σε μία βάση. Τα αποτελέσματα σε χρόνο που επιτυγχάνει ο miBLAST με συμπιεσμένο ευρετήριο, τον οποίο υλοποιήσαμε, θα τα συγκρίνουμε με τον αρχικό miBLAST, όπως και με τον κλασικό BLAST.

Για τον miBLAST, ο συνολικός χρόνος εκτέλεσης ενός ερωτήματος υπολογίζεται προσθέτοντας τον χρόνο που απαιτείται για την διαδικασία φιλτραρίσματος, με τον χρόνο για την παραγωγή της τελικής στοίχισης. Ο χρόνος κατασκευής του ευρετηρίου δεν προσμετρείται διότι με το ίδιο ευρετήριο μπορούμε να εκτελέσουμε πολλά ερωτήματα. Αφού, όπως διαπιστώσαμε πριν, καταφέρνουμε να μειώσουμε τον χρόνο εκτέλεσης του φιλτραρίσματος, αναμένουμε πως θα έχουμε κέρδος και στον συνολικό χρόνο, μίας και η λειτουργία υπολογισμού της ομοιότητας των ακολουθιών δεν τροποποιήθηκε. Ουσιαστικά η χρήση του ευρετηρίου γίνεται μόνο κατά την διάρκεια του φιλτραρίσματος. Μόλις ολοκληρωθεί αυτή η διαδικασία, τα αποτελέσματα με τις ακολουθίες της βάσης που έχουν «επιτυχία λέξης» για κάθε ακολουθία του ερωτήματος, καταγράφονται σε ένα αρχείο. Αυτό το αρχείο δίνεται σαν είσοδο στο εργαλείο υπολογισμού στοίχισης του BLAST, ο οποίος εξάγει το αποτέλεσμα τις τελικής στοίχισης.

Για τις δοκιμές, όπως και προηγουμένως, χρησιμοποιούμε την βάση δεδομένων NCBI Human UniGene Build #231, έναντι της οποίας εκτελούμε ερωτήματα με ακολουθίες που προέρχονται από την συλλογή GeneChip Human Genome U133A 2.0 Array της Affymetrix. Κάθε ακολουθία του ερωτήματος έχει μήκος 25 νουκλεοτίδια. Έγιναν δοκιμές για διάφορα μεγέθη λέξης ως παράμετρο, καθώς για ερωτήματα διαφορετικού πλήθους ακολουθιών. Για την μέτρηση του χρόνο εκτέλεσης κάθε μεθόδου χρησιμοποιήσαμε την εντολή *time* των Linux. Πριν από την εκτέλεση κάθε ερωτήματος αδειάζουμε τον χώρο της μνήμης όπου προ-αποθηκέυονται δεδομένα από τον δίσκο.

Για να συγκρίνουμε την απόδοση του τροποποιημένου miBLAST που δημιουργήσαμε, εκτελέσαμε τον NCBI BLAST 2.2.25 για τα ίδια ερωτήματα. Η συγκεκριμένη έκδοση του BLAST μπορεί να χειριστεί ερωτήματα με πολλαπλές ακολουθίες, χωρίς περιορισμό στο πλήθος των ακολουθιών. Επίσης τα αποτελέσματα που παρουσιάζει στην έξοδο έχουν την σωστή μορφή, αντίθετα με

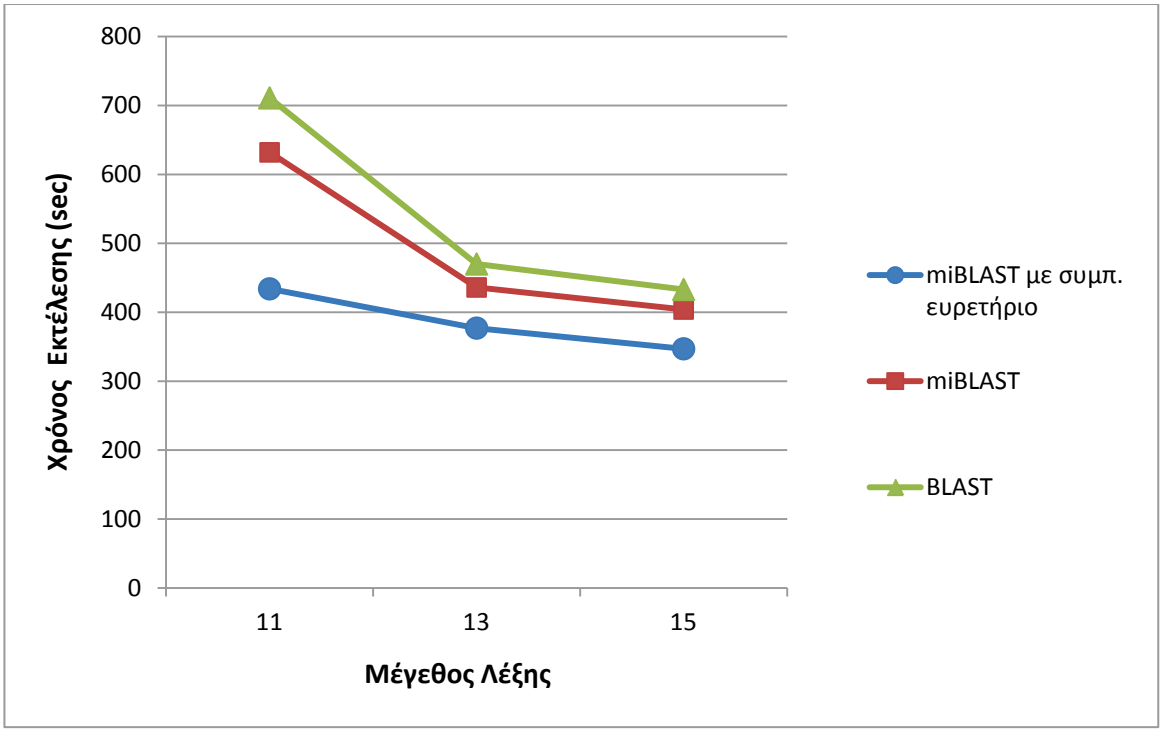
παλαιότερες εκδόσεις. Για αυτό τον λόγο δεν θα εκτελέσουμε κάθε ακολουθία του ερωτήματος ξεχωριστά, αλλά όλες μαζί όπως γίνεται και στον miBLAST.

Πίνακας 3.5: Χρόνος εκτέλεσης ενός ερωτήματος σύγκρισης με 1000 ακολουθίες, για διάφορα μεγέθη λέξης (σε sec)

Μέγεθος Λέξης	miBLAST με Συμπιεσμένο Ευρετήριο	miBLAST	BLAST
11	**434**	**632**	**711**
13	**377**	**436**	**470**
15	**347**	**404**	**433**

Στον Πίνακα 3.5 καταγράφεται ο χρόνος εκτέλεσης ενός ερωτήματος σύγκρισης με 1000 ακολουθίες. Όπως βλέπουμε, ο miBLAST που χρησιμοποιεί συμπιεσμένο ευρετήριο, είναι εμφανώς πιο γρήγορος από τον αρχικό miBLAST και από τον συνηθισμένο BLAST. Η βελτίωση στην απόδοση του miBLAST οφείλεται στο ότι εφαρμόζοντας συμπιεσμένο ευρετήριο, μειώνεται ο χρόνος που απαιτείται για την διαδικασία φιλτραρίσματος, όπως αποδείξαμε προηγουμένως. Επισημαίνουμε ότι σε αυτές τις μετρήσεις καταγράψαμε τον χρόνο εκτέλεσης του miBLAST με συμπιεσμένο ευρετήριο για την εκδοχή όπου τα δεδομένα κωδικοποιούνται με συνδυασμό διαφορικής κωδικοποίησης με Elias delta code.

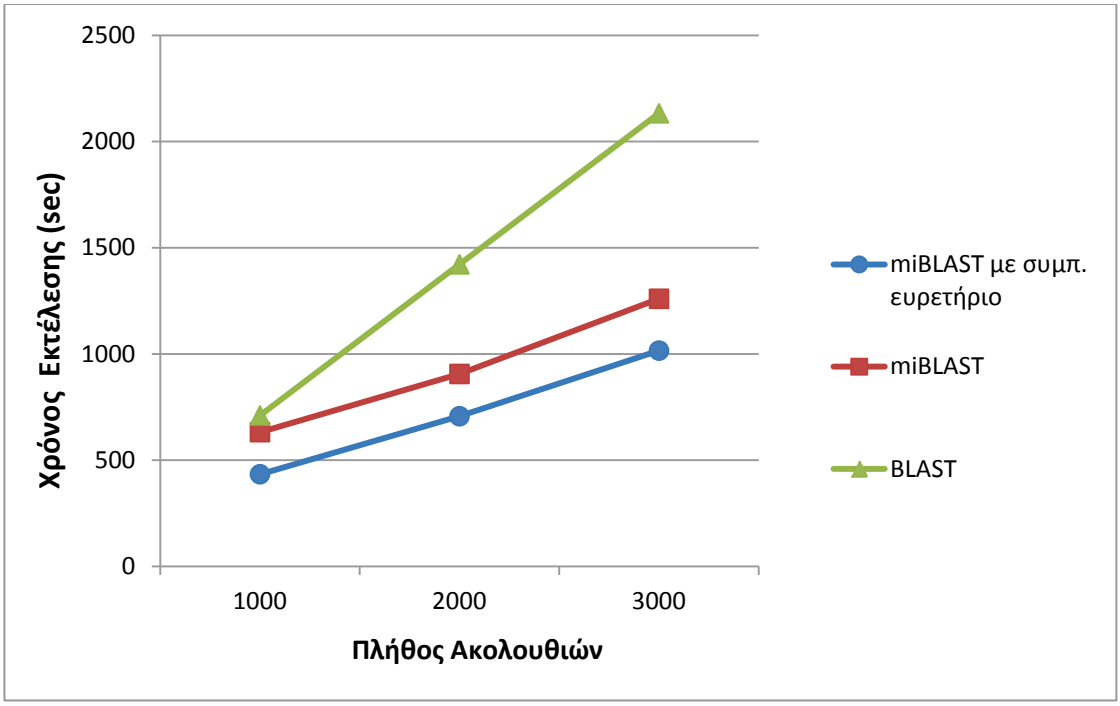
Ξέρουμε επίσης πως σε σχέση με τον BLAST, ο miBLAST είναι πιο γρήγορος στην εκτέλεση ερωτημάτων με πολλαπλές ακολουθίες, λόγω του ότι χρησιμοποιώντας το ευρετήριο βρίσκει πιο γρήγορα τις ταυτίσεις λέξεων. Καταλαβαίνουμε λοιπόν γιατί ο τροποποιημένος miBLAST επιτυγχάνει ακόμα καλύτερα αποτελέσματα συγκρινόμενος με τον BLAST.



Γράφημα 3.4: Σύγκριση του χρόνου εκτέλεσης ενός ερωτήματος σύγκρισης, για διάφορα μεγέθη λέξης

Στον Γράφημα 3.4 αποτυπώνουμε τις τιμές του Πίνακα 3.5, έτσι ώστε να κατανοήσουμε καλύτερα την διαφορά στον χρόνο εκτέλεσης. Μπορούμε να παρατηρήσουμε ότι το μεγαλύτερο κέρδος το έχουμε για μέγεθος λέξης 11, όπως και στην περίπτωση που εξετάζαμε αυτοτελώς τον χρόνο φιλτραρίσματος.

Επίσης εξετάσουμε το πως μεταβάλλεται η απόδοση του miBLAST με το συμπιεσμένο ευρετήριο, όταν χρησιμοποιείται για ερωτήματα με μεγαλύτερο πλήθος ακολουθιών. Γι αυτό λόγο εκτελέσαμε δύο επιπλέον ερωτήματα, με 2000 και 3000 ακολουθίες αντίστοιχα. Για να συγκρίνουμε την απόδοση του, τρέξαμε τα ίδια ερωτήματα στον αρχικό miBLAST και στον κλασσικό BLAST. Παρατηρήσαμε ότι ο BLAST καθυστερεί την εκτέλεση ερωτημάτων που περιέχουν πάνω από 1000 ακολουθίες, γι αυτό τον λόγο του θέσαμε τα παραπάνω ερωτήματα σπαστά ανά 1000 ακολουθίες. Στο Γράφημα 3.5 καταγράφονται οι χρόνοι εκτέλεσης αυτών των ερωτημάτων, για μέγεθος λέξης 11.



Γράφημα 3.5: Σύγκριση του χρόνου εκτέλεσης ερωτημάτων σύγκρισης διαφορετικού πλήθους ακολουθιών

Όπως φαίνεται στον παραπάνω γράφημα, η μέθοδος μας συνεχίζει να είναι γρηγορότερη από τον αρχικό miBLAST και για ερωτήματα με μεγαλύτερο πλήθος ακολουθιών. Όπως αναφέραμε και στην αξιολόγηση του miBLAST, όσο αυξάνεται το πλήθος των ακολουθιών του ερωτήματος, τόσο αυξάνεται η απόδοση του σε σχέση με τον BLAST. Αυτό συμβαίνει διότι μειώνεται το κόστος φιλτραρίσματος ανά ακολουθία. Εμείς τώρα καταφέρνουμε να έχουμε ακόμα πιο βελτιωμένη απόδοση, συγκριτικά με τον BLAST.

Συνοψίζοντας τα αποτελέσματα των πειραμάτων που εκτελέσαμε, μπορούμε να πούμε ότι με την εφαρμογή συμπιεσμένου ευρετηρίου για τον miBLAST καταφέρνουμε να βελτιώσουμε σε μεγάλο βαθμό την απόδοση του. Συγκεκριμένα, επιτυγχάνουμε την μείωση του μεγέθους του ευρετηρίου καθώς και την επιτάχυνση της διαδικασίας κατασκευής του. Επίσης έχουμε κέρδος και στον χρόνο φιλτραρίσματος της βάσης, το οποίο συνεπάγεται την επιτάχυνση συνολικά του χρόνου εκτέλεσης των ερωτημάτων σύγκρισης πάνω στην βάση.

Συμπεράσματα

Ανακεφαλαιώνοντας, αρχικά μελετήσαμε μερικούς αλγόριθμους που είναι σχεδιασμένοι να επεξεργάζονται μεγάλου όγκου βιολογικά δεδομένα. Όπως είδαμε οι περισσότεροι από αυτούς κατασκευάζουν κάποιο ευρετήριο για την βάση δεδομένων, έτσι ώστε να επιταχύνουν την λειτουργία αναζήτησης της.

Χαρακτηριστική περίπτωση είναι ο αλγόριθμος miBLAST, ο οποίος εκτελεί ερωτήματα σύγκρισης έναντι μιας μεγάλης βάσης ακολουθιών, χρησιμοποιώντας ως ερώτημα ένα σύνολο από ακολουθίες. Ο miBLAST κατασκευάζει ένα q-gram ευρετήριο για την βάση δεδομένων, το οποίο αποθηκεύεται στον δίσκο. Για να εντοπίζει τις ομοιότητες μεταξύ των ακολουθιών του ερωτήματος και της βάσης, κάνει αναζήτηση στο ευρετήριο αυτό και εφαρμόζει συνένωση με το ευρετήριο του ερωτήματος. Έτσι, καταφέρνει να εκτελεί την διαδικασία της σύγκρισης πολύ πιο γρήγορα από τον κλασικό BLAST.

Το ευρετήριο της βάσης που χρησιμοποιεί ο miBLAST, συνήθως καταλαμβάνει πολύ χώρο στον δίσκο και όσο μεγαλύτερη είναι η βάση δεδομένων τόσο περισσότερο χώρο απαιτεί. Επίσης λόγω του μεγέθους που έχει, η διαδικασία κατασκευής του αποδεικνύεται χρονοβόρα. Αυτό συμβαίνει διότι κατασκευάζεται ανά τμήματα, μίας και δεν γίνεται να χωρέσει ολόκληρο τα ευρετήριο στην κύρια μνήμη. Για αυτόν τον λόγο εμείς αποφασίσαμε να εφαρμόσουμε συμπίεση στο ευρετήριο που παράγει ο miBLAST. Έτσι χρησιμοποιήσαμε κάποιους αλγόριθμους κωδικοποίησης ακεραίων για να συμπιέσουμε τα δεδομένα που αποθηκεύονται στο ευρετήριο. Συγκεκριμένα εφαρμόσαμε την διαφορική κωδικοποίηση (delta code) σε συνδυασμό με τους κώδικες του Elias. Η διαδικασίας της συμπίεσης και της αποσυμπίεσης ενσωματώθηκε στον υπάρχοντα κώδικα του miBLAST. Για να γίνει αυτό, ήταν απαραίτητο να πραγματοποιήσουμε κάποιες μετατροπές στις συναρτήσεις κατασκευής και ανάγνωσης του ευρετηρίου. Εξ αιτίας της φύσης των κωδικοποιημένων δεδομένων, οι διαδικασίες αυτές απαιτούσαν πιο περίπλοκες τεχνικές χειρισμού του ευρετηρίου.

Από τις δοκιμές που κάναμε, διαπιστώσαμε με την εφαρμογή συμπιεσμένου ευρετηρίου για τον miBLAST, καταφέρνουμε να βελτιώσουμε περεταίρω την απόδοση του. Επιτυγχάνουμε πρωτίστως την σημαντική μείωση του μεγέθους του ευρετηρίου που παράγει, αλλά την επιτάχυνση την διαδικασία κατασκευής του.

Επίσης κάνουμε πιο γρήγορη την ανάγνωση του ευρετηρίου, επωφελούμενοι από το μικρό του μέγεθος. Αυτό συνεπάγεται στην αποδοτικότερη εκτέλεση ερωτημάτων στοίχισης πάνω σε μια μεγάλη βάση δεδομένων.

Σαν συμπέρασμα στο οποίο καταλήγουμε, είναι ότι η εφαρμογή ευρετηρίου σε μεθόδους που επεξεργάζονται μεγάλου όγκου βιολογικά δεδομένα, αποδεικνύεται να είναι πολλά υποσχόμενη. Ενώ στην ανάκτηση δεδομένων από βάσεις που περιέχουν έγγραφα, η χρησιμοποίηση ευρετηρίου είναι ιδιαίτερα διαδεδομένη, αντίθετα στις εφαρμογές βιοπληροφορικής αυτή η μέθοδος δεν έχει υιοθετηθεί σε μεγάλο βαθμό.

Επίσης σημαντικό ρόλο φαίνεται να παίζει και η συμπίεση των ευρετηρίων που κατασκευάζονται, μίας και αυξάνεται συνεχώς ο χώρος που απαιτείται για την αποθήκευση τους. Για αυτόν τον σκοπό μπορούν να εφαρμοστούν αλγόριθμοι κωδικοποίησης που θα προσαρμόζονται στην μορφή των δεδομένων που περιέχονται στο εκάστοτε ευρετήριο. Όπως διαπιστώσαμε, το μειωμένο μέγεθος του ευρετηρίου μπορεί επίσης να έχει όφελος και στον χρόνο αναζήτησης των περιεχομένων του.

Η μέθοδος της συμπίεσης θα μπορούσε λοιπόν να εφαρμοστεί και σε άλλες περιπτώσεις που χρησιμοποιείται ευρετήριο, επιφέροντας θετικά αποτελέσματα.

Βιβλιογραφία

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., 1990,Basic local alignment search tool, *J.Mol. Biol*. 215403–410

[2] Kent, W.J., 2002, BLAT-the BLAST-like alignment tool, *Genome Res.* 12656–664

[3] Zhang, Z., 2000, A greedy algorithm for aligning DNA sequences, *J. Comp. Biol*. 7203–214

[4] Morgulis A, et al., 2008, Database indexing for production megablast searches, *Bioinformatics* 24:1757-1764.

[5] Jung Kim Y, et al. 2009, ProbeMatch: rapid alignment of oligonucleotides to genome allowing both gaps and mismatches, *Bioinformatics* 25:1424-1425.

[6] A. Cox. Eland: Efficient local alignment of nucleotide data (ανέκδοτο)

[7] H. Li. Mapping and assembly with quality. http://maq.sourceforge.net.

[8] Ning, Z., Cox, A.J., Millikin, J.C., 2001, SSAHA: A Fast Search Method for Large DNA Databases, *Genome Res.* 111725–1729

[9] Burkhardt, S., Crauser, et al., 1999, q-gram based database searching using a suffix array QUASAR Proceeding of the third International Conference on Computational Molecular Biology, Lyon, France pp. 77–83

[10] Korf, I. and Gish, W., 2000, MPBLAST: improved BLAST performance with multiplexed queries, *Bioinformatics* 161052–1053

[11] Wang, H., Ooi, B.C., Tan, K.L., Ong, T.H., Zhou, L., 2003, BLAST++: BLASTing queries in batches, *Bioinformatics* 192323–2324

[12] Darling, A.E., Carey, L., Feng, W., 2003, The design, implementation and evaluation of mpiBLAST Proceeding of the Fourth International Conference on Linux Clusters, San Jose, CA

[13] Kim YJ, et al., 2005, miBLAST: scalable evaluation of a batch of nucleotide sequence queries with BLAST, *Nucleic Acids Res* 33.4335-4344.

[14] Navarro, G. and Yates, R.B., 1998, A practical q-gram index for text retrieval allowing errors *CLEI Electronic Journal* 1 pp. 1725–1729

[15] Rouillard, J., Zuker, M., Culari, E., 2003, OligoArray 2.0: design of oligonucleoide probes for DNA microarray using a thermodynamic approach, *Nucleic Acids Res*. 313057–3062

[16] Wright, M.A. and Church, G.M., 2002, An open-source oligomicroarray standard for human and mouse Nat *biotechnol* 201082–1083

[17] Florea, L., Hartzell, G., Zhang, Z., Rubin, G.M., Miller, W., 1998, A computer program for aligning a cDNA sequence with a genomic DNA sequence, *Genome Res*. 8967–974

[18] Schwartz, S., Kent, W., Smit, A., et al., 2003, Human-mouse alignments with BLASTZ *Genome Res*. 1103–107

[19] D.G. Korn, K.P. Vo., 1995, Vdelta: Differencing and Compression, Practical Reusable Unix Software, Editor B. Krishnamurthy, John Wiley & Sons, Inc.

[20] Anh V.N., Moffat A., 2004, Index Compression Using Fixed Binary Codewords, *Fifteenth Australasian Database Conference*

[21] P. Elias., 1975, Universal codeword sets and representations of the integers, *IEEE Transactions on Information Theory*. IT-21(2):194-203

[22] Williams, Zobel, 1999, Compressing Integers for Fast File Access, *The Computer Journal.* 42:193-201