ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη του αλγορίθμου BLAST σε περιβάλλον GPU»

Λάμπρος Κ. Γαλάνης
ΜΠΠΛ 10018

Επιβλέπων: Λέκτορας Α. Πικράκης
Συνεπιβλέπων: Επίκουρος Καθηγητής Μ. Ψαράκης
Συνεπιβλέπων Καθηγητής: Γεώργιος Τσιχριτζής

Πειραιάς, *«Μήνας» «2014»*

Επιτελική Σύνοψη

Η παρούσα εργασία καταπιάνεται με την ανάπτυξη του Αλγορίθμου BLAST μοριακής βιολογίας για την στοίχιση (alignment) μορίων πρωτεϊνών από μια βάση δεδομένων γνωστών πρωτεϊνών μιας άγνωστης ακολουθίας πρωτεΐνης. Επιπρόσθετα, παρουσιάζεται και προτείνεται και η χρησιμοποίηση μιας ή περισσοτέρων (συστοιχίας) καρτών – επεξεργαστών γραφικών για την επίσπευση της διαδικασίας μέσω πολυπαράλληλων υπολογισμών που μπορούν να προσφέρουν οι εν λόγω επεξεργαστές μέσω του framework CUDA

Η εργασία αυτή χωρίζεται στις κάτωθι ενότητες: Προσδιορισμός του προβλήματος στοίχισης πρωτεϊνών, περιγραφή του αλγορίθμου BLAST, περιγραφή στοιχείων της πολυπαράλληλης επεξεργασίας με τη GPU και παρουσίαση του κώδικα για τους ζητούμενους υπολογισμούς. Εν κατακλείδι παρουσιάζονται τα αποτελέσματα και τα εξαγόμενα συμπεράσματα

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Κεφάλαιο 1ο

1 Εισαγωγή

Το παρόν πόνημα αναφέρεται σε ένα μεγάλο πρόβλημα της μοριακής βιολογίας: Την στοίχιση ακολουθιών (sequence alignment) αγνώστων πρωτεϊνών με ακολουθίες γνωστών πρωτεϊνών από μια βάση δεδομένων. Δηλαδή σε ποιες περιοχές της σημειολογίας τους οι ακολουθίες αυτές ταιριάζουν. Για το σκοπό αυτό έχουν αναπτυχθεί διάφοροι αλγόριθμοι βιοπληροφορικής που είναι το σημαντικότερο εργαλείο για την επίλυση του ανωτέρω προβλήματος. Ονομαστικά αναφέρονται οι Smith – Waterman [1], Needleman - Wunsch [2] Sellers [3] οι οποίοι όπως αναφέρθηκε «μετρούν» την ομοιότητα, με βιολογικούς όρους, μεταξύ ακολουθιών ώστε να επισημανθεί η οποιαδήποτε συσχέτιση μεταξύ τους μέσα από καθαρά στοχαστικές διαδικασίες.

Ο Αλγόριθμος BLAST πρόκειται για μια εξέλιξη των παραπάνω αλγορίθμων και εμπίπτει στην κατηγορία προγραμμάτων δυναμικού προγραμματισμού, δηλαδή το πρόβλημα που επιλύει ο αλγόριθμος «σπάει» σε μικρότερα προβλήματα για τα οποία ζητείται η καλύτερη «δυνατή» λύση. Επίσης ο αλγόριθμος εμπίπτει στην κατηγορία των ευρεστικών αλγορίθμων δηλαδή για την επίλυση ενός μέρους του προβλήματος οι λύση(/εις) επιλέγονται από ένα πεπερασμένο σύνολο από τα δεδομένα του προβλήματος τα οποία πρέπει να ικανοποιούν κάποιες απαιτήσεις. Αναπτύχθηκε από τον Stephen S. Altshul [4]. Η μέθοδος αυτή πρόκειται για μια ταχεία προσέγγιση στο πρόβλημα της σύγκρισης ακολουθιών. Το Εργαλείο Εύρεσης Βασικής Τοπικής Στοίχισης (Basic Local Alignment Search Tool) ανιχνεύει άμεσα τις τοπικές στοιχίσεις που έχουν σημασία για την ομοιότητα και δίδουν κάποιο μέτρο στατιστικά και μαθηματικά ωφέλιμο ήτοι το μέγιστο τμήμα ομοιότητας μεταξύ δύο ακολουθιών (Maximal Segment Pair) και το αντίστοιχό τους σκορ.

Ο βασικός αλγόριθμος είναι απλός και σταθερός και μπορεί να εφαρμοστεί με ποικίλους τρόπους ανάλογα με το περιεχόμενο και το ζητούμενο της ανάλυσης: από ανίχνευση ομοιοτήτων μεταξύ ακολουθιών DNA ή και ακολουθιών πρωτεϊνών και ταυτοποίηση γονιδίων / γονιδιωμάτων. Επιπρόσθετα με την ευελιξία και τις δυνατότητες ανίχνευσης μέσω των συναρτήσεων του η μέθοδος BLAST είναι μια τάξη μεγέθους ταχύτερη από τις υπάρχουσες, γεγονός που την καθιστά ιδιαίτερα ελκυστική μέθοδο εντοπισμού ομοιοτήτων, δεδομένου ότι οι διαδικασίες και οι υπάρχοντες τρόποι εντοπισμού είναι ιδιαίτερα μνημοβόρες και απαιτητικές σε υπολογιστική ισχύ και χρόνο λόγω του ότι οι ανιχνεύσεις γίνονται σε βάση δεδομένων που μπορεί να περιέχουν και πολλές εκατομμύρια γνωστές ακολουθίες.

Είναι γνωστό από την αρχιτεκτονική των καρτών γραφικών (GPUs), ότι οι υπολογισμοί για την αναπαράσταση των κινούμενων εν γένει εικόνων στην οθόνη γίνονται παράλληλα για clusters από pixels ή και pixels γενικότερα. Αυτού του είδους αρχιτεκτονική πολυπαράλληλης επεξεργασίας, γνωστή και ως SIMD ή SIMT (Single Instruction Multiple Data / Threads) δημιούργησε κάποια στιγμή λόγω των πλεονεκτημάτων που προσφέρει (την παράλληλη δηλαδή υπολογιστική δυνατότητα πολλών πράξεων κύκλου μηχανής ταυτόχρονα) την απαίτηση για εκμετάλλευση των δυνατοτήτων της κάρτας γραφικών και για υπολογισμούς ή

επεξεργασία δεδομένων. Η απαίτηση αυτή, μέχρι στιγμής κυρίως από ακαδημαϊκή κοινότητα τους ερευνητές και ορισμένες εξειδικευμένες εμπορικές εφαρμογές (κυρίως εφαρμογές επεξεργασίας εικόνας, video και ήχου), οδήγησε στην ανάπτυξη του framework CUDA και του OpenCL. Δηλαδή της προγραμματιστικής διεπαφής μεταξύ των λειτουργιών του επεξεργαστή και μιας γλώσσας προγραμματισμού σχετικά ανώτερου επιπέδου. Το framework CUDA προσφέρει μια ευρεία γκάμα μακροεντολών και συναρτήσεων καθώς και εντολών για τη C/C++ που προσφέρουν την προαναφερθείσα λειτουργικότητα της nvidia κάρτας GPU στο περιβάλλον του εκάστοτε λειτουργικού του προσωπικού υπολογιστή.

Το framework και οι προσφερόμενες βιβλιοθήκες της cuda απευθύνονται σε μικροεπεξεργαστές nvidia καρτών γραφικών και δίνουν την δυνατότητα μέσω συναρτήσεων στην δημιουργία threads και warps (ομάδες παράλληλων threads). Μέσω αυτών κάποιος χρήστης μπορεί να οργανώσει την χρησιμοποίηση παράλληλων υπολογισμών, τη χρήση μακροέντολών και τη χρήση εντολών όπως της γλώσσας προγραμματισμού C/C++ και να γράψει ένα πρόγραμμα για GPU. Επίσης μπορεί να οργανώσει κατάλληλα την αξιοποίηση των πόρων της GPU όπως η μνήμη (με οποιαδήποτε μορφή μπορεί να εκφράζεται σε έναν επεξεργαστή SIMD) και η επεξεργαστική ισχύς. Ουσιαστικά δηλαδή προσφέρεται ένα πακέτο λογισμικού για την εκμετάλλευση και την αξιοποίηση αρχιτεκτονικών πολυπαράλληλης επεξεργασίας μέσω καρτών γραφικών GPU της nvidia και την επίσπευση έτσι υπολογισμών που χρησιμοποιεί κάποιο πρόγραμμα μέσω των καρτών GPU. Έδω θα πρέπει να παρατηρήσουμε ότι τα αποτελέσματα αυτής της αξιοποίησης είναι εντυπωσιακά: μπορεί να γίνει επιτάχυνση των προγραμμάτων έως 2-5x όπως π.χ σε προγράμματα επεξεργασίας video και όπως θα δούμε και παρακάτω και στους αλγορίθμους ενδιαφέροντος αυτού του εκπονήματος.

Έτσι η ανάγκη για ολοένα ταχύτερους υπολογισμούς στο πεδίο της βιοπληροφορικής οδήγησε στη θεώρηση των δυνατοτήτων των καρτών γραφικών. Η έρευνα και οι εφαρμογές πληθαίνουν ολοένα και περισσότερο ενώ μέχρι σήμερα θεωρείται ότι έχει αποκαλυφθεί πάνω από το 60% των γονιδιωμάτων και των πρωτεϊνών που μπορεί να δεσμεύσει ένας οργανισμός (μικροοργανισμός ή πολυκυτταρικός οργανισμός). Αυτό συνεπάγεται υψηλή απαίτηση σε ταχύτητα ανίχνευσης, επεξεργασίας / ταυτοποίησης πρωτεϊνών και γονιδιωμάτων από τα υπάρχοντα και συνεχώς αναπτυσσόμενα συστήματα πληροφορικής. Η προοπτική σχετικά εύκολης παραλληλοποίησης κάποιων από τους υπολογισμούς των αλγορίθμων της βιοπληροφορικής απέκτησε ιδιαίτερο ενδιαφέρον μιας και αυτό θα επιτάχυνε επιπρόσθετα το χρόνο εκτέλεσης των λειτουργιών τμημάτων των αλγορίθμων βιοπληροφορικής. Έτσι από το 2009 και μετά άρχισαν να φαίνονται οι πρώτες προσπάθειες εκμετάλλευσης αυτής της τεχνολογίας [5,6] με μάλλον θετικά αποτελέσματα.

Η μεταπτυχιακή αυτή εργασία καταπιάνεται με την υλοποίηση ενός από των παραπάνω πηγών [5]. Δηλαδή την υλοποίηση του BLAST σε GPU ή συστάδα GPUs. Εκτός των άλλων, δεδομένου ότι ο αλγόριθμος είναι μεγάλος σε έκταση και ακολουθεί αρκετά βήματα, περιγράφονται και υλοποιούνται τα τμήματα αυτά που μπορούν να γίνουν παράλληλα. Η εργασία εκτός της εισαγωγής, χωρίζεται και στα τμήματα: περιγραφή και μελέτη του προβλήματος (problem definition) και περιγραφή του BLAST. Περιγραφή των σταδίων που ακολουθεί το πρόγραμμα και του αλγορίθμου [5] και περιγραφή του παράλληλης υλοποίησης σε GPU. Στο τέλος εξάγονται συμπεράσματα ενώ προτείνεται και επιπλέον δουλειά που μπορεί να γίνει στο πρόγραμμα πολυπαράλληλης ανίχνευσης ομοιοτήτων πρωτεϊνών.

Κεφάλαιο 2ο

2 Περιγραφή του υπό μελέτη προβλήματος

Για την κατανόηση του υπό μελέτη προβλήματος θα προσεγγίσουμε αρχικά το πρόβλημα στοίχισης ακολουθιών DNA ορίζοντας αρχικά την έννοια της ακολουθίας μέσα από ορισμούς και θεωρία της μοριακής βιολογίας.

Το DNA πρόκειται για το γενετικό υλικό που κληρονομείται, και κρατά το αποτύπωμα της ταυτότητας του οργανισμού από μια γενιά στην επόμενη και καλείται δυοξυριβονουκλεϊκό οξύ. Το ολοκληρωμένο μόριο DNA σε οποιοδήποτε οργανισμό, καλείται γονιδίωμα και μερικές φορές και «βιβλίο της ζωής», αφού περιέχει όλες τις πληροφορίες για τα χαρακτηριστικά του οργανισμού. Κάθε φορά που ένα κύτταρο χωρίζεται, το DNA αντιγράφεται με μια διαδικασία που ονομάζεται αντιγραφή γονιδιώματος (DNA Replication). Η κατανόηση όλων των πληροφοριών που περιέχει το DNA και ειδικότερα το ανθρώπινο DNA είναι μια από τις σπουδαιότερες αναζητήσεις του ανθρώπου και της ιστορίας του, ειδικότερα στην εποχή που διανύουμε ιδιαίτερα από πλευράς της γνώσης, τεχνολογίας, της μοριακής βιολογίας και της αντιμετώπισης ανθρώπινων ασθενειών και ασθενειών των ζώων. Η τελευταία λέξη της φαρμακευτικής, η τεχνολογία οργανικών υλικών γενικότερα και η βιομηχανία εξαρτώνται πλέον από την γνώση γονιδιωμάτων και των πληροφοριών που αυτά φέρουν, για την δημιουργία εξατομικευμένων φαρμάκων, κατάλληλων οργανικών υλών για σκοπούς κτηνιατρικής ή γεωπονίας, και φυσικά την γνώση των οργανισμών και των οργανικών υλών και των συσχετισμών που αυτοί έχουν μεταξύ τους.

Λόγω του ότι το μόριο του DNA είναι σχετικά πολύ μεγάλο έχουν γίνει και συνεχίζονται έρευνες, σχετικά με το τι πληροφορία φέρει για περισσότερα από 50 χρόνια, μέχρι και σήμερα. Αν και το «αλφάβητο» που φέρει του δεν είναι και ιδιαίτερα πολύπλοκο λόγω του ότι αποτελείται από επιμέρους μόρια αδενίνης, κυτοσίνης, γουανίνης και θυμίνης, η προσπάθεια «αποκωδικοποίησής» του, εξακολουθεί να είναι ένα μυστήριο... Για λόγους απλότητας αυτά τα συστατικά μόρια καλούνται ως A,C,G,T στην καθομιλουμένη καθώς και στη βιοπληροφορική. Το DNA συνήθως υφίσταται ως (ελικοειδώς) διπλά περιελιγμένο μόριο, αλλά συνήθως ενδιαφερόμαστε για την μια έλικα του μορίου. Ένα παράδειγμα μιας τέτοιας έκφρασης μπορεί να είναι η κάτωθι:

GAATTC

Η παραπάνω έκφραση δηλώνει έναν οργανισμό ο οποίος αποτελείται από 6 νουκλεοτίδια και έχει μήκος 6 νουκλεοτίδια. Η θεωρία της μοριακής βιολογίας δηλώνει ότι το DNA έχει επίσης και πολικότητα η οποία δηλώνεται ως 5’ για την αρχή του DNA και 3’ για το τέλος του DNA. Για παράδειγμα το άκρο 3’ ενός DNA μπορεί να θεωρηθεί το άκρο τέλους του έλικα. Επιπρόσθετα, του αλφαβήτου των τεσσάρων γραμμάτων, μπορεί να θεωρηθεί και το αλφάβητο των 15 λέξεων που περιγράφουν την «ασάφεια» που μπορεί να υπεισέρχεται. Αυτό το αλφάβητο φαίνεται στον παρακάτω πίνακα:

Σύμβολο	Νουκλεοτίδιο/α	Μνημονικός κανόνας
R	**A / G**	**puRine**
Y	**C / T**	**pYrimidine**
W	**A / T**	**Weak hydrogen bonds**
S	**G / C**	**Strong hydrogen bonds**
K	**G / T**	**Keto in major move**
M	**A / C**	**aMino in major move**
B	**C / G / T**	**not A**
D	**A / G / T**	**not C**
H	**A / C / T**	**not G**
V	**A / C / G**	**not T**
N	**A / C / G / T**	**aNy**

Πίνακας 2. Κώδικες ασαφειών

Ο κανόνας ταιριάσματος στην αλυσίδα DNA σύμφωνα με τη θεωρία, είναι ότι το A κάνει δεσμό με το T, το C κάνει δεσμό με το G. Έτσι στο παρακάτω παράδειγμα μπορεί να προσδιοριστεί η διπλή ελικοειδής μορφή του αλφαβήτου:

```
GAATTC
||||||
CTTAAG
```

Αυτό το παράδειγμα μπορεί να γίνει μια καλή περιγραφή της έννοιας *παλίνδρομο* δηλαδή όταν κάποια αλυσίδα DNA μπορεί να γραφεί και να διαβαστεί και ανάποδα από τον άλλο έλικα.

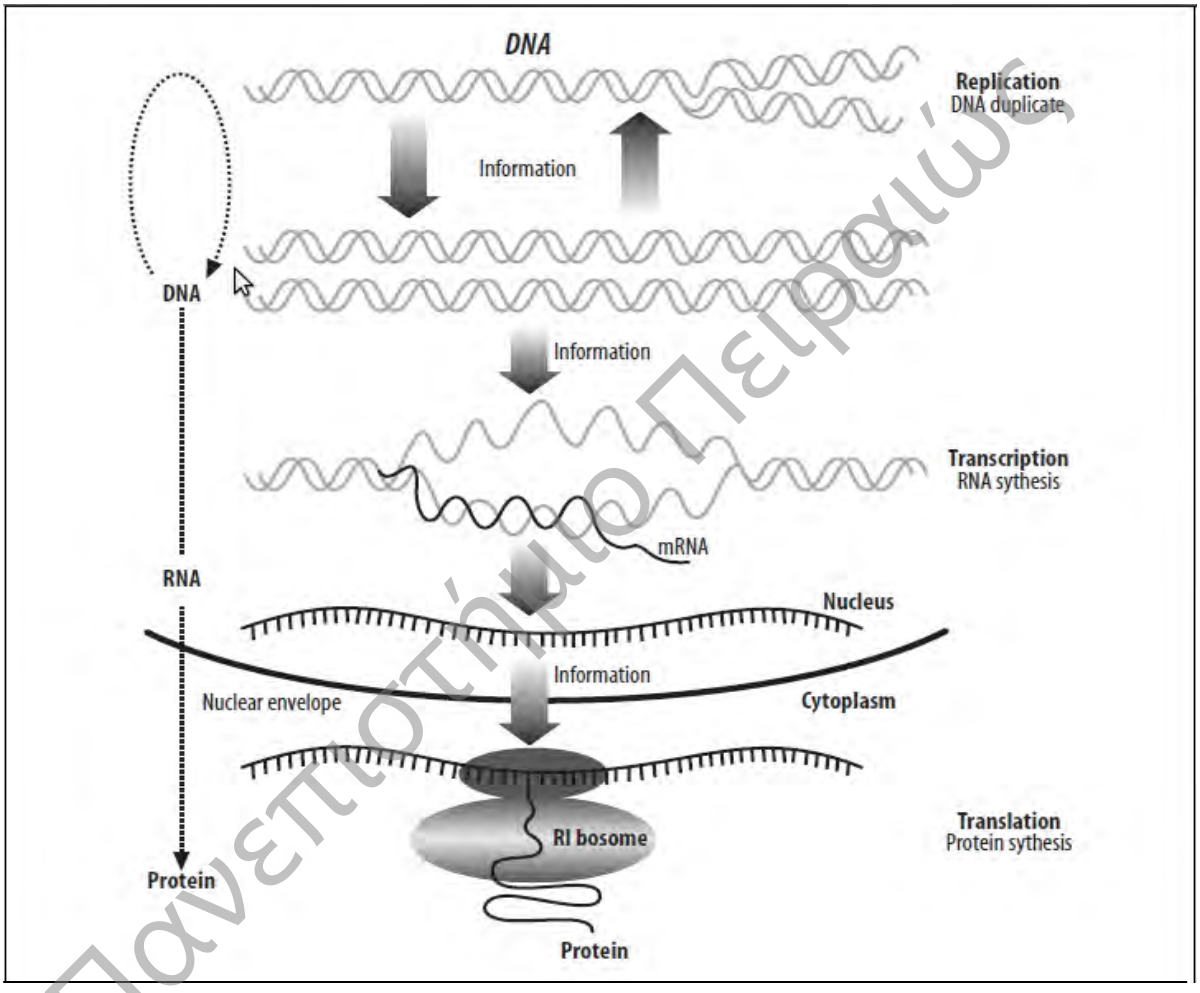
Για την περαιτέρω κατανόηση, όσο αναφορά την έννοια του γονιδίου (gene) της έννοιας δηλαδή ολόκληρου του γενετικού υλικού, η θεωρία μοριακής βιολογίας για το BLAST [1]. αναφέρει ότι ένα γονίδιο είναι μια λειτουργική μονάδα που εξάγεται από τα γονιδιώματα. Το γονιδίωμα ενός οργανισμού είναι καταγεγραμμένο στα χρωμοσώματα που φέρει. Κάποια χρωμοσώματα έχουν πολύ μικρά παράθυρα λειτουργικότητας για καθορισμένο χώρο και χρόνο, ενώ άλλα είναι ασαφή. Κάποια γονίδια παράγουν, μέσω του μηχανισμού αντιγραφής DNA, μόρια RNA τα οποία «δεν μεταφράζονται σε πρωτεΐνες» δηλαδή μη κωδικοποιητέα RNAs.

2.1 Κεντρικό δόγμα του DNA

Εδώ σε αυτή τη μελέτη θα χρησιμοποιήσουμε το Κεντρικό Δόγμα του DNA (central dogma) που πολύ συνοπτικά αναφέρει ότι η πληροφορία που φέρουν τα γονίδια και που τελικά

χρησιμοποιείται από τον οργανισμό συμβαίνει με τρείς τρόπους, γνωστούς και ως γενικές μεταβιβάσεις (general transfers) και που είναι οι κάτωθι:

- Το DNA μπορεί να αντιγραφεί σε όμοιο DNA (DNA replication)
- Κομμάτι του DNA μπορεί να αντιγραφεί σε αγγελιοφόρο RNA (transcription)
- Οι πρωτεΐνες που "επεξεργάζεται" το κύτταρο μπορούν να δημιουργηθούν χρησιμοποιώντας την πληροφορία που φέρει το αγγελιοφόρο RNA-mRNA (translation)



Εικ. 1 Αναπαράσταση του κεντρικού δόγματος

Οι υπόλοιποι συνδυασμοί αντιγραφής του DNA, μεταβίβασης πληροφορίας σε RNA και μετάφρασης σε πρωτεΐνη ονομαζόμενες και ως ειδικές μεταβιβάσεις δεν λαμβάνονται υπ' όψη για την ανίχνευση ακολουθιών πληροφορίας. Έτσι προκύπτουν οι εξής ορισμοί για το RNA και τις πρωτεΐνες:

2.1.1 RNA

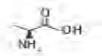
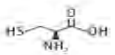
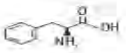
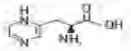
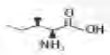
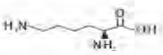
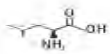
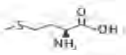
Εδώ ενδιαφέρει κυρίως το φαινόμενο της μεταβίβασης τμήματος του DNA από την αντιγραφή τμήματος του DNA στο RNA μέσω μιας πρωτεΐνης καλούμενη ως RNA πολυμεράση (RNA polymerase). Από χημικής απόψεως το RNA μοιάζει με το DNA εκτός του ότι χρησιμοποιείται η

Uracil από τη Thymine και είναι μονά περιελιγμένη σε αντίθεση από το DNA το οποίο είναι διπλά περιελιγμένο όπως αναφέρθηκε και παραπάνω. Υπάρχουν διάφορα είδη μορίων RNA όπως το transfer RNA, το ribosomal RNA και ncRNA τα οποία υπόκεινται σε χημικές αλλαγές αλλά φέρουν την πληροφορία σαν μόρια RNA. Τα μόρια RNA τα οποία ενδιαφέρουν στο παρόν πόνημα είναι τα μόρια RNA που σχετίζονται με τις πρωτεΐνες και καλούνται αγγελιοφόρα RNA μόρια – mRNA.

2.1.2 Πρωτεΐνες

Οι πρωτεΐνες συνιστούν τις «δομές» και τις «μηχανές» σε ένα κύτταρο. Χημικά πρόκειται για μόρια με εντελώς διαφορετική χημική δομή από ότι τι DNA και το RNA λόγω του ότι αποτελούνται από αμινοξέα (συχνά αποκαλούνται και aa (amino acids)) παρά από νουκλεϊκά οξέα. Οι πρωτεΐνες έχουν μια χρήσιμη ιδιότητα, μπορούν να δημιουργήσουν πολύ συγκεκριμένες 3-Δ γεωμετρίες που εξαρτώνται από ακολουθίες αμινοξέων από τις οποίες μεταφράστηκαν. Συνοπτικά, οι ακολουθίες αμινοξέων καθορίζουν την γεωμετρία των πρωτεϊνών και το σχήμα τους την λειτουργία που εκτελούν. Έτσι μια πρωτεΐνη σχήματος δύσκαμπτης ράβδου μπορεί να χρησιμοποιηθεί σαν δομικό στοιχείο που δίδει σταθερότητα σε ένα κύτταρο. Έτσι για παράδειγμα το Κολλαγόνο και η Κερατίνη δίδουν μεγαλύτερη αντοχή και διατηρησιμότητα στις τρίχες και το δέρμα ενός οργανισμού που τις χρησιμοποιεί. Μια πρωτεΐνη που φέρει ένα σχήμα άγκιστρου μπορεί να χρησιμοποιηθεί ως τμήμα ενός κυττάρου με μηχανικές ιδιότητες. Για παράδειγμα η μυοσίνη που χρησιμοποιείται στους μυς ενός οργανισμού. Τελικά ενώ το DNA και το RNA χρησιμοποιούνται και φέρουν πληροφορίες, οι πρωτεΐνες κάνουν πράγματα να συμβαίνουν.

Το αλφάβητο των πρωτεϊνών αποτελείται από 20 στοιχεία που αναπαριστώνται από σύμβολα του Αγγλικού αλφαβήτου ήτοι A, C, D, E, F, G, H, I, K, L, M,N, P, Q, R, S, T, V, W και Y

Alanine	Ala	A	Hydrophobic	
Cysteine	Cys	C	Neutral; forms disulfide bridges	
Aspartate	Asp	D	Negatively charged	
Glutamate	Glu	E	Negatively charged	
Phenylalanine	Phe	F	Hydrophobic; aromatic	
Glycine	Gly	G	Neutral; smallest amino acid	
Histidine	His	H	Positively charged; aromatic	
Isoleucine	Ile	I	Hydrophobic	
Lysine	Lys	K	Positively charged	
Leucine	Leu	L	Hydrophobic	
Methionine	Met	M	Hydrophobic; start amino acid	

Asparagine	Asn	N	Neutral ; hydrophilic
Proline	Pro	P	Hydrophobic
Glutamine	Gln	Q	Neutral ; hydrophilic
Arginine	Arg	R	Positively charged
Serine	Ser	S	Neutral; hydrophilic
Threonine	Thr	T	Neutral ; hydrophilic
Valine	Val	V	Hydrophobic
Tryptophan	Trp	W	Hydrophobic; aromatic
Tyrosine	Tyr	Y	Hydrophobic; aromatic

πιν. 2 Πίνακας αμινοξέων

Χρησιμοποιώντας τους παραπάνω συμβολισμούς μια πρωτεΐνη μπορεί να γραφεί όπως παρακάτω:

MLVGSRA

Όπως το DNA και το RNA, οι πρωτεΐνες φέρουν επίσης πολικότητα και η ονοματολογία τους προέρχεται από τη χημική τους δομή. Επίσης η σύμβαση είναι να αναφέρεται η πρωτεΐνη από αριστερά προς τα δεξιά. Το αριστερό άκρο αναφέρεται ως N-terminus και το δεξί άκρο ως C-terminus. Έτσι, όταν αναφέρεται ότι το N-terminus (αριστερό άκρο) αφαιρείται κατά τη διαδικασία της μετάφρασης, εννοείται η αρχή της πρωτεΐνης η οποία μάλιστα είθισται να είναι (με κάποιες εξαιρέσεις) η Μεθιονίνη (M).

Αν και η σημειολογία των πρωτεϊνών αναφέρεται σε ακολουθία συμβόλων μιας διάστασης, στην πραγματικότητα οι πρωτεΐνες λαμβάνουν τρεις διαστάσεις στο χώρο είτε και τέσσερεις αν ληφθεί υπ' όψιν ότι οι πρωτεΐνες μπορούν να αλλάξουν γεωμετρία ανάλογα το περιβάλλον που βρίσκονται.

2.2 Γενετικός κώδικας

Περιγραφικά η μετάφραση του DNA και RNA σε μια ακολουθία πρωτεϊνών είναι η ακόλουθη: Στο συστατικό μέρος ενός κυττάρου καλούμενο και ως ριβοσώμα (ribosome), το οποίο είναι ουσιαστικά μια «μηχανή» αποτελούμενη από πρωτεΐνες και ncRNAs, διαβάζεται η ακολουθία

mRNA και εγγράφεται η ακολουθία της πρωτεΐνης. Η ανάγνωση του mRNA γίνεται διαδοχικά ανά τρία νουκλεοτίδια, τα οποία καλούνται και ως κωδικόνια (codons). Κάθε κωδικόνιο αντιστοιχεί σε ένα και μοναδικό αμινοξύ. Η αντιστοίχηση των κωδικονίων σε αμινοξέα ονομάζεται γενετικός κώδικας. Επειδή τα κωδικόνια έχουν μήκος τριών νουκλεοτιδίων και επειδή υπάρχουν τεσσερα διαφορετικά νουκλεοτίδια, έπεται ότι θα υπάρχουν 64 κωδικόνια (4^3). Παρόλα αυτά, υπάρχουν 20 αμινοξέα. Οπότε υπάρχει πλεονασμός στην πληροφορία του γενετικού κώδικα με κάποιους από τους συνδυασμούς των νουκλεοτιδίων να μεταφράζονται σε ίδια αμινοξέα. Ο πλεονασμός αυτός μπορεί να αναλυθεί σε περαιτέρω κανόνες και πρότυπα (patterns) που ακολουθούνται από τη φύση κατά τη μετάφραση. Έτσι για παράδειγμα συνήθως σε ένα κωδικόνιο δεν παίζει ρόλο η κατάληξή του, δηλαδή το τελευταίο νουκλεοτίδιο της τριπλέτας. Επίσης ένα κώδικόνιο με Τ στη μεσαία θέση της τριπλέτας θα μεταφραστεί σε υδροφοβικό αμινοξύ, κλπ. Ένας άλλος κανόνας είναι ότι υπάρχουν ειδικοί συνδυασμοί που ονομάζονται κωδικόνια τέλους. Όταν το ριβοσώμα αναγνωρίσει μια τέτοια τριπλέτα σταματά η μετάφραση και το μόριο της πρωτεΐνης αφήνεται να πραγματοποιήσει τη λειτουργία του.

Αυτό που έχει σημασία σ' αυτή την εργασία είναι απλά η μετάφραση και όχι οι κανόνες που υπεισέρχονται σ' αυτή. Έτσι συνοπτικά η μετάφραση των κωδικονίων φαίνεται στο παρακάτω πίνακα

	T	C	A	G
T	TTT Phe (F) TTC " TTA Leu (L) TTG "	TCT Ser (S) TCC " TCA " TCG "	TAT Tyr (Y) TAC TAA Ter TAG Ter	TGT Cys (C) TGC TGA Ter TGG Trp (W)
C	CTT Leu (L) CTC " CTA CTG "	CCT Pro (P) CCC " CCA " CCG "	CAT His (H) CAC " CAA Gin (Q) CAG "	CGT Arg (R) CGC " CGA " CGG "
A	ATT Ile (I) ATC " ATA " ATG Met (M)	ACT Thr (T) ACC " ACA " ACG "	AAT Asn (N) AAC " AAA Lys (K) AAG "	AGT Ser (S) AGC " AGA Arg (R) AGG "
G	GTT Val (V) GTC " GTA " GTG "	GCT Ala (A) GCC " GCA " GCG "	GAT Asp (D) GAC " GAA Glu (E) GAG "	GGT Gly (G) GGC " GGA " GGG "

Πιν. 3 Η συνήθης μετάφραση των κωδονίων

Για παράδειγμα έστω η ακολουθία νουκλεοτιδίων:

TTTATATCACAC

Αν μεταφραστεί από το πρώτο γράμμα, λαμβάνεται η παρακάτω ακολουθία αμινοξέων

FISH

Αν όμως μεταφραστεί με αρχή το δεύτερο νουκλεοτίδιο λαμβάνεται μια άλλη ακολουθία:

LYHT

Δεδομένου ότι το AC θα μεταφραστεί σε T ανεξάρτητα από το υποτιθέμενο επόμενο νουκλεοτίδιο. Γι αυτό το λόγο ορίζονται «πλαίσια μετάφρασης» (reading frames) σαν κυλιόμενα παράθυρα για κάθε ακολουθία του DNA που θέλουμε να μεταφραστεί.

2.2.1 Βιολογικές ακολουθίες και ομοιότητα

Είδαμε ότι οι βιολογικές ακολουθίες όπως οι πρωτεΐνες και το DNA μπορούν να ορίσουν βασικές λειτουργίες σε ένα οργανισμό. Επίσης είναι γνωστό ότι σύμφωνα με τη θεωρία της εξέλιξης αυτές οι ακολουθίες μπορούν να τροποποιηθούν στο χρόνο μέσω των μηχανισμών της τυχαίας μετάλλαξης, της φυσικής επιλογής (natural selection) και της γενετικής παρέκκλισης (genetic drift). Έτσι οι βιολογικές ακολουθίες παρουσιάζουν ομοιότητες και διαφορές επειδή ορίζονται αρχικά από ένα όμοιο οργανισμό και ακολουθούν στο χρόνο διαφορετικά μονοπάτια και παρεκκλίσεις. Πέρα από το γεγονός αυτό και κατά συνέπεια με τα προηγούμενα οι ομοιότητες και η στοίχιση «αγνώστων» ακολουθιών με γνωστές, προσδίδει χαρακτηριστικά και ιδιότητες στην «άγνωστη» ακολουθία όπως αυτές αποκτήθηκαν κατά την εξέλιξη του οργανισμού.

2.3 Αλγόριθμοι εύρεσης ομοιοτήτων μεταξύ ακολουθιών

Μπορούν να θεωρηθούν δύο τύποι για την εύρεση ομοιοτήτων και την στοίχιση ακολουθιών: Η τοπική στοίχιση και η ολική στοίχιση. Στην ολική στοίχιση, οι δύο ακολουθίες στοιχίζονται καθ' όλο το μήκος τους και βρίσκεται η βέλτιστη στοίχιση. Στην τοπική στοίχιση βρίσκεται η καλύτερη στοίχιση της υποακολουθίας – μικρότερης ακολουθίας.

2.3.1 Αλγόριθμος Needleman-Wunsch

Ένας ιστορικά σημαντικός αλγόριθμος ολικής στοίχισης είναι ο [2]. Συνοπτικά ο αλγόριθμος χρησιμοποιεί ένα πίνακα ομοιοτήτων (similarity matrix) που ορίζει τα scores της στοίχισης δύο νουκλεοτιδίων και ορίζει μια τιμή κύρωσης (penalty) για κάθε φορά που υπεισέρχεται κενό μεταξύ της στοίχισης δύο ακολουθιών.

Για παράδειγμα, αν ο πίνακας ομοιοτήτων ήταν ο παρακάτω:

	A	G	C	T
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Τότε το σκορ της στοίχισης

AGACTAGTTAC

CGA - - -GACGT

θα ήταν με κύρωση της παύλας -5 :

S(A,C) + S(G,G) + S(A,A) + (3xd) + S(G,G) + S(T,A) + S(T,C) + S(A,G) + S(C,T)

= -3 + 7 + 10 - (3x5) + 7 + -4 + 0 + -1 + 0 = 1

Το ζητούμενο σ' αυτόν τον αλγόριθμο είναι η εύρεση της στοίχισης με το μεγαλύτερο σκορ. Για το σκοπό αυτό ορίζεται ένας δισδιάστατος πίνακας F που σχηματίζεται ως εξής:

- Υπάρχει μια στήλη στα αριστερά του πίνακα για την ακολουθία A και μια γραμμή πάνω από τον πίνακα για την ακολουθία B
- Καθώς γίνεται επεξεργασία του πίνακα τα στοιχεία F_{ij} πρώτα ανατίθενται στα πεδία $F_{0,j}$ και $F_{i,0}$ οι τιμές F{0j} = d*j και F{i0} = d*i
- Τα υπόλοιπα στοιχεία ορίζονται αναδρομικά βάση της εξής σχέσης:

  F{ij} = max{F{i-1,j-1} + S(A{i}, B{j}), F{i,j-1} + d, F{i-1,j} + d}

Ο ψευδοκώδικας για αυτές τις λειτουργίες φαίνεται παρακάτω:

```
for i=0 to length(A)
F(i,0) ← d*i
for j=0 to length(B)
  F(0,j) ← d*j
for i=1 to length(A)
  for j=1 to length(B)
  {
    Match ← F(i-1,j-1) + S(Ai, Bj)
    Delete ← F(i-1, j) + d
    Insert ← F(i, j-1) + d
    F(i,j) ← max(Match, Insert, Delete)
 }
```

Άπαξ και υπολογιστεί ο πίνακας F , η τιμή F(n , m) δίδει το μέγιστο σκορ από όλες τις πιθανές στοιχίσεις. Για να υπολογιστεί η στοίχιση που δίδει αυτό το σκορ ο αλγόριθμος ξεκινά από το δεξί κάτω στοιχείο του πίνακα και υπολογίζει την τιμή λαμβάνοντας υπόψη την τιμή του γειτονικού στοιχείου και το πώς αυτό προήλθε. Αν υπάρχει match τότε τα A(i) και B(i) είναι στοιχισμένα, αν υπάρχει delete τότε υπάρχει στοίχιση με τη παύλα ή το χαρακτήρα κενού στο στοιχείο A(i) αν υφίσταται το insert τότε υπάρχει στοίχιση με την παύλα στο στοιχείο B(j). Γενικότερα μπορεί να υπάρξουν πολλές στοιχίσεις που δίνουν το ίδιο maximum score.

2.3.2 Τοπική στοίχιση – Αλγόριθμος Smith – Waterman

Ο αλγόριθμος τοπικής στοίχισης που περιγράφεται εδώ [1]είναι μια πολύ απλή παραλλαγή του Needleman Wunsch. Οι αλλαγές περιγράφονται συνοπτικά παρακάτω:

- Οι ακριανές τιμές του πίνακα F(i,0) και F(0,j) είναι μηδενικές αντί για κλιμακούμενα μειωμένες τιμές
- Ένα maximum score δεν μπορεί να έχει αρνητική τιμή και κάθε στοιχείο του πίνακα είναι μη αρνητικός αριθμός
- Η ανίχνευση του μονοπατιού της λύσης δεν ξεκινά από το δεξί κάτω άκρο του πίνακα αλλά από την υψηλότερη τιμή που φέρει ο πίνακας και σταματά όταν βρεθεί αρνητική τιμή

Ο πίνακας που δημιουργείται φέρει στοιχεία που υπολογίζονται ως εξής

$$H(i,0) = 0, 0 \le i \le m$$

$$H(0, j) = 0, 0 \le j \le n$$

$$H(i, j) = \max\begin{Bmatrix} 0 & \\ H(i-1, j-1) + s(a_i, b_j) & match \\ \max_{k \ge 1}\{H(i-k, j) + W_k\} & insertion \\ \max_{l \ge 1}\{H(i, j-l) + W_l\} & deletion \end{Bmatrix}, 1 \le i \le m, 1 \le j \le n$$

όπου:

a, b στοιχεία του αλφάβητου

m το μήκος της ακολουθίας a

n το μήκος της ακολουθίας b

s(a,b) η συνάρτηση ομοιότητας ή αλλιώς το σκορ του πίνακα ομοιότητας

W_i η συνάρτηση ή η τιμή του κόστους εισαγωγής κενού συμβόλου – παύλας

Όπως αναφέρθηκε η ανίχνευση της βέλτιστης τοπικής στοίχισης ξεκινά από τη μεγαλύτερη τιμή του πίνακα H και πηγαίνει προς τα πίσω σε ένα από τα στοιχεία (i-1,j), (i,j-1) και (i-1,j-1) ανάλογα με τη διεύθυνση κίνησης (match, insertion, deletion) που δημιουργήθηκε ο πίνακας.

2.3.3 Σύγκριση μεταξύ τοπικής και ολικής στοίχισης

Ένα κίνητρο για την τοπική στοίχιση είναι η δυσκολία να βρεθούν σωστές στοιχίσεις σε περιοχές με μικρή ομοιότητα σε «μακρινά» συσχετισμένες ακολουθίες, λόγω του γεγονότος ότι οι μεταλλάξεις σε εξελικτικό χρόνο έχουν προσθέσει πολύ «θόρυβο» ώστε να υπάρξει μια λογική σύγκριση μεταξύ ακολουθιών που υπεισέρχονται στη σύγκριση. Εδώ η τοπική στοίχιση παραμελεί τέτοιες περιοχές και «δίνει σημασία» μόνο σε περιοχές που δίνουν θετικό σκορ δηλαδή σε στοιχίσεις που έχουν διατηρήσει κάποια ομοιότητα.

Ένα άλλο πλεονέκτημα της χρήσης τοπικών στοιχίσεων είναι ότι υπάρχει ένα αξιόπιστο στατιστικό μοντέλο (ανεπτυγμένο από Karlin, Altschul [8]) για βέλτιστες τοπικές στοιχίσεις. Αυτό σημαίνει ότι μπορεί να ανιχνευθούν στοιχίσεις οι οποίες να έχουν σκορ κοντά στο αναμενόμενο επιτρέποντας έτσι να ανιχνευθούν και αρκετά ανόμοιες αλλά στοιχίσεις ομόλογων ακολουθιών (δηλαδή ακολουθίες που έχουν κάποιο κοινό πρόγονο).

Το γεγονός ότι υπάρχει αυτό το στατιστικό μοντέλο δίνει την δυνατότητα να περιοριστεί ο χώρος ανίχνευσης στον πίνακα στοίχισης Η. Με αυτή την προσέγγιση σχηματίζεται μια διαγώνιος στον πίνακα ανίχνευσης Η μεταξύ των αναμενόμενων τιμών και η ανίχνευση περιορίζεται μεταξύ ενός μικρότερου αριθμού στοιχείων του πίνακα. Το εύρος του πίνακα καλείται και εύρος ζώνης (bandwidth) και η αναζήτηση των λύσεων γίνεται εντός αυτού του εύρους. Οι μηχανισμοί μείωσης της απαιτούμενης μνήμης και υπολογιστικής ισχύος περιγράφονται παρακάτω.

2.4 Ομοιότητα ακολουθιών

Σ' αυτή την ενότητα θα περιγραφούν με περισσότερη ακρίβεια οι έννοιες σκορ και οι στατιστικοί υπολογισμοί που χρησιμοποιούνται στον αλγόριθμο BLAST και γενικότερα στους αλγορίθμους εύρεσης ομοιοτήτων μεταξύ ακολουθιών.

2.4.1 Εισαγωγή στη θεωρία των πληροφοριών

Στη θεωρία πληροφοριών η έννοια της πληροφορίας ορίζεται σαν την ποσότητα άρσης της αβεβαιότητας για ένα συμβάν ή αλλιώς το μέτρο της «έκπληξης» που μπορεί να προκληθεί από ένα συμβάν. Η πληροφορία από ένα συμβάν μπορεί να μετρηθεί και το μέτρο της λαμβάνει την ακόλουθη τιμή:

$$H(p) = \log_2 \frac{1}{p} = -\log_2 p \quad (bits)$$

Για παράδειγμα το γεγονός ότι η πιθανότητα ένα παιδί να μην του αρέσει ένα παγωτό είναι 0.25 τότε η πληροφορία που λαμβάνεται γι αυτό το γεγονός είναι 2 bits και το να του αρέσει 0,41 bits

Για ένα αλφάβητο με περιορισμένο αριθμό εμφανίσεως των συμβόλων του ορίζεται η έννοια της εντροπίας σαν το μέτρο της μέσης πληροφορίας για την εμφάνιση κάποιου συμβόλου:

$$H = -\sum_{i}^{n} p_i \log_2 p_i \quad (bits)$$

Για παράδειγμα το μέτρο της εντροπίας για την ρίψη ενός νομίσματος αν θεωρηθούν τα ενδεχόμενα να έρθει κορώνα ή γράμματα ισοπίθανα είναι:

-((0.5)(-1) + (0.5)(-1)) = 1 bit

Μια τυχαία ακολουθία DNA που φέρει ισοπίθανα τα σύμβολα A, T, G, C, θα έχει εντροπία:

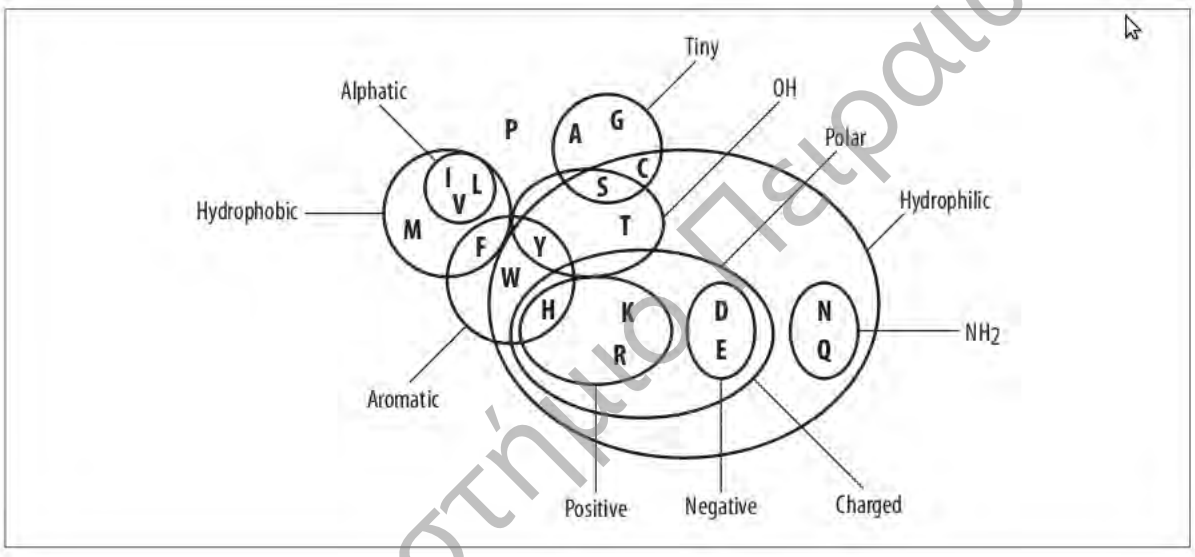
- ((0.25)(-2) + (0.25)(-2) + (0.25)(-2) + (0.25)(-2)) = 2 bits

Ενώ μια ακολουθία που φέρει κατά 90% A ή T και κατά 10% G, C θα έχει εντροπία:

- (2(0.45)(-1.15) + 2(0.05)(-4.32)) = 1.47 bits

2.4.2 Ομοιότητα αμινοξέων

Μοριακοί βιολόγοι συνήθως σκέφτονται την ομοιότητα των αμινοξέων όσον αφορά τη χημική ομοιότητα (βλ. Πίνακα 2). Η Εικ. 2 απεικονίζει μια πρόχειρη ποιοτική κατηγοριοποίηση. Από μια εξελικτική σκοπιά, είναι αναμενόμενο οι μεταλλάξεις που αλλάζουν ριζικά τις χημικές ιδιότητες να είναι σπάνιο φαινόμενο, επειδή μπορεί να καταλήξει να καταστρέψει την τρισδιάστατη δομή της πρωτεΐνης Αντίθετα, οι αλλαγές μεταξύ παρόμοιων αμινοξέων πρέπει να συμβαίνει σχετικά συχνά.



Εικ 2 Οι σχέσεις μεταξύ πρωτεϊνών

Στα τέλη της δεκαετίας του '60 και στις αρχές της δεκαετίας του '70, η Margaret Dayhoff εφεύρε ποσοτικές τεχνικές για τη μέτρηση της ομοιότητας αμινοξέων. Χρησιμοποιώντας ακολουθίες που ήταν διαθέσιμες εκείνη την εποχή, που κατασκευάστηκε πολλαπλές στοιχίσεις των σχετικών πρωτεϊνών και τις σύγκρινε με τις συχνότητες υποκατάστασης αμινοξέων. Όπως ήταν αναμενόμενο, υπάρχει αρκετά μεγάλο κομμάτι διακύμανσης στη συχνότητα αντικατάστασης αμινοξέων, και των patterns των χημικών ιδιοτήτων που αναμένονταν. Για παράδειγμα, φαινυλαλανίνη (F) είναι πιο συχνά συνδεδεμένη με την ίδια. Επίσης εμφανίζεται σχετικά συχνά με την τυροσίνη (Y) και την τρυπτοφάνη (W), τα οποία μοιράζονται παρόμοιες δομές αρωματικού δακτυλίου (βλ. Πίνακα 2-1), και σε μικρότερο βαθμό με τα άλλα υδρόφοβα αμινοξέα (M, V, I και L). Η φαινυλαλανίνη συνδέεται συχνα σε συνδυασμό με υδρόφιλα αμινοξέα (R, K, D, E, και άλλοι). Παρακάτω φαίνονται μερικά από αυτά τα μοτίβα σε πολλαπλή ευθυγράμμιση ακολουθιών τα οποία αντιστοιχούν σε ένα τμήμα του κυτοχρώματος b πρωτεΐνης από διάφορους οργανισμούς:

```
PGNPFATPLEILPEWYLYPVFQILRVLPNKLLGIACQGAIPLGLMMVPFIE
PANPFATPLEILPEWYFYPVFQILRTVPNKLLGVLAMAAVPVGLLTVPFIE
PANPMSTPAHIVPEWYFLPVYAILRSIPNKLGGVAAIGLVFVSLLALPFIN
PANPLVTPPHIKPEWYFLFAYAILRSIPNKLGGVLALLFSILMLLLVPFLH
PANPLSTPAHIKPEWYFLFAYAILRSIPNKLGGVLALLLSILVLIFIPMLQ
PANPLSTPPHIKPEWYFLFAYAILRSIPNKLGGVLALLLSILILIFIPMLQ
IANPMNTPTHIKPEWYFLFAYSILRAIPNKLGGVIGLVMSILIL..YIMIF
ESDPMMSPVHIVPEWYFLFAYAILRAIPNKVLGVVSLFASILVL..VVFVL
IVDTLKTSDKILPEWFFLYLFGFLKAIPDKFMGLFLMVILLFSL..FLFIL
```

Η Dayhoff αναπαρέστησε την ομοιότητα μεταξύ των αμινοξέων ως αναλογία πιθανοτήτων log2, επίσης, γνωστό ως αποτέλεσμα LOD. Για τον προσδιορισμό της βαθμολογίας LOD ενός αμινοξέος, λαμβάνουμε την log2 του λόγου της παρατηρούμενης συχνότητας σύνδεσης των αμινοξέων με την τυχαία αναμενόμενη συχνότητα του σύνδεσής τους. Αν οι παρατηρούμενες και αναμενόμενες συχνότητες είναι ίσες, το σκορ LOD είναι μηδέν. Μια θετική βαθμολογία δείχνει ότι ένα ζευγάρι των γραμμάτων είναι κοινή, ενώ ένα αρνητικό αποτέλεσμα δείχνει μια λιγότερο πιθανή στοίχιση. Ο γενικός τύπος για κάθε ζεύγος αμινοξέων παρουσιάζεται στην παρακάτω εξίσωση:

$$S_{ij} = \log\left({}^{q_{ij}} \big/ _{p_i p_j} \right)$$

2.4.3 Πίνακες βαθμολόγησης (scoring matrices)

Μια δισδιάστατη μήτρα που περιέχει όλες τις πιθανές βαθμολογίες ζευγών αμινοξέων ονομάζεται μήτρα βαθμολόγησης. Οι πίνακες βαθμολόγησης που ονομάζονται επίσης πίνακες αντικατάστασης, ονομάζονται έτσι διότι οι βαθμολογίες αντιπροσωπεύουν ποσοστά σχετικής εξελικτικής αντικατάστασης. Οι μήτρες βαθμολόγησης με λίγα λόγια αντιπροσωπεύουν την εξέλιξη. Παρακάτω φαίνεται ο πίνακας Blosum62:

```
#  Matrix made from blosum62
#  * column uses minimum score
#  BLOSUM Clustered Scoring Matrix in 1/2 Bit Units
#  Cluster Percentage: >= 62
#  Entropy =   0.6979, Expected =  -0.5209
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  4 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1 -1 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -1 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
```

```
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -1 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0 -1 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1 -1 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -1 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
X -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```

Τα σκορ είναι πραγματικοί αριθμοί, αλλά συνήθως αναπαρίστανται ως ακέραιοι σε αρχεία κειμένου και προγράμματα ηλεκτρονικών υπολογιστών. Για να διατηρηθεί η ακρίβεια, τα αποτελέσματα είναι γενικά πολλαπλασιάζονται με κάποιο συντελεστής κλίμακας πριν από τη μετατροπή τους σε ακέραιους αριθμούς. Για παράδειγμα, ένας βαθμός LOD των -1,609 nats μπορεί να κλιμακωθεί με συντελεστή δύο και στη συνέχεια στρογγυλοποιείται σε μια ακέραια τιμή του -3.

Σκορ που έχουν κλιμακωθεί και μετατρέπονται σε ακέραιους έχουν μια αδιάστατη ποσότητα και ονομάζονται πρώτες βαθμολογίες.

2.4.3.1 Οι πίνακες PAM και BLOSUM

Δύο διαφορετικά είδη των πινάκων βαθμολόγησης αμινοξέων, PAM (Ποσοστό Αποδεκτής μετάλλαξης) και BLOSUM (πίνακες υποκατάστασης μπλοκ), είναι σε ευρεία χρήση. Οι μήτρες PAM δημιουργήθηκαν από τη Margaret Dayhoff και συνεργάτες της και έτσι μερικές φορές αναφέρεται ως τις μήτρες Dayhoff. Αυτές οι πίνακες αξιολόγησης έχουν μια ισχυρή θεωρητική συνιστώσα και κάνουν μερικές εξελικτικές υποθέσεις. Οι μήτρες BLOSUM, από την άλλη πλευρά, είναι πιο εμπειρικοί και προέρχονται από ένα μεγαλύτερο σύνολο δεδομένων. Οι περισσότεροι ερευνητές σήμερα προτιμούν να χρησιμοποιούν BLOSUM μήτρες, διότι πειράματα που χρησιμοποιούν μήτρες BLOSUM έχουν παρατηρηθεί να έχουν μεγαλύτερη ευαισθησία στη στοίχιση ακολουθιών.

Υπάρχουν αρκετές μήτρες PAM, η καθεμία με μια αριθμητική κατάληξη. Η μήτρα PAM1 δομήθηκε με ένα σύνολο των πρωτεϊνών που ήταν όλοι 85 τοις εκατό ή περισσότερο όμοιες η μια με την άλλη. Οι άλλες μήτρες στο σύνολο PAM κατόπιν κατασκευάστηκαν με πολλαπλασιασμό της μήτρας PAM1 με τον εαυτό της: 100 φορές για το PAM100, 160 φορές για το PAM 160 και ούτω καθεξής, σε μια προσπάθεια να διαμορφώσει την πορεία της εξέλιξης της αλληλουχίας. Αν και πολύ θεωρητική, είναι σίγουρα μια λογική προσέγγισης. Υπήρχε μικρή ποσότητα δεδομένων ακολουθιών πρωτεΐνης στα 1970, όταν οι εν λόγω μήτρες. δημιουργήθηκαν, έτσι ώστε αυτή η προσέγγιση να ήταν ένας καλός τρόπος για να προβληθούν σε μεγαλύτερες «αποστάσεις».

Τη δεκαετία του 90 όπου οι «γνωστές» ακολουθίες πρωτεϊνών σχημάτιζαν βάσεις δεδομένων πολύ μεγαλύτερης κλίμακας μπορούσε να γίνει μια πιο εμπειρική προσέγγιση του προβλήματος. Οι πίνακες BLOSUM σχηματίσθηκαν από τμήματα στοιχισμένων γνωστών ακολουθιών χωρίς κενά (μπλοκ ακολουθιών) και επιπλέον έγινε ομαδοποίηση αυτών των μπλοκ σε βάση τοις εκατό ομοιότητας τους. Έτσι τα μπλοκ που χρησιμοποιήθηκαν για να σχηματισθεί ο BLOSUM62 έχουν τουλάχιστο 62% ομοιότητα με κάποιο άλλο μέλος του μπλοκ.

2.4.4 Στατιστική σημαντικότητα των σκορ τοπικής στοίχισης

Μία σημαντική εξέλιξη στη βιολογική σύγκριση σειράς εμφανίστηκε το 1990, όταν Karlin και Altschul δημοσίευσαν στατιστική ανάλυση των τοπικών σκορ ομοιότητας ακολουθίας χωρίς κενά [8], και το πρόγραμμα BLAST ενσωματώνει αυτά τα στατιστικά στοιχεία [4]. Αν και μια μέθοδος για την εκτίμηση της στατιστικής σημαντικότητας των σκορ ομοιότητας ακολουθίας, το πρόγραμμα RDF, που συμπεριλήφθηκε με το πρόγραμμα FASTP [9], μαζί με την υπόδειξη ότι σκορ ομοιότητα αλληλουχίας που ήταν 6 τυπικές αποκλίσεις πάνω από τη μέση τιμή της κατανομής ανακατεμένων σκορ αλληλουχίας (z> 6) ήταν "πιθανώς" σημαντική, δεν υπήρχε στατιστική βάση για την παρατήρηση αυτή. Οι εργασίες από Waterman και Arratia [10] και Karlin and Altschul [8] έδειξαν ότι η τοπική σκορ ομοιότητας, τουλάχιστον για ευθυγραμμίσεις χωρίς διάκενα, περιγράφηκαν με ακρίβεια από την κατανομή ακραίων τιμών, η οποία μπορεί να είναι γραφτεί ως :

$$p(S \geq x) = 1 - \exp\left(-Kmne^{-\lambda x}\right)$$

Όπου τα λ και K μπορούν να υπολογιστούν από τα στοιχεία του πίνακα βαθμολόγησης που θα χρησιμοποιηθεί s_{ij} και τις συνθέσεις των στοιχισμένων ακολουθιών p_i , p_j και m,n τα μήκη των δύο ακολουθιών.

Τα ακριβή στατιστικά στοιχεία ομοιότητας μας επιτρέπουν να διακρίνουμε αξιόπιστα μεταξύ στατιστικά σημαντικών ομοιοτήτων, οι οποίες αντανακλούν ομολογία, και οι ομοιότητες που θα μπορούσαν να έχουν προκύψει κατά τύχη, όπως στις ανάλογες ακολουθίες. Η διαθεσιμότητα των «Karlin-Altschul» στατιστικών στοιχείων στο πρόγραμμα BLAST [4] διαχωρίζεται σε "πρώτης γενιάς" προγράμματα βαθμολόγησης από τις μεθόδους «δεύτερης γενιάς».

Χωρίς ακριβή στατιστικά στοιχεία, είναι αδύνατο να κάνει κανείς μεγάλης κλίμακας ερμηνεία μιας άγνωστης ακολουθίας.

2.4.5 Στατιστική στοιχίσεων χωρίς διάκενα

Τα πρώτα στατιστικά μοντέλα για την τοπική και την ολική βαθμολογία ευθυγράμμισης εφαρμόζονται στα πειράματα παρόμοιων υποκακολουθιών αμινοξέων ή νουκλεοτιδίων, τα οποία είναι ισοδύναμα με ευθυγραμμίσεις χωρίς κενά. Arratia, Gordan, and Waterman [10, 11], και Karlin and Altschul [8,12] απέδειξαν ότι οι τοπικές βαθμολογίες ομοιότητας αναμένεται να ακολουθήσουν την κατανομή ακραίων τιμών. Ο Waterman παρουσιάζει ένα διαισθητικό επιχείρημα [13], όπου επισημαίνει ότι ο αναμενόμενος αριθμός των εμφάνισης κορώνας σε l φορές, σε n ρίψεις ενός νομίσματος είναι $E \approx np^l$ όπου p η πιθανότητα εμφάνισης κορώνας. Αυτή η εξίσωση απορρέει από το γεγονός ότι ο αναμενόμενος αριθμός λήψης κορώνας είναι η πιθανότητα να έρθει κορώνα σε κάθε ρίψη επί τον αριθμό των ρίψεων. Αν ο

μεγαλύτερος αριθμός να έρθει κορώνα είναι 1 τότε $1=np^{R_l}$ οπότε $R_l=\log_{1/p}(n)$. Ο συνολικός αριθμός ρίψης κορώνας είναι ισοδύναμο πείραμα με το να βρεθεί η μεγαλύτερη σε μήκος στοίχιση ή στοίχιση μεγαλύτερης περιοχής σε μια πρωτεΐνη με τη χρησιμοποίηση ενός πίνακα βαθμολόγησης που αποδίδει μια θετική τιμή σε κάποια ομοιότητα πρωτεϊνών και μείον άπειρον σε όλα τα άλλα ενδεχόμενα μη ομοιότητας πρωτεϊνών. Βάσει αυτής της παρατήρησης η πιθανότητα να λάβουμε θετικό σκορ είναι $\sum p_i$ για κάθε πρωτεΐνη p_i που διατηρείται το θετικό σκορ

Το παράδειγμα με τις διαδοχικές ρίψης κορώνας ή βαθμολόγησης με $-\infty$ σε κάθε μη ταίριασμα δείχνει ότι τοπικές ομοιότητες αναμένονται να αυξηθούν ανάλογα με το λογάριθμο του μήκους της ακολουθίας. Κατά την σύγκριση ομοιότητας δύο ακολουθιών πρωτεϊνών, ήτοι $a_{1\ldots m}, b_{1\ldots n}$ ο υπολογισμός της πιθανότητας είναι αρκετά ανάλογος με το προηγούμενο παράδειγμα. Εδώ αντί να υπολογιστεί η πιθανότητα εμφάνισης k διαδοχικών κορωνών όπου $p_k=pp_{k-1}$ θεωρούμε την περίπτωση ομοιότητας ακολουθιών σε m πρωτεΐνες ή ισοδύναμα να δίνεται κορώνα όπου έχουμε ταίριασμα δηλ. $a_i = b_j$. Αν υποτεθεί ότι τα γράμματα ή οι πρωτεΐνες σε δύο ακολουθίες έχουν την ίδια πιθανότητα εμφάνισης, τότε, η πιθανότητα ταιριάσματος μήκους l από a_i, b_j σε a_{i+l-1}, b_{j+l-1} θα είναι p^l. Παρόλα αυτά θα υπάρχουν (m-l+1) x(n-l+1) αφετηρίες ταιριάσματος οπότε η αναμενόμενη τιμή εμφάνισης ταιριάσματος μήκους l θα είναι $E(l) = mnp^l$. Αντιστρέφοντας, το αναμενόμενο μήκος ταιριάσματος δύο ακολουθιών μήκους m και n όταν το σκορ ταιριάσματος είναι θετικό και το σκορ μη ταιριάσματος είναι μείον άπειρον, θα είναι $M_{nm} = \log_{1/p}(mn)$ ή $M_{nm} = 2\log_{1/p}(n)$ όταν m=n. Έτσι η αναμενόμενη τιμή μήκους θα είναι $E(S \geq x) \propto nmp^x$ ή ισοδύναμα $E(S \geq x) \propto nme^{x\ln p}$ ή $nme^{-\lambda x}$ όπου λ=-lnp

Η εργασία [8] επεκτείνει τα παραπάνω αποτελέσματα για τις τοπικές στοιχίσεις. Για να βεβαιωθεί η υπόθεση ότι οι στοιχίσεις είναι τοπικές πρέπει $E(s_{ij}) = \sum_{ij} p_i p_j s_{ij} \leq 0$. Τότε ο αναμενόμενος αριθμός στοιχίσεων με σκορ S θα πρέπει να είναι: $E(S \geq x) = Knme^{-\lambda x}$ Τα K και λ είναι αντίστοιχα οι τιμές αρνητικής σταθεράς διόρθωσης, με την έννοια ότι δεν είναι δυνατό να υπάρξουν nm ανεξάρτητες θέσεις εκκίνησης που μπορούν να παράγουν σκορ S≥x, και κλιμάκωσης του σκορ ώστε να υπολογιστεί σωστά ή πιθανότητα εμφάνισης στοίχισης. Συγκρινόμενο με το λ το K έχει μια μικρή επιρροή στους υπολογισμούς. Για στοιχίσεις χωρίς κενά το λ μπορεί να υπολογιστεί μοναδικά από την εξίσωση

$$\sum_{i,j} p_i p_j e^{\lambda s_{i,j}} = 1$$

Για τον υπολογισμό της συνάρτησης πυκνότητας πιθανότητας χρησιμοποιείται η συνάρτηση poisson η οποία δίνει την συνάρτηση πυκνότητας πιθανότητας να συμβεί ένα γεγονός, όταν είναι γνωστή η αναμενόμενη μέση τιμή να συμβεί το γεγονός αυτό. Η πιθανότητα poisson να πραγματοποιηθούν n γεγονότα όταν η μέση αναμενόμενη τιμή να συμβούν είναι μ δίνεται από τη σχέση $P(n) = e^{-\mu}\mu^{\nu}/\mathrm{n}!$. Γενικότερα επειδή ενδιαφερόμαστε για την εμφάνιση του γεγονότος (ομοιότητας ακολουθιών) για περισσότερα από 1 στοιχεία (πρωτεΐνες) τότε $P(n\geq 1)=1-P_0$. Συνεπώς η συνάρτηση πυκνότητας πιθανότητας που απορρέει θα είναι:

$$P(S \geq x) = 1 - \exp(-\mu) = 1 - \exp(-Knme^{-\lambda x})$$

Η οποία ακολουθεί την γνωστή κατανομή ακραίων τιμών (Extreme Value Distribution).

2.4.6 Στατιστική στοιχίσεων με διάκενα

Τα στατιστικά στοιχεία που αναπτύχθηκαν παραπάνω έχουν μια ισχυρή θεωρητική θεμελίωση μόνο για τις τοπικές ευθυγραμμίσεις που δεν επιτρέπεται να έχουν κενά. Ωστόσο, πολλά υπολογιστικά πειράματα [14-21] και ορισμένα αναλυτικά αποτελέσματα [22] υποδεικνύουν έντονα ότι η ίδια θεωρία ισχύει, όπως και σε ευθυγραμμίσεις με διάκενα. Στις ευθυγραμμίσεις χωρίς διάκενα, οι στατιστικές παράμετροι μπορούν να υπολογιστούν, χρησιμοποιώντας τύπους αναλυτικών, από τις βαθμολογίες υποκατάστασης και τις συχνότητες εμφάνισης πρωτεϊνών των αλληλουχιών που συγκρίνονται. Για ευθυγραμμίσεις με διάκενα , αυτές οι παράμετροι θα πρέπει να εκτιμηθούν από μια μεγάλης κλίμακας σύγκριση των «τυχαίων» .ακολουθιών.

Μερικά προγράμματα αναζήτησης βάσης δεδομένων, όπως FASTA [23] ή διάφορες υλοποιήσεις του αλγορίθμου Smith-Waterman [24], παράγουν βέλτιστα τοπικά σκορ ευθυγράμμισης για την σύγκριση της ακολουθίας ερωτήματος με κάθε ακολουθία στη βάση δεδομένων. Τα περισσότερα από αυτά τα αποτελέσματα συνεπάγονται από άσχετες ακολουθίες, και συνεπώς μπορεί να χρησιμοποιηθεί για να εκτιμηθεί το λάμδα και K [17 - 21]. Η προσέγγιση αυτή αποφεύγει την τεχνική ενός τυχαίου μοντέλου ακολουθιών με τη χρησιμοποίηση πραγματικών ακολουθιών, με επακόλουθο την εσωτερική δομή και τις συσχετίσεις τους, αλλά θα πρέπει να αντιμετωπίσει το πρόβλημα του αποκλεισμού από τα αποτελέσματα εκτίμησης τα ζεύγη των συσχετισμένων ακολουθιών. Τα προγράμματα BLAST έχουν επιτύχει ένα μεγάλο μέρος της ταχύτητάς τους, αποφεύγοντας τον υπολογισμό του σκορ βέλτιστης στοίχισης για όλες τις ακολουθίες, αλλά από μια χούφτα άσχετων ακολουθιών. Ως εκ τούτου, πρέπει να βασίζονται σε ένα προ-εκτίμηση της λάμδα παραμέτρων και K, για ένα επιλεγμένο σύνολο μητρών υποκατάστασης και κόστος διάκενου. Αυτή η εκτίμηση θα μπορούσε να γίνει με τη χρήση πραγματικών ακολουθιών, αλλά στηρίχθηκε αντί σε μοντέλο τυχαίων ακολουθιών [14], το οποίο φαίνεται να αποδίδει αρκετά ακριβή αποτελέσματα [21].

2.5 BLAST

Το BLAST είναι ένα από τα πιο ευρέως χρησιμοποιούμενα προγράμματα βιοπληροφορικής, [2], επειδή αντιμετωπίζει ένα θεμελιώδες πρόβλημα την στοίχιση ακολουθιών και ο ευρετικός αλγόριθμος που χρησιμοποιεί το κάνει πολύ πιο γρήγορο από ότι τον υπολογισμό μιας βέλτιστης ευθυγράμμισης. Αυτή η έμφαση στην ταχύτητα είναι ζωτικής σημασίας που κάνει τον αλγόριθμο πρακτικό για τις τεράστιες βάσεις δεδομένων γονιδιώματος που διατίθενται σήμερα, παρόλο που επόμενοι αλγόριθμοι μπορεί να είναι ακόμη πιο γρήγοροι.

Πριν αναπτυχθούν γρήγοροι αλγόριθμοι όπως ο BLAST και ο FASTA, οι αναζητήσεις σε βάσεις δεδομένων για πρωτεΐνη ή ακολουθίες νουκλεϊκών οξέων ήταν πολύ χρονοβόρες, επειδή χρησιμοποιήθηκε η διαδικασία πλήρους ευθυγράμμισης (π.χ., ο αλγόριθμος Smith-Waterman).

Ενώ ο BLAST είναι ταχύτερος από τον Smith-Waterman, δεν μπορεί να «εγγυηθεί τις βέλτιστες ευθυγραμμίσεις των ακολουθιών ερωτήματος και της βάσης δεδομένων», όπως ο Smith-Waterman κάνει. Η βελτιστότητα Smith-Waterman «εξασφαλίζει την καλύτερη δυνατή απόδοση για την ακρίβεια και τα πιο ακριβή αποτελέσματα» σε βάρος του χρόνου και της υπολογιστικής ισχύος.

Ο BLAST είναι περισσότερο χρονικά αποδοτικός από ό, τι ο FASTA στην αναζήτηση μόνο για τα πιο σημαντικά μοτίβα στις ακολουθίες, αλλά με τη συγκριτική ευαισθησία. Αυτό θα μπορούσε να επιτευχθεί περαιτέρω με την κατανόηση του αλγορίθμου του BLAST όπως περιγράφεται παρακάτω

Παραδείγματα άλλων ερωτημάτων που οι ερευνητές χρησιμοποιούν το BLAST να απαντήσει είναι:

- Τα όποια βακτηριδιακά είδη έχουν μια πρωτεΐνη που σχετίζεται στην καταγωγή με μια ορισμένη πρωτεΐνη με γνωστή αλληλουχία αμινοξέων
- Ποια άλλα γονίδια κωδικοποιούν πρωτεΐνες που εμφανίζουν δομές ή μοτίβα, όπως αυτά που μόλις έχουν καθοριστεί

ο BLAST είναι επίσης συχνά χρησιμοποιείται ως μέρος των άλλων αλγορίθμων που απαιτούν προσέγγιση ομοιότητας ακολουθιών.

Ο αλγόριθμος BLAST και το πρόγραμμα του υπολογιστή που εφαρμόζει αναπτύχθηκαν από τον Stephen Altschul, Warren Gish, και ο David Lipman στο αμερικανικό Εθνικό Κέντρο για τις πληροφορίες βιοτεχνολογίας (NCBI), Webb Miller στο Pennsylvania State University, και Gene Myers στο Πανεπιστήμιο της Αριζόνα . Είναι διαθέσιμο στο διαδίκτυο στην ιστοσελίδα του NCBI. Εναλλακτικές υλοποιήσεις περιλαμβάνουν AB-BLAST (παλαιότερα γνωστή ως WU-BLAST), FSA-BLAST (τελευταία ενημέρωση το 2006), και ScalaBLAST.

Κεφάλαιο 3ο

3 Ο Αλγόριθμος BLAST

Για να εκτελεστεί, ο BLAST απαιτεί μια ακολουθία ερωτήματος για την αναζήτηση, και μια ακολουθία για την αναζήτηση (ονομάζεται επίσης η ακολουθία στόχος – target sequence) ή μια βάση δεδομένων ακολουθιών που περιέχει πολλαπλές τέτοιες ακολουθίες. Ο BLAST θα βρεί υπο-ακολουθίες στην βάση δεδομένων η οποίες θα είναι παρόμοιες με τις υπο-ακολουθίες στο ερώτημα. Σε τυπική χρήση, η αλληλουχία ερωτήματος είναί πολύ μικρότερη από τη βάση δεδομένων, π.χ., το ερώτημα μπορεί να είναι χίλια νουκλεοτίδια, ενώ η βάση δεδομένων είναι αρκετά δισεκατομμύρια νουκλεοτίδια.

Η κύρια ιδέα του BLAST είναι ότι συχνά υπάρχουν ζεύγη τμημάτων υψηλής βαθμολογίας (High Segment Pairs) που περιέχεται σε μια στατιστικά σημαντική ευθυγράμμιση. BLAST ψάχνει για ευθυγραμμίσεις ακολουθίας υψηλής βαθμολόγησης μεταξύ της αλληλουχίας ερωτήματος και αλληλουχίες στη βάση δεδομένων χρησιμοποιώντας μια ευρετική προσέγγιση που προσεγγίζει τον αλγόριθμο Smith-Waterman. Η εξαντλητική προσέγγιση Smith-Waterman είναι πολύ αργή για την αναζήτηση μεγάλων γονιδιωματικών βάσεις δεδομένων όπως η GenBank. Ως εκ τούτου, ο αλγόριθμος BLAST χρησιμοποιεί μια ευρετική προσέγγιση που είναι λιγότερο ακριβής από τον αλγόριθμο Smith-Waterman, αλλά πάνω από 50 φορές πιο γρήγορα. Η ταχύτητα και η σχετικά καλή ακρίβεια του BLAST είναι μεταξύ των βασικών καινοτόμων τεχνικών των προγραμμάτων BLAST.

Μια επισκόπηση του αλγορίθμου BLASTP (μια πρωτεΐνη στην αναζήτηση πρωτεΐνης) έχει ως εξής:

1. **Αφαίρεση περιοχής χαμηλής πολυπλοκότητας ή επαναλήψεις υποακολουθίας στην ακολουθία ερωτήματος**. «Περιοχή χαμηλής πολυπλοκότητας» σημαίνει μία περιοχή μίας ακολουθίας που αποτελείται από μερικά είδη των στοιχείων. Αυτές οι περιοχές μπορεί να δώσουν υψηλή βαθμολογία που μπερδεύουν το πρόγραμμα να βρει τις πραγματικές σημαντικές αλληλουχίες στη βάση δεδομένων, οπότε θα πρέπει να φιλτράρονται. Οι περιφέρειες θα πρέπει να σημειώνεται με ένα X (ακολουθίες πρωτεϊνών) ή N (ακολουθίες νουκλεϊκών οξέων) και, στη συνέχεια, να αγνοούνται από το πρόγραμμα BLAST. Για να φιλτραριστούν οι περιοχές χαμηλής πολυπλοκότητας, χρησιμοποιείται το πρόγραμμα SEG για πρωτεϊνικές ακολουθίες και το DUST πρόγραμμα χρησιμοποιείται για τις ακολουθίες DNA. Από την άλλη πλευρά, το XNU πρόγραμμα χρησιμοποιείται για να καλύψει τις διαδοχικές επαναλήψεις σε αλληλουχίες πρωτεϊνών
2. **Πραγματοποιείται μια k-γραμμάτων λίστα λέξεων της ακολουθίας ερωτήματος.** Λαμβάνει τιμές k = 3, για παράδειγμα, παραθέτονται λέξεις μήκους 3 στην πρωτεϊνική ακολουθία ερωτήματος (k είναι συνήθως 11 για μία ακολουθία DNA) "διαδοχικά", μέχρις ότου το τελευταίο γράμμα της ακολουθίας ερωτήματος να συμπεριληφθεί. Η μέθοδος απεικονίζεται στο σχήμα [...].

3. **Λίστα με τις πιθανές λέξεις που ταιριάζουν**. Αυτό το στάδιο είναι μία από τις κύριες διαφορές μεταξύ BLAST και FASTA. Ο FASTA νοιάζεται για όλες τις κοινές λέξεις στις ακολουθίες βάσεων δεδομένων και το ερώτημα που παρατίθενται στο βήμα 2 Ωστόσο, BLAST νοιάζεται μόνο για τις λέξεις με υψηλό σκορ. Οι βαθμολογίες δημιουργούνται από τη σύγκριση της λέξης στη λίστα του βήματος 2 με όλες τις λέξεις 3 γραμμάτων. Με τη χρήση του πίνακα βαθμολόγησης (μήτρα υποκατάστασης) για να σκοράρει η σύγκριση του κάθε ζεύγους υπολειμμάτων, υπάρχουν 20 ^ 3 πιθανά αποτελέσματα ταιριάσματος για μια λέξη 3 γραμμάτων. Για παράδειγμα, η βαθμολογία που λαμβάνεται με σύγκριση PQG με PEG και PQA είναι 15 και 12, αντίστοιχα. Για το DNA, ένα ταίριασμα βαθμολογείται ως +5 και μια αναντιστοιχία ως -4, ή +2 και -3. Μετά από αυτό, μια λέξη γειτονιά βαθμολογίας κατωφλίου T χρησιμοποιείται για να μειώσει τον αριθμό των πιθανών λέξεων που ταιριάζουν. Οι λέξεις των οποίων τα αποτελέσματα είναι μεγαλύτερη από το κατώφλι T θα παραμείνουν στην πιθανή λίστα ταιριάσματος λέξεων, ενώ εκείνοι με χαμηλότερο σκορ θα πρέπει να απορρίπτονται. Για παράδειγμα, PEG διατηρείται, αλλά PQA εγκαταλείπεται όταν το T είναι 13.
4. **Οργάνωση των υπόλοιπων λέξεων υψηλής βαθμολόγησης σε ένα αποτελεσματικό δένδρο αναζήτησης**. Αυτό επιτρέπει στο πρόγραμμα να συγκρίνει γρήγορα τις λέξεις υψηλής βαθμολόγησης με τις ακολουθίες της βάσης δεδομένων.
5. **Επανάληψη των βημάτων 3 έως 4 για κάθε λέξη k στην ακολουθία ερωτήματος**.
6. **Σάρωση των ακολουθιών της βάσης δεδομένων για την ακριβή αντιστοιχία με τις υπόλοιπες λέξεις με υψηλό σκορ**. Το πρόγραμμα BLAST σαρώνει τις αλληλουχίες της βάσης δεδομένων για το υπόλοιπο λέξη υψηλής βαθμολόγησης, όπως PEG, από κάθε θέση. Εάν βρεθεί μια ακριβή αντιστοιχία, αυτό το ταίριασμα χρησιμοποιείται για επέκταση μιας πιθανής ευθυγράμμισης χωρίς διάκενα μεταξύ του ερωτήματος και της ακολουθίας της βάσης δεδομένων.
7. **Επέκταση των ακριβών στοιχίσεων για ζεύγη τμημάτων υψηλής βαθμολόγησης (HSP)**. Η αρχική έκδοση του BLAST απλώνεται μια μεγαλύτερη ευθυγράμμιση μεταξύ του ερωτήματος και των ακολουθιών βάσης δεδομένων από τα αριστερά και προς τα δεξιά, από την θέση όπου συνέβη η ακριβής αντιστοιχία. Η παράταση δεν σταματά μέχρις ότου η συσσωρευμένη συνολική βαθμολογία του HSP αρχίζει να μειώνεται. Για να κερδηθεί περισσότερος χρόνος, έχει αναπτυχθεί μια νεότερη έκδοση του BLAST, που ονομάζεται BLAST2 ή BLAST με διάκενα. BLAST2 υιοθετεί ένα χαμηλότερο όριο βαθμολογίας λέξεων γειτονιάς για να διατηρηθεί το ίδιο επίπεδο ευαισθησίας για την ανίχνευση ομοιότητα της ακολουθίας. Ως εκ τούτου, η πιθανή λίστα λέξεων που ταιριάζουν στο βήμα 3 γίνεται μεγαλύτερη. Στη συνέχεια, οι ακριβείς περιοχές ταιριάσματος, σε απόσταση A η μία από την άλλη στην ίδια διαγώνιο, θα ενωθούν ως πλέον νέα περιοχή. Τέλος, οι νέες περιοχές επεκτείνονται με την ίδια μέθοδο όπως και στην αρχική έκδοση του BLAST, και τα HSPs »(Τμήμα ζεύγους υψηλής-βαθμολόγησης) αποτελέσματα των εκτεταμένων περιφερειών δημιουργούνται στη συνέχεια με τη χρήση ενός πίνακα υποκατάστασης όπως πριν.
8. **Παράθεση σε λίστα όλων των HSPs στη βάση δεδομένων της οποίας σκορ είναι αρκετά υψηλά.** Παραθέτονται το HSPs των οποίων τα σκορ είναι μεγαλύτερα από την εμπειρικά προσδιορισθείσα βαθμολογία αποκοπής S. Με την εξέταση της κατανομής

των βαθμολογιών ευθυγράμμισης μοντελοποιείται και υπολογίζεται με σύγκριση τυχαίων ακολουθιών, ένα σκορ αποκοπής S έτσι ώστε η τιμή του είναι αρκετά μεγάλη για να εγγυηθεί την σημασία των HSPs.

9. **Υπολογισμός της σημασίας που φέρει το HSP.** Στη συνέχεια ο BLAST αξιολογεί την σημαντικότητα του HSP που ανακαλύφθηκε με αξιοποίηση της κατανομής ακραίων τιμών του Gumbel Σύμφωνα με τη Gumbel EVD, η πιθανότητα p παρατήρησης μιας βαθμολογίας S ίση με ή μεγαλύτερη από x δίνεται από την εξίσωση:

$$p(S \geq x) = 1 - \exp\left(-e^{-\lambda(x-\mu)}\right) \text{ όπου } \mu = \left[\log(Km'n')\right]/\lambda$$

Οι στατιστικές παράμετροι λ και K προσδιορίζονται κατόπιν ταιριάσματος των καμπυλών της κατανομής των σκορ τοπικής στοίχισης χωρίς διάκενα με την Gumbel EVD. Τα λ και K θα εξαρτώνται από τον πίνακα αντικατάστασης (substitution matrix), τις τιμές ποινής για διάκενα και τις συχνότητες εμφάνισης πρωτεϊνών των ακολουθιών. Τα m' και n' ονομάζονται αποτελεσματικά μήκη των ακολουθιών του ερωτήματος και της τρέχουσας ακολουθίας της βάσης δεδομένων αντίστοιχα. Το αρχικό μήκος της αλληλουχίας βραχύνεται στο αποτελεσματικό μήκος για να αντισταθμίσει την επίδραση ακμής (ένα ξεκίνημα ευθυγράμμισης κοντά στο τέλος ενός από τα ερωτήματος ή αλληλουχία της βάσης δεδομένων δεν είναι πιθανό να έχει αρκετή ακολουθία για να χτίσει μια βέλτιστη ευθυγράμμιση). Μπορεί να υπολογιστεί ως:

$$m' = m - \ln(\mathrm{Kmn})/H$$
$$n' = n - \ln(Kmn)/H$$

όπου H είναι η αναμενόμενη μέση βαθμολογία ανά ευθυγραμμισμένο ζεύγος πρωτεϊνών σε μια ευθυγράμμιση των δύο τυχαίων ακολουθιών. Altschul και Gish έδωσε τις τυπικές τιμές, λ = 0,318, {K} = 0,13, και H = 0,40, για τοπική ευθυγράμμιση χωρίς κενά χρησιμοποιώντας την BLOSUM62 ως μήτρα αντικατάστασης. Η χρησιμοποίηση των τυπικών τιμών για την εκτίμηση της σημασίας ονομάζεται η μέθοδος ανίχνευσης του πίνακα αναζήτησης, δεν είναι ακριβής και κάθε φορά πρέπει να γίνεται νέα εκτίμηση. Το αναμενόμενο σκορ E ενός ταιριάσματος βάσης δεδομένων είναι ο αριθμός των φορών που μια άσχετη ακολουθία βάσεων δεδομένων θα αποκτήσει ένα σκορ μεγαλύτερο από S x κατά τύχη. Το αναμενόμενη τιμή E που λήφθηκε σε μια αναζήτηση για μια βάση δεδομένων από D ακολουθίες δίνεται από

$$E \approx 1 - e^{-p(S>x)D}$$

Επιπλέον, όταν p < 0.1 τότε E=pD

Αυτή η προσδοκία ή αναμενόμενη τιμή του "E" (που συχνά αποκαλείται μια βαθμολογία E ή E-value ή e-value) η εκτίμηση της σημασίας του σκορ HSP για στοίχιση χωρίς διάκενα τοπική ευθυγράμμιση αναφέρεται στα αποτελέσματα του BLAST. Ο υπολογισμός που παρουσιάζεται εδώ είναι τροποποιημένο εάν οι μεμονωμένες HSPs συνδυάζονται, όπως όταν γίνονται ευθυγραμμίσεις με διάκενα (που περιγράφεται κατωτέρω), λόγω της μεταβολής των στατιστικών παραμέτρων.

10. **Επέκταση δύο ή περισσοτέρων HSPs σε μια μεγαλύτερη ευθυγράμμιση**. Μερικές φορές, βρίσκουμε δύο ή περισσότερες περιοχές HSP σε μία ακολουθία βάσης

δεδομένων που μπορεί να γίνει σε μεγαλύτερη ευθυγράμμιση. Αυτό παρέχει επιπλέον αποδείξεις για τη σχέση μεταξύ του ερωτήματος και της ακολουθίας βάσης δεδομένων. Υπάρχουν δύο μέθοδοι, η μέθοδος Poisson και η μέθοδος αθροίσματος σκορ, για να συγκριθεί η σημασία των πρόσφατα συνδυασμένων περιοχών HSP. Ας υποθέσουμε ότι υπάρχουν δύο συνδυασμένες περιοχές HSP με τα ζεύγη των σκορ (65, 40) και (52, 45), αντίστοιχα. Η μέθοδος Poisson δίνει μεγαλύτερη σημασία στο σύνολο με τη μέγιστη χαμηλότερη βαθμολογία (45> 40). Ωστόσο, η μέθοδος αθροίσματος σκορ προτιμά το πρώτο σετ, επειδή 65 + 40 (105) είναι μεγαλύτερη από 52 + 45 (97). Η αρχική BLAST χρησιμοποιεί τη μέθοδο Poisson. Το BLAST χωρίς διάκενα και το WU-BLAST χρησιμοποιεί τη μέθοδο του αθροίσματος των βαθμολογιών.

11. **Εκτύπωση της τα τοπικής στοίχισης Smith-Waterman χωρίς διάκενα του ερωτήματος για κάθε μια από τις ακολουθίες της βάσης δεδομένων**. Η αρχική BLAST παράγει μόνο ευθυγραμμίσεις χωρίς διάκενα συμπεριλαμβάνει τις αρχικά εβρέθησες HSPs χωριστά, ακόμα και όταν υπάρχουν περισσότερα από ένα HSP εντοπισμένα σε μία ακολουθία βάσης δεδομένων. Το BLAST2 παράγει ένα ενιαίο ευθυγράμμιση με τα κενά που μπορεί να περιλαμβάνουν το σύνολο των περιοχών HSP που βρέθηκαν αρχικά. Σημειώνεται ότι ο υπολογισμός της βαθμολογίας και των αντίστοιχων τιμών της E περιλαμβάνει τη χρήση των κατάλληλων κυρώσεων για τα διάκενα.
12. **Αναφορά κάθε ταιριάσματος του οποίου η αναμενόμενη βαθμολογία είναι χαμηλότερη από ό, τι η παράμετρος αναμενόμενης τιμής E**.

3.1 Το πρόγραμμα BLAST

Το πρόγραμμα BLAST μπορεί είτε να κατεβεί και να τρέξει ως μια γραμμή εντολών βοηθητικό πρόγραμμα "blastall» ή μέσω web για δωρεάν εφαρμογή. Το BLAST web server, που φιλοξενείται από το NCBI, επιτρέπει σε οποιονδήποτε με ένα web browser να εκτελέσει αναζητήσεις ομοιότητας από τις συνεχώς ενημερωμένες βάσεις δεδομένων πρωτεϊνών και DNA που περιλαμβάνουν τις περισσότερες από τις πρόσφατα ανακαλυφθείσες ακολουθίες οργανισμών.

Το πρόγραμμα BLAST βασίζεται σε μια μορφή ανοιχτού κώδικα, δίνοντας σε όλους πρόσβαση σε αυτό και να μπορέσουν να έχουν τη δυνατότητα να αλλάξουν τον κώδικα του προγράμματος. Αυτό έχει οδηγήσει στη δημιουργία πολλών BLAST "spin-offs".

Υπάρχει τώρα μια χούφτα των διαφορετικών προγραμμάτων BLAST διαθέσιμα, τα οποία μπορούν να χρησιμοποιηθούν ανάλογα με το τι έκανε ένας μελετητής τι προσπαθεί να κάνει και με τι εργάζεται. Αυτά τα διαφορετικά προγράμματα ποικίλλουν στην είσοδο ακολουθίας ερωτήματος, η βάσης δεδομένων που αναζητείται, και αυτό που συγκρίνεται. Αυτά τα προγράμματα και τα στοιχεία τους αναφέρονται παρακάτω:

BLAST είναι στην πραγματικότητα μια οικογένεια προγραμμάτων (όλα περιλαμβάνονται στην blastall εκτελέσιμο). Αυτά περιλαμβάνουν:

Νουκλεοτιδίων-νουκλεοτιδίων BLAST (blastn)

Το πρόγραμμα αυτό, δίνεται ένα ερώτημα DNA, επιστρέφει τις πιο όμοιες ακολουθίες DNA από τη βάση δεδομένων DNA που καθορίζει ο χρήστης.

Πρωτεΐνης-πρωτεΐνης BLAST (BLASTP)

Το πρόγραμμα αυτό, δίνεται ένα ερώτημα πρωτεΐνη, επιστρέφει τις πιο όμοιες ακολουθίες πρωτεϊνών από τη βάση δεδομένων της πρωτεΐνης που καθορίζει ο χρήστης.

Επαναληπτικός ως προς τη θέση (Position Specific iterative)BLAST (PSI-BLAST) (blastpgp)

Το πρόγραμμα αυτό χρησιμοποιείται για να βρεθούν μακρινοί συγγενείς μιας πρωτεΐνης. Πρώτα, ένας κατάλογος όλων των στενά σχετικών πρωτεϊνών δημιουργείται. Αυτές οι πρωτεΐνες συνδυάζονται σε ένα γενικό "προφίλ" ακολουθιών, η οποία συνοψίζει σημαντικά χαρακτηριστικά παρόντα σε αυτές τις ακολουθίες. Ένα ερώτημα με τη βάση δεδομένων πρωτεΐνης κατόπιν τρέξει με αυτό το προφίλ, και μια μεγαλύτερη ομάδα των πρωτεϊνών που βρέθηκαν. Αυτή η μεγαλύτερη ομάδα χρησιμοποιείται για να κατασκευάσει ένα άλλο προφίλ, και η διαδικασία επαναλαμβάνεται.

Με τη συμπερίληψη σχετικών πρωτεϊνών στην αναζήτηση, ο PSI-BLAST είναι πολύ πιο ευαίσθητος στην μαζεύοντας μακρινό-εξελικτικές σχέσεις σε αντίθεση με το βασικό πρόγραμμα BLAST πρωτεΐνης-πρωτεΐνης. Περισσότερες λεπτομέρειες δίνονται παρακάτω.

Μετάφραση πρωτεΐνης - Νουκλεοτιδίων 6-πλαισίων (blastx)

Το πρόγραμμα αυτό συγκρίνει τα έξι πλαισίων εννοιολογικά προϊόντα μετάφρασης μιας ακολουθίας ερωτήματος νουκλεοτιδίου (και οι δύο κλώνοι) έναντι μιας βάσης δεδομένων ακολουθιών πρωτεΐνης.

6- πλαισίων νουκλεοτιδίων μετάφραση-νουκλεοτιδίων μετάφρασης 6-πλαίσιων (tblastx)

Αυτό το πρόγραμμα είναι η πιο αργή της οικογένειας BLAST. Μεταφράζει την νουκλεοτιδική ακολουθία ερωτήματος σε όλες τις έξι πιθανών πλαίσιων περιπτώσεις και τη συγκρίνει με τις μεταφράσεις έξι πλαισίων μιας βάσης δεδομένων νουκλεοτιδικής αλληλουχίας. Ο σκοπός του tblastx είναι να βρεθούν πολύ μακρινές σχέσεις μεταξύ των ακολουθιών νουκλεοτιδίων.

Πρωτεΐνης-νουκλεοτιδίων μετάφραση 6-πλαισίων (tblastn)

Το πρόγραμμα αυτό συγκρίνει ένα ερώτημα πρωτεΐνης έναντι των έξι πλαισίων ανάγνωσης μιας βάσης δεδομένων νουκλεοτιδικών ακολουθιών.

Μεγάλος αριθμός των ακολουθιών ερωτήματος (Megablast)

Κατά τη σύγκριση μεγάλου αριθμού των ακολουθιών εισόδου μέσω της γραμμής εντολών BLAST, "Megablast" είναι πολύ ταχύτερη από ό, τι αν τρέχει ο BLAST πολλές φορές. Θα συνενώνει πολλές ακολουθίες εισόδου μαζί για να σχηματίσουν μια μεγάλη ακολουθία πριν από την αναζήτηση στη βάση δεδομένων Το BLAST, στη συνέχεια πραγματοποιεί μετα-αναλύσεις των αποτελεσμάτων αναζήτησης για να συγκεντρώσει επιμέρους ευθυγραμμίσεις με τις στατιστικές τιμές.

3.2 BLAST με διάκενα και PSI-BLAST: Μια νέα γενιά προγραμμάτων στοίχισης πρωτεϊνών

Όπως είδαμε, παραλλαγές του αλγορίθμου BLAST [4] έχουν ενσωματωθεί σε πολλά δημοφιλή προγράμματα για την αναζήτηση πρωτεϊνών και DNA σε αντίστοιχες βάσεις δεδομένων για τις ομοιότητες ακολουθιών. Τα προγράμματα BLAST έχουν γραφτεί για να συγκρίνουν ερωτήματα πρωτεϊνών ή DNA με πρωτεΐνη ή DNA βάσεις δεδομένων σε οποιοδήποτε συνδυασμό, με ακολουθίες DNA που συχνά υποβάλλονται εννοιολογική μετάφραση πριν, ενώ δεν υπάρχει σύγκριση. Εδώ θα χρησιμοποιήσουμε το πρόγραμμα BLASTP, το οποίο συγκρίνει ερωτήματα πρωτεϊνών σε βάσεις δεδομένων πρωτεϊνών, ως πρωτότυπο BLAST, αν και οι ιδέες που παρουσιάζονται επεκτείνονται άμεσα και σε άλλες εκδόσεις που περιλαμβάνουν τη μετάφραση ενός ερωτήματος DNA ή βάσης δεδομένων. Μερικές από τις βελτιώσεις που περιγράφονται εδώ εφαρμόζονται, καθώς και προς Σύγκριση DNA-DNA, αλλά δεν έχουν ακόμα εφαρμοστεί.

Το BLAST είναι μια ευριστική που προσπαθεί να βελτιστοποιήσει ένα ειδικό μέτρο ομοιότητας. Αυτό επιτρέπει μια ανταλλαγή μεταξύ της ταχύτητας και ευαισθησίας, με τη ρύθμιση της παραμέτρου «κατώφλι», Τ. Α υψηλότερη τιμή του Τ αποδίδει μεγαλύτερη ταχύτητα, αλλά και μια αυξημένη πιθανότητα να χαθούν αδύναμες ομοιότητες. Το πρόγραμμα BLAST απαιτεί χρόνο ανάλογη με το γινόμενο των μηκών της αλληλουχίας ερωτήματος και τη βάση δεδομένων που ερευνήθηκε. Δεδομένου ότι ο ρυθμός της αλλαγής στο μέγεθος των δεδομένων υπερβαίνει σήμερα ότι ταχύτητες επεξεργαστή, οι υπολογιστές που τρέχουν BLAST υποβάλλονται σε αύξηση του φορτίου. Ωστόσο, ο συνδυασμός πολλών νέων αλγοριθμικών ιδέες επιτρέπουν μια νέα έκδοση του BLAST να επιτευχθεί με βελτιωμένη ευαισθησία σε σημαντικά αυξημένη ταχύτητα

Οι αλλαγές που έχουν υπεισέλθει είναι οι εξής

(i) Για την αύξηση της ταχύτητας, το κριτήριο για την επέκταση ζεύγη λέξεων έχει τροποποιηθεί. Το αρχικό πρόγραμμα BLAST επιδιώκει ομοιότητα ζευγών λέξεων βαθμολογίας των οποίων κατά τη στοίχιση να είναι τουλάχιστον Τ. Κάθε τέτοιος «εντοπισμός» επεκτείνεται στη συνέχεια, για να ελεγχθεί αν περιέχεται σε στοίχιση με υψηλό σκορ. Για την προεπιλεγμένη τιμή Τ, αυτό το στάδιο επέκτασης καταναλώνει το μεγαλύτερο μέρος του χρόνου επεξεργασίας. Η νέα μέθοδος «δύο-εντοπισμών» (two hit) απαιτεί την ύπαρξη δύο μη επικαλυπτόμενων ζεύγη λέξεων στην ίδια διαγώνιο, και σε απόσταση Α από το ένα το άλλο, πριν να γίνει η επέκταση. Για να επιτευχθεί συγκρίσιμη ευαισθησία, η παράμετρος ορίου Τ πρέπει να μειωθεί, παρέχοντας περισσότερα χτυπήματα από ό, τι στο παρελθόν. Ωστόσο, επειδή μόνο ένα μικρό ποσοστό αυτών των συμπτώσεων επεκτείνονται, το μέσο ποσό των υπολογισμών που απαιτούνται μειώνεται.

(ii) Έχει προστεθεί η δυνατότητα να παραχθούν ευθυγραμμίσεις με χάσματα. Το αρχικό πρόγραμμα BLAST συχνά βρίσκει πολλές ευθυγραμμίσεις που περιλαμβάνει μια ενιαία ακολουθία βάσης δεδομένων η οποία, όταν εξετάζεται από κοινού με την ακολουθία ερώτημα, είναι στατιστικά σημαντικές. Αν αγνοηθεί μία από αυτές τις ευθυγραμμίσεις μπορεί να θέσει σε κίνδυνο το συνδυασμένο αποτέλεσμα. Με την εισαγωγή ενός αλγορίθμου για την παραγωγή στοιχίσεων με διάκενα, καθίσταται αναγκαίο να βρει μόνο μια και όχι όλες τις στοιχίσεις χωρίς διάκενα και να υπαχθούν όλες οι υποακολουθίες σε ένα σημαντικό

αποτέλεσμα. Αυτό επιτρέπει η παράμετρος Τ να αυξηθεί, αυξάνοντας την ταχύτητα της αρχικής σάρωσης της βάσης δεδομένων.

Ο νέος αλγόριθμος χρησιμοποιεί, για την ευθυγράμμιση με διάκενα, δυναμικό προγραμματισμό να επεκτείνει ένα κεντρικό ζεύγος ευθυγραμμισμένων πρωτεϊνών και στις δύο κατευθύνσεις. Για την ταχύτητα, νωρίτερα, είχαν αναπτυχθεί ευριστικές μέθοδοι [14,23] που περιόριζαν τις στοιχίσεις που παράγονται σε μια προκαθορισμένη λωρίδα της πορείας γραφήματος δυναμικού προγραμματισμού [25]. Η νέα προσέγγισή θεωρεί μόνο στοιχίσεις που στη βαθμολογία δεν υπερβαίνουν σε σκορ Xg χαμηλότερα από το καλύτερο σκορ που υπάρχει. Έτσι, ο αλγόριθμος είναι σε θέση να προσαρμόσει καλύτερα την περιοχή του γραφήματος δεδομένων

(Iii) Οι αναζητήσεις BLAST μπορεί να επαναλαμβάνονται, με χρησιμοποίηση μήτρας βαθμολογίας συγκεκριμένων θέσεων που δημιουργείται από σημαντικές ευθυγραμμίσεις που βρέθηκαν στην επανάληψη i και χρησιμοποιείται για την επόμενη επανάληψη i + 1. Συνήθως μέθοδοι αναζήτησης μοτίβου ή προφίλ συχνά είναι πολύ πιο ευαίσθητες από τις μεθόδους σύγκρισης ζευγών στην ανίχνευση μακρινών σχέσεων. Ωστόσο, η δημιουργία ενός σύνολου μοτίβων ή ένα προφίλ που περιγράφει μια οικογένεια πρωτεϊνών, και την αναζήτηση μιας βάσης δεδομένων με αυτά, τυπικά εμπλέκει τρέξιμο πολλών διαφορετικών προγραμμάτων, με σημαντική παρέμβαση του χρήστη σε διάφορα στάδια. Ο αλγόριθμος BLAST είναι εύκολα γενικεύσιμος να χρησιμοποιήσει μια αυθαίρετη ειδική μήτρα βαθμολογίας θέσης αντί μιας ακολουθίας ερωτήματος και τη συσχετισμένη μήτρα υποκατάστασης. Ως εκ τούτου, έχουμε αυτοματοποιημένη τη διαδικασία της δημιουργίας μιας τέτοιας μήτρας από την έξοδο που παράγεται από μια αναζήτηση BLAST, και να προσαρμοστεί ο αλγόριθμος BLAST να εκμεταλλευτεί αυτή την μήτρα ως είσοδο. Ο προκύπτον επανάληψης συγκεκριμένης θέσης αλγόριθμος BLAST, ή PSI-BLAST, και το αντίστοιχο πρόγραμμα ενδέχεται να μην είναι τόσο ευαίσθητο με τα διαθέσιμα προγράμματα βέλτιστης αναζήτησης μοτίβου, αλλά η ταχύτητα και η ευκολία της χρήσης του μπορεί να φέρει τη δύναμη αυτών των μεθόδων σε μια πιο κοινή και διαδεδομένη χρήση.

3.2.1 Η μέθοδος δύο εντοπισμών

Η κεντρική ιδέα του αλγορίθμου BLAST είναι ότι μια στατιστικά σημαντική στοίχιση είναι πιθανό να περιέχει ένα ζεύγος λέξεων υψηλής βαθμολόγησης. Ο BLAST πρώτα σαρώνει τη βάση δεδομένων για τις λέξεις (συνήθως μήκους τριών για πρωτεΐνες) που σκοράρουν τουλάχιστον Τ όταν ευθυγραμμισμένη με κάποια λέξη εντός της ακολουθίας ερωτήματος. Κάθε ευθυγραμμισμένο ζεύγος λέξεων που πληροί την προϋπόθεση αυτή ονομάζεται ένας εντοπισμός. Το δεύτερο στάδιο των ελέγχων του αλγορίθμου διερευνάται εάν κάθε εντοπισμός βρίσκεται μέσα σε στοίχιση με το σκορ επαρκές να αναφερθεί. Αυτό γίνεται με την επέκταση ενός εντοπισμού και στις δύο κατευθύνσεις, μέχρι το σκορ της τρέχουσας στοίχισης έχει μειωθεί περισσότερο από ό, τι Χ κάτω από την μέγιστη βαθμολογία πού είχε επιτευχθεί. Αυτό το βήμα επέκτασης είναι υπολογιστικά πολύ δαπανηρό, με τις παραμέτρους Τ και Χ αναγκαίες για την επίτευξη εύλογης ευαισθησίας σε ασθενείς στοιχίσεις, το στάδιο επέκτασης τυπικά αντιπροσωπεύει> 90% του χρόνου εκτέλεσης του BLAST. Επομένως, είναι επιθυμητό να μειωθεί η αριθμός των επεκτάσεων που εκτελούνται.

Ο νέος και εκλεπτυσμένος αλγόριθμος βασίζεται στην παρατήρηση ότι μία HSP ενδιαφέροντος είναι πολύ μεγαλύτερη από ό, τι ένα μόνο ζεύγος λέξεων, και μπορεί επομένως να συνεπάγεται πολλαπλούς εντοπισμούς στην ίδια διαγώνιο και μέσα σε μια σχετικά μικρή απόσταση η μία της άλλης. (Η διαγώνιος από έναν εντοπισμό που περιλαμβάνει λέξεις που αρχίζουν στις θέσεις (x1, x2) από τις ακολουθίες της βάσης δεδομένων και το ερώτημα μπορεί να οριστεί ως x1 - x2 Η απόσταση μεταξύ δύο χτυπήματα στην ίδια διαγώνιο είναι η διαφορά μεταξύ της πρώτης από τις συντεταγμένες τους.). Αυτό το συμπέρασμα μπορεί να χρησιμοποιηθεί για τον πιο αποτελεσματικό εντοπισμό HSPs. Συγκεκριμένα, επιλέγουμε ένα παράθυρο μήκους A, και να επικαλείται η επέκταση μόνο όταν οι δύο μη-επικαλυπτόμενοι εντοπισμοί που βρέθηκαν σε κοντινή απόσταση A ο ένας από τον άλλον στην ίδια διαγώνιο. Κάθε εντοπισμός που επικαλύπτει τον πιο πρόσφατο αγνοείται. Η αποτελεσματική εκτέλεση απαιτεί έναν πίνακα για την καταγραφή, για κάθε διαγώνιο, της πρώτης συντεταγμένης του πιο πρόσφατου εντοπισμού που βρέθηκε. Δεδομένου ότι οι αλληλουχίες βάσεων δεδομένων σαρώνονται διαδοχικά, αυτή η συντεταγμένη αυξάνει πάντα για διαδοχικούς εντοπισμούς. Η ιδέα της αναζήτησης πολλαπλών εντοπισμών στο ίδια διαγώνια χρησιμοποιήθηκε για πρώτη φορά στο πλαίσιο των βιολογικών ερευνών σε βάσεις δεδομένων από Wilbur και Lipman [26].

Επειδή χρειάζονται δύο εντοπισμοί ταιριάσματος και όχι ένα για να επικαλεστεί η επέκταση, η παράμετρος ορίου T πρέπει να μειωθεί για να διατηρηθεί συγκρίσιμη η ευαισθησία. Το αποτέλεσμα είναι ότι γίνονται πολλοί περισσότεροι εντοπισμοί μοναδικού στοιχείου, αλλά μόνο ένα μικρό κλάσμα από αυτούς έχουν συναφές δεύτερο εντοπισμό στην ίδια διαγώνιο που να ενεργοποιεί μια επέκταση. Η μεγάλη πλειοψηφία των εντοπισμών μπορεί να απορριφθεί μετά τον δευτερεύοντα υπολογισμό τους, τσεκάροντας, για την κατάλληλη διαγώνια, τη συντεταγμένη του πιο πρόσφατου εντοπισμού, στον έλεγχο κατά πόσο είναι σε κοντινή απόσταση με A του με τον τρέχον εντοπισμό, και, τέλος, κατά την αντικατάστασή των παλιών με τις νέες συντεταγμένες. Εμπειρικά, η οικονομία υπολογιστικής ισχύος που απαιτείται συμβαίνει λόγω των λιγότερων επεκτάσεις που προκαλούνται και που υπερκαλύπτονται από το πρόσθετο υπολογισμό που απαιτείται για την επεξεργασία του μεγαλύτερου αριθμού επισκέψεων.

Για να αναλυθούν οι σχετικές ταχύτητες των one-hit και τις μεθόδους δύο-hit, χρησιμοποιώντας τις παραμέτρους T=13 A=40 και τα ίδια λ και K, παρατηρούμε ότι η μέθοδος δύο-hit παράγει κατά μέσο όρο ~ 3,2 φορές, πιο πολλούς εντοπισμούς, αλλά μόνο ~ 0,14 φορές, περισσότερες επεκτάσεις. Επειδή χρειάζεται περίπου ένα ένατο όσο διάστημα για να αποφασίσει εάν ένα ταίριασμα χρειάζεται να επεκταθεί στην πραγματικότητα να την παρατείνει, το στάδιο επεξεργασίας με τη μέθοδο δύο-hit είναι περίπου δύο φορές ταχύτερο όσο το ίδιο συστατικό της μεθόδου one-hit.

3.2.2 Επαναληπτική εφαρμογή του BLAST σε θέση-ειδικούς πίνακες σκορ

Αναζητήσεις σε βάσεις δεδομένων χρησιμοποιώντας τις θέση-ειδικές μήτρες σκορ, που ονομάζονται επίσης προφίλ ή μοτίβα, συχνά βρισκόμαστε σε πολύ καλύτερη θέση να ανιχνεύσουμε αδύναμες σχέσεις από ό, τι οι αναζητήσεις σε βάσεις δεδομένων που χρησιμοποιούν μια απλή ακολουθία ως ερώτημα [27-37]. Χρησιμοποιώντας αυτές τις μεθόδους, όμως, εμπλέκει συχνά τη χρήση πολλών διαφορετικών προγραμμάτων και σε αρκετά μεγάλο απαιτείται ένας βαθμός εξειδίκευσης. Κατά συνέπεια, να καταστήσει τη

δύναμη των αναζητήσεων μοτίβου πιο άμεσα διαθέσιμη, έχει γραφτεί μια διαδικασία για να κατασκευαστεί μια θέση-ειδική μήτρα βαθμολογίας αυτόματα από την έξοδο του ένα τρέξιμο του BLAST, και να τροποποιηθεί ο BLAST ώστε να λειτουργεί με τη χρήση μιας τέτοιας μήτρας στη θέση ενός απλού ερωτήματος. Το προκύπτον πρόγραμμα PSI-BLAST συχνά είναι ουσιαστικά πιο ευαίσθητο από ό, τι το αντίστοιχο αρχικό πρόγραμμα BLAST, αλλά για κάθε επανάληψη να διαρκεί λίγο περισσότερο από ό, τι στο αρχικό πρόγραμμα BLAST για να τρέξει. Σε σχετικές εργασίες, Henikoff και Henikoff [38] έχουν περιγράψει τον τρόπο, την μικρή τροποποίηση στον κώδικα του BLAST, έτσι ώστε να μπορεί να λειτουργεί σε μια θέση ειδική μήτρα βαθμολογίας, μια μοναδική τεχνητή ακολουθία δηλαδή η οποία προσεγγίζει μια τέτοια μήτρα μπορεί να χρησιμοποιηθεί ως ένα ερώτημα με το αρχικό πρόγραμμα BLAST.

Η κατασκευή μιας θέσης ειδικού πίνακα βαθμολογίας είναι μια διαδικασία πολλών σταδίων, και σε κάθε στάδιο η επιλογή θα πρέπει να γίνεται μεταξύ μια σειράς από εναλλακτικές διαδρομές. Η δημιουργία αυτού του πίνακα καθοδηγείται από τους στόχους της αυτόματης λειτουργίας, της ταχύτητας της εκτέλεσης, και τη γενική απλότητα. Τα θέματα που συζητούνται παρακάτω είναι: (i) η γενική αρχιτεκτονική της μήτρας βαθμολογίας (ii) η κατασκευή της πολλαπλής στοίχισης από την οποία προέρχεται η μήτρα (iii) τα βάρη για τις ακολουθίες εντός της πολλαπλής ευθυγράμμισης, και η αξιολόγηση του αποτελεσματικού αριθμού από ανεξάρτητες παρατηρήσεις από τις οποίες θα δημιουργείται (iv) εκτίμηση των συχνοτήτων στόχου, και η κατασκευή των βαθμολογιών της μήτρας (v) εφαρμογή του BLAST σε μια θέση-ειδική μήτρα, και η στατιστική αξιολόγηση των αποτελεσμάτων αναζήτησης.

3.2.2.1 Αρχιτεκτονική του πίνακα των σκορ

Η στοίχιση μιας απλής ακολουθίας με ένα μοτίβο ενσωματώνεται από μία θέση-ειδική μήτρα βαθμολογίας είναι σχεδόν εντελώς ανάλογη προς την στοίχιση δύο απλών αλληλουχιών. Η μόνη πραγματική διαφορά είναι ότι το σκορ για την ευθυγράμμιση ένα γράμμα με μία θέση μοτίβου δίνεται από την ίδια τη μήτρα, μάλλον παρά με αναφορά σε μια μήτρα υποκατάστασης. Για τις πρωτεΐνες, ένα ερώτημα του μήκους L και μια μήτρα υποκατάσταση διαστάσεων 20 x 20 αντικαθίστανται από μία θέση-ειδική μήτρα διαστάσεων L × 20. Θέση-ειδικές κυρώσεις διακένου μπορεί να οριστούν επίσης ως [33, 39]. Όπως και με ζευγαρωτή σύγκριση ακολουθιών, μπορεί κανείς να επιλέξει ανάμεσα εύρεση της καλύτερης ολικής στοίχισης της μήτρας και της απλής ακολουθίας [2], βρίσκοντας την καλύτερη ευθυγράμμιση της πλήρους μήτρας με ένα τμήμα της αλληλουχίας [40], και την εξεύρεση των καλύτερων τοπικών στοιχίσεων μεταξύ της μήτρας και της ακολουθίας [24].

Θέση-ειδικές μήτρες σκορ πρωτεΐνης αντλούν ενέργεια από δύο πηγές. Το πρώτο είναι βελτιωμένη εκτίμηση των πιθανοτήτων με τις οποίες τα αμινοξέα εμφανίζονται σε διάφορες θέσεις του μοτίβου, που οδηγεί σε ένα πιο ευαίσθητο σύστημα βαθμολόγησης. Η δεύτερη είναι ο σχετικά ακριβής ορισμός των ορίων των σημαντικών μοτίβων. Απαιτώντας την πλήρη στοίχιση ενός ή περισσοτέρων μοτίβων, αντί να επιδιώκεται μια αυθαίρετη τοπική στοίχιση, το μέγεθος του χώρου αναζήτησης μπορεί να μειωθεί σημαντικά, μειώνοντας έτσι το επίπεδο του τυχαίου θορύβου. Δυστυχώς, υπάρχουν πολλά εμπόδια για την αυτοματοποίηση και την οριοθέτηση μιας σειράς μοτίβων από την έξοδο της έρευνας μιας βάσης δεδομένων. Η ακολουθία ερωτήματος μπορεί να περιέχει μια ποικιλία από διαφορετικούς τομείς, και να μοιράζεται διάφορα υποσύνολα αυτών με διαφορετικές πρωτεΐνες στη βάση δεδομένων.

Επιπλέον, ορίζοντας την κατάλληλη έκταση, ακόμη και ενός μόνο μοτίβου μπορεί να είναι εξαιρετικού ενδιαφέροντος εργασία [41]

Ως εκ τούτου, έχει επιλέγει η παραίτηση από τα δυνητικά πλεονεκτήματα του περιορισμού του μήκους των παράγωγων μήτρων, και στη συνέχεια, απαιτώντας αυτούς να στοιχίζονται πλήρως με τα τμήματα των ακολουθιών βάσεων δεδομένων [39]. Αντ' αυτού, κάθε μήτρα που κατασκευάζεται έχει μήκος ακριβώς ίση προς εκείνη της αρχικής ακολουθίας του ερωτήματος. Κατά την αναζήτηση της βάσης δεδομένων με μια τέτοια μήτρα, επιδιώκουμε τοπικές ευθυγραμμίσεις, σε πλήρη αναλογία με εκείνες που ζητείται από το BLAST, όταν χρησιμοποιείται για την απλή σύγκριση ακολουθίας - ακολουθίας. Τέλος, δεν γίνεται προσπάθεια να υπολογιστούν συγκεκριμένες θέσεις σκορ για το διάκενο για χρήση με συγκεκριμένη υποκατάσταση. θέσης-σκορ Αντί αυτού, σε κάθε επανάληψη του PSI-BLAST, χρησιμοποιούνται οι ίδιες βαθμολογίες διακένου που χρησιμοποιούνται στην πρώτη, απλή εκτέλεση του BLAST. Ο λόγος είναι ότι δεν υπάρχει καλή θεωρία για την εξαγωγή του κόστους διακένου από μια πολλαπλή ευθυγράμμιση και ότι, όπως θα συζητηθεί παρακάτω, αποφεύγοντας με τα θέση-ειδικά κόστη διακένου μπορεί να γίνει μια λογική εκτίμηση της στατιστικής σημασίας των προκυπτόντων τοπικών ευθυγραμμίσεων.

3.2.2.2 Κατασκευή των πολλαπλών στοιχίσεων

Για τη δημιουργία πολλαπλών στοιχίσεων από την έξοδο BLAST, απλά συλλέγονται όλα τα τμήματα ακολουθίας βάσεων δεδομένων που έχουν στοιχηθεί με το ερώτημα με αναμενόμενη τιμή κάτω από το όριο, με προεπιλογή, το 0,01. Το ερώτημα χρησιμοποιείται ως κύριο ή πρότυπο, για την κατασκευή μιας πολλαπλής στοίχισης M. Κάθε σειρά (δηλαδή, τμήμα βάσης δεδομένων) πανομοιότυπο με τμήμα του ερωτήματος με το οποίο στοιχίζεται εκκαθαρίζεται, και μόνο ένα αντίγραφο διατηρείται με γραμμές που είναι > 98% ταυτόσημες η μια με την άλλη. Τα κατά ζεύγη στήλες στοίχισης που αφορούν τους χαρακτήρες κενού που εισάγονται στο ερώτημα απλώς αγνοούνται, έτσι ώστε το M να έχει ακριβώς το ίδιο μήκος με το ερώτημα. Επειδή έχουμε να κάνουμε με τις τοπικές ευθυγραμμίσεις, οι στήλες του M μπορεί να περιλαμβάνουν διαφορετικό αριθμό των σειρών, και πολλές στήλες μπορεί να μην περιλαμβάνουν τίποτα, αλλά μόνο το ερώτημα. Δεν γίνεται καμία προσπάθεια να βελτιωθεί το M συγκρίνοντας ακολουθίες της βάσης δεδομένων μια με την άλλη, ή με οποιαδήποτε άλλη αληθινή διαδικασία πολλαπλής ευθυγράμμισης.

Όπως θα συζητηθεί, οι μήτρες βαθμολογιών που κατασκευάστηκαν για μία δεδομένη στήλη ευθυγράμμισης πρέπει να εξαρτάται όχι μόνο από τις πρωτεΐνες που εμφανίζονται εκεί, αλλά και από εκείνα που αποτελούν άλλες στήλες. Για να γίνει αυτή η εξάρτηση εύκολη στη τυποποίηση, ωστόσο, θα πρέπει να ψαλιδιστούν οι πρώτες πολλαπλές στοιχίσεις M μας σε μια απλούστερη και «μειωμένη» στοίχιση. Αυτός ο ψαλιδισμός γίνεται ανεξάρτητα για κάθε στήλη, έτσι ώστε η μειωμένη πολλαπλή ευθυγράμμιση M_C γενικά θα ποικίλει από μία στήλη C έως την επόμενη Για την κατασκευή του M_C, πρέπει πρώτα να προσδιοριστεί το σύνολο R των ακολουθιών περιλαμβάνει να είναι ακριβώς εκείνα που συμβάλλουν μια πρωτεΐνη στη στήλη C. Στην συνέχεια ορίζουμε τις στήλες του MC να είναι ακριβώς εκείνες οι στήλες του M κατά την οποία όλες οι ακολουθίες της R να εκπροσωπούνται. Κατά την κατασκευή, η μειωμένη πολλαπλή ευθυγράμμιση M_C έχει πρωτεΐνες ή χαρακτήρες κενού σε κάθε σειρά και στήλη (Σχ.

5α), και συνεπώς είναι δεκτική προς τους διάφορους χειρισμούς που περιγράφονται κατωτέρω.

3.2.2.3 Βάρη των ακολουθιών

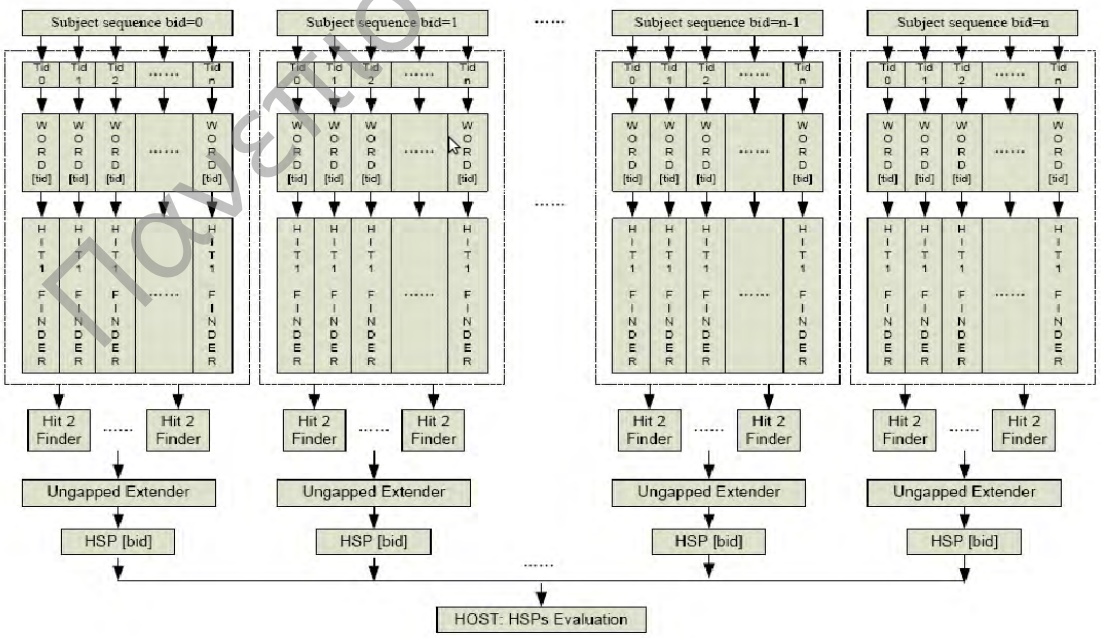
Κατά την κατασκευή ενός πίνακα βαθμολογίας από μια πολλαπλή στοίχιση, είναι λάθος να δοθεί για όλες τις ακολουθίες της ευθυγράμμισης ίσο βάρος. Μια μεγάλη σειρά από στενά σχετιζόμενες ακολουθίες φέρνει λίγο περισσότερες πληροφορίες από ό, τι ένα απλό μέλος, αλλά και μόνο το μέγεθός του μπορεί να επιτρέψει εύκολα να «πλειοψηφήσει» από ένα μικρό αριθμό πιο αποκλίνουσες σειρές. Ένας τρόπος να υπερβληθεί αυτή τη δυσκολία είναι να εκχωρηθούν βάρη στις διάφορες ακολουθίες, με εκείνες που έχουν πολλούς στενούς συγγενείς να λαμβάνουν μικρότερο βάρος. Οι πολλές μέθοδοι στάθμισης ακολουθιών που έχουν προταθεί [42-50] συχνά παράγουν περίπου ισοδύναμα αποτελέσματα. Λόγω της ταχύτητας και την απλότητα της, έχει εφαρμοστεί μια τροποποιημένη εκδοχή της μεθόδου στάθμισης ακολουθιών του Henikoff και Henikoff [46]. Οι χαρακτήρες διακένου αντιμετωπίζονται ως 21^{ος} ξεχωριστός χαρακτήρας, καθώς και οι τυχόν στήλες που αποτελούνται από όμοιες πρωτεΐνες αγνοούνται κατά τον υπολογισμό των βαρών. Αναφερόμενοι στην παρατηρούμενη συχνότητα εμφανίσεως πρωτεϊνών σε μια στήλη, θα εννοείται η ζυγισμένη συχνότητά τους παρά η παρατηρούμενη.

Κατά την κατασκευή σκορ μήτρας, όχι μόνο η παρατηρούμενη συχνότητα πρωτεϊνών μιας στήλης είναι σημαντική, αλλά επίσης και ο πραγματικός αριθμός των ανεξάρτητων παρατηρήσεων από τις οποίες αποτελείται: μία στήλη που αποτελείται μόνο από βαλίνη και μια μόνο ισολευκίνη φέρει διαφορετικές πληροφορίες από μια που αποτελείται από πέντε ανεξάρτητες περιπτώσεις πρωτεϊνών η κάθε μια. Κατά συνέπεια, θα πρέπει να εκτιμηθεί ο σχετικός αριθμός N_C από ανεξάρτητες παρατηρήσεις που αποτελείται από τη πολλαπλή στοίχιση M_C. Μια απλή μέτρηση του αριθμού των ακολουθιών στο M_C είναι ένα φτωχό μέτρο, για 10 ταυτόσημες ακολουθίες συνεπάγεται λιγότερες ανεξάρτητες παρατηρήσεις από ό, τι 10 αποκλίνουσες ακολουθίες. Προτείνεται λοιπόν, ως μια απλή πρώτη εκτίμηση για το N_C ο μέσος αριθμός των διαφορετικών τύπων πρωτεϊνών, συμπεριλαμβάνοντας και τους χαρακτήρες διακένου, που παρατηρείται στις διάφορες στήλες του M_C. Αυτή η εκτίμηση δεν είναι σαφώς το ιδανικό, αφού γίνεται κορεσμός σε 21 οπότε δεν έχει σημασία πόσες ανεξάρτητες ακολουθίες που περιέχονται στο M_C. Ωστόσο, για τα δεδομένα που είναι πιθανό να αντιμετωπιστούν, το N_C είναι συνήθως πολύ μικρότερο από το 21, και ως εκ τούτου, ίσως μια αρκετά καλή προσέγγιση για τους σκοπούς του αλγορίθμου. Όπως θα φανεί, δεν είναι η απόλυτη τιμή του N_C που είναι σημαντικό, αλλά μάλλον σχετική τιμή του από μία στήλη στην άλλη. Το N_C είναι ουσιαστικά το ίδιο μέτρο της μεταβλητότητας ευθυγράμμισης που προτείνεται από τον Henikoff και Henikoff [51] για χρήση με ένα διαφορετικό τρόπο.

3.3 Πολυπαράλληλη υλοποίηση της μεθόδου δύο εντοπισμών και της επέκτασης χωρίς διάκενα

Από τη στιγμή που ο αλγόριθμος BLAST με διάκενα αποτελείται από μια σειρά ανεξάρτητων βημάτων, δύο συναρτήσεις πυρήνα της GPU προτείνονται ώστε να καλυφθεί ένα μεγάλο μέρος του αλγορίθμου: η μέθοδος των δύο εντοπισμών και η επέκταση χωρίς διάκενα. Για το

σκοπό αυτό πρώτα διαβάζονται οι ακολουθίες της βάσης δεδομένων ανά μπλοκ ίδιου αναγνωριστικού blockIdx. Η γεωμετρία της πολυπαράλληλης επεξεργασίας μπορεί να γίνει με μονοδιάστατο blockIdx και threadIdx per block. Τα threads του κάθε μπλοκ ψάχνουν την ομοιότητα μεταξύ της υποκείμενης ακολουθίας της βάσης δεδομένων και του lookup table με μια διαδικασία που περιγράφεται παρακάτω. Καθώς η διαδικασία του εντοπισμού γίνεται ανά νήμα (thread) του μπλοκ της GPU δεν είναι ανάγκη να γίνει συγχρονισμός των threads. Τότε οι επαναλήψεις των threads μπορεί να οριστούν βάσει της γεωμετρίας και εν γένει των ικανοτήτων της GPU και του μεγέθους της βάσης δεδομένων. Παρόλα αυτά, είναι γενικά καλή ιδέα να οριστεί ένας μικρός αριθμός threads ανά block ο οποίος θα μειώνει γενικά το χρόνο που κάποιος multiprocessor θα μένει αδρανής [5]. Το σχήμα Εικ 3 δείχνει μια απλοποιημένη αρχιτεκτονική του πρώτου πυρήνα (kernel). Κάθε τριπλέτα ή κωδικόνιο πρωτεϊνών μορφοποιούνται σαν λέξη που με τη σειρά της κωδικοποιείται σε τιμή 15 bit δείκτη που χρησιμοποιείται στην άμεση ανίχνευση από το thread του πρώτου εντοπισμού. Κάθε thread εκτελεί έναν εντοπισμό ανά υποκείμενη ακολουθία της βάσης δεδομένων και της ακολουθίας ερωτήματος. Γι αυτό το λόγο γίνεται ανάθεση μνήμης στην GPU και του lookup table αλλά και των τμημάτων της βάσεως δεδομένων. Εδώ πρέπει να επισημανθεί ότι οι αναζητήσεις του πρώτου εντοπισμού γίνονται παράλληλα, μέχρι την εξάντληση του τμήματος της βάσης δεδομένων. Κατόπιν η ανίχνευση του δεύτερου εντοπισμού ψάχνει για ομοιότητες με δεδομένη απόσταση στην κύρια διαγώνιο του πρώτου εντοπισμού. Αυτό το βήμα δεν γίνεται παράλληλα, αλλά γίνεται αναζήτηση στο ίδιο block της αρχιτεκτονικής της GPU. Γι αυτό το λόγο οι τιμές εισόδου και το ευριστικό στάδιο επηρεάζουν τη απόδοση του προγράμματος, μιας και είναι δυνατό να γίνεται αναζήτηση σε ένα μόνο thread και τα υπόλοιπα threads και blocks να είναι αδρανή. Σε περίπτωση που γίνει ανίχνευση δύο εντοπισμών τότε καλείται το δεύτερο kernel για την επέκταση χωρίς κενά, που θα δώσει τελικά το HSP των ακολουθιών. Εδώ πάλι δεν γίνεται παράλληλα η επέκταση χωρίς κενά, γεγονός που μπορεί να επιβραδύνει την απόδοση του προγράμματος



Εικ 3 Αναπαράσταση πολυπαράλληλης επεξεργασίας για την μέθοδο πρώτου εντοπισμού

3.4 Μέθοδος του πρώτου εντοπισμού και η δομή lookup table

Το lookup table θα πρέπει να περιέχει όλες τις πιθανές λέξεις, ο αριθμός των οποίων θα εξαρτάται από το μήκος της λέξης w και τον αριθμό γραμμάτων της αλφαβήτου των πρωτεϊνών. Από την στιγμή που υπάρχουν 25 συνεχή γράμματα από το A μέχρι το στον πίνακα BLOSUM62 που χρησιμοποιείται με πέντε μη χρησιμοποιούμενα γράμματα (B,J,O,U,X) κάθε λέξη θα μπορεί να κωδικοποιηθεί σε 5 bit λέξη-αριθμό. Για w=3 η διεύθυνση του lookup table θα είναι από 0 έως 25368 [5] με τις μη χρησιμοποιούμενες λέξεις να μην αντιστοιχούν σε διεύθυνση της δομής του lookup table. Οι λέξεις που έχουν σχηματισθεί από την ακολουθία ερωτήματος, συγκρίνονται διαδοχικά μια προς μια με τις διευθύνσεις του lookup table. Τα τρία γράμματα των λέξεων μπορούν να δώσουν μια μοναδική διεύθυνση του πίνακα lookup αν γίνουν οι εξής πράξεις address>>10, (address>>5)&31, address&31 αντίστοιχα. Οι τελεστές >> και & είναι οι bitwise τελεστές της δεξιάς ολίσθησης και του λογικού bitwise AND αντίστοιχα. Εφόσον βρεθεί μια αντιστοιχία με λέξη με σκορ μεγαλύτερο του κατωφλίου, η διεύθυνση του πίνακα επισημαίνεται και δημιουργείται έτσι η δομή του πίνακα lookup. Αφού γίνει αυτό, οι λέξεις από την υποκείμενη ακολουθία της βάσης δεδομένων, διαβάζονται και μετασχηματίζονται σε λέξεις των 15 bit που ονομάζονται δείκτες (offsets) του πίνακα lookup. Αυτοί οι δείκτες χρησιμοποιούνται στο να βρεθεί η καταχώρηση στον πίνακα lookup δηλαδή αν υπάρχει εγγραφή και η θέση του στην ακολουθία ερωτήματος. Για παράδειγμα έστω η λέξη DCT από μια εγγραφή της βάσης δεδομένων. Η τιμή του δείκτη από τον πίνακα lookup υπολογίζεται ως εξής: (D – A)<<10 | (C-A)<<5 | (T - A) . Αν υποτεθεί ότι το επόμενο γράμμα είναι το M ή επόμενη λέξη θα είναι το CTM. Τότε κρατείται στη μνήμη το αποτέλεσμα του CT και γίνεται bitwise left shifting κατά 5. Το αποτέλεσμα γίνεται ORed με την τιμή του (M - A), ώστε να εξαχθεί το offset της λέξης CTM. Εφόσον βρεθεί εντοπισμός, τότε κρατείται η τιμή της θέσης της υποκείμενης ακολουθίας (X - index) και η θέση της ακολουθίας ερωτήματος (Y index) και αυτά περνιούνται στο επόμενο στάδιο του δεύτερου εντοπισμού.

Κεφάλαιο 4ο

4 Περιγραφή του πηγαίου κώδικα

Ο πηγαίος κώδικας που περιγράφεται σ' αυτό το κεφάλαιο πρόκειται για μια τροποποίηση του κώδικα που βρίσκεται στο site [52] και που περιγράφεται στη [53]. Ο κώδικας χρησιμοποιεί το μηχανισμό πεπερασμένων αυτομάτων για την εύρεση στοιχίσεων, ενώ κατά τ' άλλα είναι παρόμοιος με τον αλγόριθμο BLAST 2.0, στη χρήση θέσης-συγκεκριμένων πινάκων και μέθοδο δύο εντοπισμών, με τη διαφορά ότι μεγάλος μέρος του αλγορίθμου γίνεται πολυπαράλληλα. Ο κώδικας περιγράφεται παρακάτω.

Η εκτέλεση του προγράμματος ξεκινά από την main(int4 argc, char* argv) και το πρόγραμμα δέχεται ορίσματα όπως αυτά φαίνονται παρακάτω:

```
  -d  Database
    default =
  -i  Query File
    default =
  -A  Multiple Hits window size
    default = 40
  -f  Threshold for extending hits
    default = 13
  -e  Expectation value (E)
    default = 10.0
  -m  alignment view options:
    0 = pairwise
    default = 0
  -y  Dropoff (X) for blast extensions in bits
    default = 7.0
  -P  0 for multiple hit
    default = 0
  -F  Filter query sequence using DUST/SEG (1=True or 0=False)
    default = 1
  -G  Cost to open a gap (Matrix dependant)
    default = 11
  -E  Cost to extend a gap (Matrix dependant)
    default = 1
  -O  Cost to open a gap (Semi-gapped alignment, Matrix dependant)
    default = 7
  -X  X dropoff value for gapped alignment (in bits)
    default = 15.0
  -N  Number of bits to trigger gapping
    default = 22.0
  -Z  X dropoff value for final gapped alignment (in bits)
    default = 25.0
  -M  Matrix
    default = BLOSUM62
  -v  Number of database sequences to show one-line descriptions for (V)
    default = 500
  -b  Number of database sequences to show alignments for (B)
    default = 250
```

```
  -W  Word size
    default = 3
  -z  Effective length of the database (use zero for the real size)
    default = 0
  -q  Penalty for a mismatch
    default = -3
  -a  Display alignments for all members of a cluster  (1=True or 0=False)
    default = 1
  -g  0,1,2,...,n - Specify any particular device to be used
    default = All suitable devices are used
```

Αυτοί οι παράμετροι διαβάζονται με την συνάρτηση `parameters_processArguments(argc, argv)` η οποία αποδίδει σε μεταβλητές τα ορίσματα του προγράμματος. Εδώ μπορούμε να πούμε ότι βασικά ορίσματα πρέπει να είναι η ακολουθία ερωτήματος (που πρέπει να είναι σε μορφή ncbi), η βάση δεδομένων που θα πρέπει πρώτα να γίνει σε μορφή FASTA.

4.1 Η μορφοποίηση ncbi

Η μορφή των δεδομένων (ncbi) που θα δοθεί σαν είσοδος στο πρόγραμμα είναι μια μορφή εγγραφής δεδομένων ακολουθιών. Συνοπτικά, αποτελείται από δύο τμήματα, από την περιγραφή της ακολουθίας και την ακολουθία. Τα δεδομένα πρέπει να ξεκινούν με το σύμβολο > ακολουθούμενο από ένα αναγνωριστικό και κατόπιν την περιγραφή. Η περιγραφή είναι σε ελεύθερο κείμενο χωρίς τον χαρακτήρα αλλαγής γραμμής. Στο πεδίο της περιγραφής μπορούν να υπάρξουν παραπάνω από μια περιγραφές χωριζόμενες από το χαρακτήρα ^A. Οι περιγραφές περιέχουν πεδία με αναγνωριστικά χωρισμένα με το χαρακτήρα |. Παράδειγμα τέτοιου αναγνωριστικού μπορεί να είναι το gi που υποδηλώνει ότι ακολουθεί πεδίο με ακέραιους. Στη συνέχεια ακολουθεί η ακολουθία που αποτελείται από χαρακτήρες που συμβολίζουν πρωτεΐνες και που βρίσκονται παραθεμένοι 80 ανά γραμμή περισσότερα αναφέρονται στο [7]. Ακολουθεί για παράδειγμα μια ακολουθία από τη βάση δεδομένων nr.

```
lou@thymus:~/Coding/Metapt_ergasia/BLAST/CUDA_BLASTP$ cat query
>gi|66816243|ref|XP_642131.1|  hypothetical  protein  DDB_G0277827
[Dictyostelium discoideum AX4]^Agi|1705556|sp|P54670.1|CAF1_DICDI RecName:
Full=Calfumirin-1;  Short=CAF-1^Agi|793761|dbj|BAA06266.1|  calfumirin-1
[Dictyostelium  discoideum]^Agi|60470106|gb|EAL68086.1|  hypothetical
protein DDB_G0277827 [Dictyostelium discoideum AX4]
MASTQNIVEEVQKMLDTYDTNKDGEITKAEAVEYFKGKKAFNPERSAIYLFQVYDKDNDGKITIKELAGDIDFD
KALKEY
KEKQAKSKQQEAEVEEDIEAFILRHNKDDNTDITKDELIQGFKETGAKDPEKSANFILTEMDTNKDGTITVKEL
RVYYQK
VQKLLNPDQ
```

4.2 Αρχεία του προγράμματος BLASTP και η μορφοποίηση τη βάσης δεδομένων

Οι βάσεις δεδομένων που περιέχουν τέτοιες ακολουθίες μπορεί κάποιος να τις κατεβάσει από το site του ncbi και κατόπιν να τις επεξεργαστεί με την εντολή ./CUDA-formatdb που γίνεται

compile μαζί με το εκτελέσιμο CUDA-BLASTP. Για παράδειγμα η έξοδος του προγράμματος CUDA-formatdb είναι η κάτωθι:

```
lou@thymus:~/Coding/Metapt_ergasia/BLAST/CUDA_BLASTP/CUDA-formatdb /tmp/nr
PROTEIN database detected.
Formatting database...
done.
19074 sequences processed.
0 wildcards encoded.
1 volume(s) created.
Longest/shortest sequence was 5017/26 letters
```

Η εκτέλεση του προγράμματος CUDA-formatdb παράγει τα μορφοποιημένα αρχεία:

```
-rw-r----- 1 lou        users      230400053 Jul  1 11:16 nr
-rw-r----- 1 lou        users         110567 Jul  1 11:16 nr.data
-rw-r----- 1 lou        users      224140291 Jul  1 11:16 nr.descriptions
-rw-r----- 1 lou        users       46675050 Jul  1 11:15 nr.gz
-rw-r----- 1 lou        users              4 Jul  1 11:16 nr.info
-rw-r----- 1 lou        users         209814 Jul  1 11:16 nr.offsetinfo
-rw-r----- 1 lou        users        6052271 Jul  1 11:16 nr.sequences
-rw-r----- 1 lou        users        6014121 Jul  1 11:16 nr.texture1
-rw-r----- 1 lou        users              5 Jul  1 11:16 nr.thread0info
-rw-r----- 1 lou        users              4 Jul  1 11:16 nr.thread2info
-rw-r----- 1 lou        users              4 Jul  1 11:16 nr.thread3info
-rw-r----- 1 lou        users              4 Jul  1 11:16 nr.thread4info
```

Τα αρχεία περιέχουν τα εξής:

Το αρχεία με την κατάληξη .threadinfo περιέχουν πληροφορίες για τις ενεργές κάρτες γραφικών και κατά συνέπεια πολυπαράλληλης επεξεργασίας, τα αρχεία με κατάληξη .texture περιέχουν συγκεντρωμένες όλες τις ακολουθίες ανά μπλοκ μνήμης της κάρτας γραφικών και συνήθως έχουν μέγεθος έως 256 MB. Τα αρχεία με κατάληξη .sequences περιέχουν συγκεντρωμένα όλες τις ακολουθίες, χωρίς κενά όπως και τα .textures, αλλά είναι πολύ μεγαλύτερου μεγέθους, περίπου 2 GB. Τα offsets των ακολουθιών, δηλαδή σε ποια θέση του αρχείου αρχίζει η εκάστοτε ακολουθία, βρίσκονται στο αρχείο με κατάληξη .offsetinfo. Τα υπόλοιπα αρχεία αφορούν τις περιγραφές των ακολουθιών της βάσης δεδομένων μαζί με τα offsets τους σε μορφή δεκαδικών αριθμών.

Επειδή τα δεδομένα αυτά αφορούν την βάση δεδομένων και την μορφοποίησή τους δεν θα αναπτυχθούν λεπτομερώς. Περισσότερα στοιχεία μπορεί κανείς να βρει στο αρχείο readme που βρίσκεται μαζί με τον πηγαίο κώδικα και στα [7], [52] και [53]

4.3 Ανάγνωση αρχείου ακολουθίας ερωτήματος

Ο κώδικας συνεχίζει με τη ανάγνωση της (ncbi formatted) ακολουθίας ερωτήματος. Αυτό γίνεται σταδιακά, πρώτα ανοίγοντας το αρχείο με την `readFasta_open(parameters queryFile)` και έπειτα με την `readFasta_readSequence()`. Η διαδικασία της ανάγνωσης ορίζει γενικές μεταβλητές στις οποίες αναθέτονται οι περιγραφή της ακολουθίας και η ακολουθία χωρίς τους χαρακτήρες αλλαγής γραμμής. Οι χαρακτήρες της περιγραφής και της

ακολουθίας τότε αντιγράφονται στις global μεταβλητές `queryDescription` και globalQuery οι οποίες είναι τύπου πίνακα χαρακτήρων (char*) με τελευταίο στοιχείο το \0. Στη συνέχεια ανιχνεύεται η ακολουθία είναι ακολουθία συμβόλων πρωτεϊνών ή νουκλεϊκών οξέων. Η αναγνώριση αυτή γίνεται με την `encoding_determineAlphabetType(query, strlen(query))`. Η αναγνώριση επιτυγχάνεται αν τα σύμβολα είναι λιγότερα από το 90% A, G, C, T. Σε αυτή την περίπτωση υπάρχει κωδικοποίηση πρωτεϊνών, αλλιώς κωδικοποίηση νουκλεϊκών οξέων. Εδώ πρέπει να επισημανθεί, ότι ενδιαφέρει μόνο η ακολουθία πρωτεϊνών οπότε το πρόγραμμα δοκιμάζεται μόνο με χαρακτήρες και είσοδο πρωτεϊνών. Σε περίπτωση που τα δεδομένα δεν είναι επαρκή ή δεν μπορούν να αναγνωριστούν οι χαρακτήρες εισόδου τότε εμφανίζεται μήνυμα λάθους και γίνεται έξοδος του προγράμματος.

4.4 Αρχικοποίηση μεταβλητών και δομή του lookup table

Η αρχικοποίηση μεταβλητών ξεκινάει άπαξ και γίνει η αναγνώριση της ακολουθίας ερωτήματος. Στη `seg_segSequence(query)` γίνεται αρχικοποίηση ενός struct τύπου sequence, στη `parameters_loadDefaults(queryAlphabetType);` γίνεται αρχικοποίηση των παραμέτρων που δεν έχουν δοθεί σαν ορίσματα στη main, βάσει του τύπου αλφάβητου των δεδομένων. Οι default τιμές δίνονται όπως παραπάνω ή όπως φαίνονται στο αρχείο readme. Κατόπιν τυπώνεται η περιγραφή του query όπως αυτή έχει δοθεί, με την printf(). Η αρχικοποίηση του lookup table γίνεται με τη δημιουργία των μεταβλητών struct BlastAaLookupTable* lookupt, struct LookupTableOptions* opt, struct BLAST_SequenceBlk* sequence, struct BlastSeqLoc* blstlc και την ανάθεση μνήμης σ' αυτές. Οι τύποι για το lookup table όπως χρησιμοποιούνται και πάρθηκαν από το BLAST πρόγραμμα του ncbi ορίζονται ως εξής:

```
typedef struct LookupTableOptions {
  double threshold; /**< Score threshold for putting words in a lookup table
                (fractional values are allowed, and could be
                important if there is scaling involved) */
  ELookupTableType lut_type; /**< What kind of lookup table to construct? */
  Int4 word_size; /**< Determines the size of the lookup table */
  Int4 mb_template_length; /**< Length of the discontiguous words */
  Int4 mb_template_type; /**< Type of a discontiguous word template */
  char* phi_pattern;  /**< PHI-BLAST pattern */
  EBlastProgramType program_number; /**< indicates blastn, blastp, etc. */
}
```

και για το lookup table:

```
typedef struct BlastAaLookupTable {
   Int4 threshold;       /**< the score threshold for neighboring words */
   Int4 mask;            /**< part of index to mask off, that is, top
                         (wordsize*charsize) bits should be discarded. */
   Int4 charsize;        /**< number of bits for a base/residue */
   Int4 word_length;     /**< Length in letters of the full word match
                         required to trigger extension */
   Int4 lut_word_length; /**< Length in bases of a word indexed by the
                         lookup table */
   Int4 alphabet_size; /**< number of letters in the alphabet */
   Int4 backbone_size;   /**< number of cells in the backbone */
   Int4 longest_chain;   /**< length of the longest chain on the backbone */
```

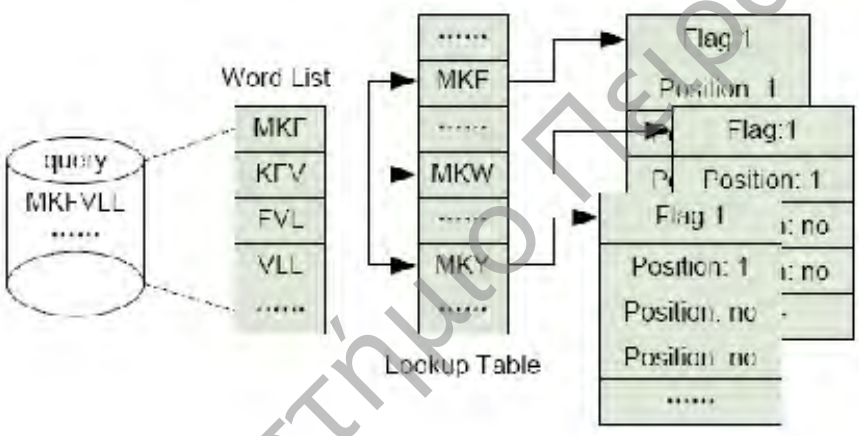
```
    Int4 ** thin_backbone; /**< the "thin" backbone. for each index cell,
                           maintain a pointer to a dynamically-allocated
                           chain of hits. */
    EBoneType bone_type;   /**< type of bone used:
                           0:  normal backbone (using Int4)
                           1:  small backbone  (using Uint2)
                           will be determined in Finalize call */
    void * thick_backbone; /**< may point to BackboneCell, SmallboneCell,
                           or TinyboneCell.
                           the "thick" backbone. after
                           queries are indexed, compact the
                           backbone to put at most
                           AA_HITS_PER_CELL hits on the
                           backbone, otherwise point to
                           some overflow storage  */
    void * overflow;       /**< may point to Int4 or Uint2,
                           the overflow array for the compacted
                           lookup table */
    Int4  overflow_size;   /**< Number of elements in the overflow array */
    PV_ARRAY_TYPE *pv;     /**< Presence vector bitfield; bit positions that
                           are set indicate that the corresponding thick
                           backbone cell contains hits */
    Boolean use_pssm;      /**< if TRUE, lookup table construction will assume
                           that the underlying score matrix is position-
                           specific */
    void *scansub_callback;/**< function for scanning subject sequences */

    Int4 neighbor_matches; /**< the number of neighboring words found while
                           indexing the queries, used for informational/
                           debugging purposes */
    Int4 exact_matches;    /**< the number of exact matches found while
                           indexing the queries, used for informational/
                           debugging purposes */
}
```

Οι βασικές μεταβλητές που πρέπει να οριστούν ώστε να δημιουργηθεί σωστά το lookup table του ncbi είναι οι opt->lut_type , opt->word_size, opt->threshold και opt->program_number. Με την ανάθεση αυτών των μεταβλητών γίνεται η αρχικοποίηση του lookup table ως εξής: int4 result1=BlastAaLookupTableNew(opt,&lookupt); . ως ορίσματα περνιούνται τα options και το lookup table που πρόκειται να αρχικοποιηθεί. Στην προαναφερθείσα συνάρτηση που βρίσκεται στο αρχείο blast_aalookup.c γίνεται ανάθεση των μεταβλητών threshold = 13 word_length=3 backbone_size=28540, alphabet_size=28, mask=11111111111111111_2 και ανάθεση μνήμης στο thin_backbone (της δομής που περιέχει το lookup table). Η δημιουργία του lookup table γίνεται αργότερα με την **void BlastAaLookupIndexQuery**(BlastAaLookupTable * lookup, Int2 ** matrix, BLAST_SequenceBlk * query, BlastSeqLoc * location,Int4 query_bias). Τα ορίσματα της συνάρτησης είναι η αρχικοποιημένη δομή του lookup table, την δομή του query, την δομή BlastSeqLoc και η τιμή offset query bias. Η δομή BlastSeqLoc ουσιαστικά πρόκειται για δομή που χρησιμοποιείται αν υπάρχουν παραπάνω από ένα queries στην αναζήτηση και σ' αυτή την περίπτωση περιέχει τα offsets των θέσεων των ακολουθιών αναζήτησης. Παρακάτω περιγράφεται η διαδικασία δημιουργίας του lookup table.

4.4.1 Δημιουργία του lookup table

Με το που καλείται η `BlastAaLookupIndexQuery` γίνεται κλήση της `s_AddNeighboringWords` η οποία πρώτα καλεί την `BlastLookupIndexQueryExactMatches` για την δημιουργία μιας δομής exact_backbone με καταχωρήσεις τις ακριβείς λέξεις που εξάγονται από το query. Η καταχώρηση γίνεται ανάλογα με το index της θέσης που καταλαμβάνει η λέξη. Στη θέση αυτή του lookup table εισέρχεται δείκτης που δείχνει σε μια δομή στήλης ακεραίων που περιέχει στη θέση 0 το μήκος της στήλης και στη θέση 1 οι εντοπισμοί που έχουν βρεθεί. Η δομή αυτή ονομάζεται chain και δημιουργείται ως εξής. Αν δεν υπάρχει καταχώρηση δηλαδή στη θέση του δείκτη στης θέσης υπάρχει το NULL καταχωρείται χώρος για δύο στοιχεία των τεσσάρων bytes. Στη θέση 0 ορίζεται το μήκος σε bytes της καταχώρησης στη θέση 1 ο αριθμός των εντοπισμών και στη θέση 2 το offset της θέσης που βρίσκεται η λέξη στην ακολουθία ερωτήματος. Αν ο δείκτης έχει ήδη καταχώρηση τότε κατανέμεται περισσότερος χώρος και γίνεται καταχώρηση του offset της λέξης στο στοιχείο 4 της δομής του chain. Αυτή η δομή μπορεί να φανεί στο παρακάτω σχήμα:



Εικ 4 Η δομή του lookup table

Η διαδικασία συνεχίζει με την δημιουργία του lookup table. Οι ακριβείς καταχωρήσεις ενσωματώνονται στο lookup table με την `s_AddWordHits` που καλείται παρακάτω στην `s_AddNeighboringWords`. Σ' αυτή τη συνάρτηση υπολογίζεται το σκορ για κάθε λέξη που βρέθηκε από την `BlastLookupIndexQueryExactMatches`. Αν το σκορ της λέξης είναι κάτω από το κατώφλι τότε η λέξη ενσωματώνεται αυτολεξεί και η προσθήκη γίνεται με την `BlastLookupAddWordHit`. Ο κώδικας προχωράει και με την προσθήκη γειτονικών λέξεων με την `s_AddWordHitsCore` αναδρομικά σύμφωνα με την παράγραφο [...]. Σ' αυτή τη συνάρτηση χρησιμοποιείται μια δομή προφίλ της λέξης info και το σκορ που έχει και σχηματίζονται όλες οι γειτονικές της λέξεις ανάλογα με το αλφάβητο που χρησιμοποιείται. Η προσθήκη γίνεται και πάλι με την `BlastLookupAddWordHit`, αν το σκορ της λέξης είναι μεγαλύτερο από το κατώφλι.

4.5 Δημιουργία της θέσης-ειδικού πίνακα (PSS matrix)

Ο κώδικας συνεχίζει με την δημιουργία του θέσης-ειδικού πίνακα σύμφωνα με την PSSMatrix_create. Ο πίνακας θα έχει στήλες ίσες με το μήκος του ερωτήματος και θα έχει γραμμές ίσες με το αλφάβητο που χρησιμοποιείται. Η δομή του PSSMatrix είναι η ακόλουθη:

```
struct PSSMatrix
{
   int4 length;
   int4 strandLength;
         int4 highestValue;
         int4 lowestValue;
         int2** matrix;
         unsigned char* queryCodes;
   unsigned char* bestMatchCodes;
   unsigned char* bytePackedCodes;
         unsigned char* xorCodes;
};
```

Ο πίνακας θα έχει αριθμό στηλών όσο το μήκος του ερωτήματος και αριθμό γραμμών ίσο με 25. Αρχικά μετατρέπονται τυχόν μικρά γράμματα σε κεφαλαία και στη συνέχεια καταχωρείται μνήμη για τον πίνακα. Η πρώτη στήλη θα είναι στήλη δεικτών ενώ καταχωρείται μνήμη για την αποθήκευση της κωδικοποιημένης ακολουθίας ερωτήματος και αποθηκεύονται μεταβλητές όπως το μέγιστο και το ελάχιστο σκορ. Κατόπιν για κάθε χαρακτήρα της ακολουθίας ερωτήματος αποθηκεύεται ο κώδικας που αντιστοιχεί στο μεγαλύτερο σκορ στη συγκεκριμένη θέση. Τυπικά πρόκειται για τον ίδιο χαρακτήρα με το γράμμα της ακολουθίας στη συγκεκριμένη θέση. Στο τέλος αφού βρεθούν οι καλύτεροι «κώδικες» για κάθε χαρακτήρα της ακολουθίας επιστρέφεται η δομή του PSSMatrix.

4.6 Η κύρια συνάρτηση blastp

Η συνάρτηση ξεκινάει με το να ελεγχθεί αν έχει ανιχνευτεί σωστά το αλφάβητο και το ποια μορφή ερωτήματος και βάσης δεδομένων έχει οριστεί για το πρόγραμμα. Στη συνέχεια γίνεται η αρχικοποίηση των στατιστικών παραμέτρων που απαιτούνται από τον αλγόριθμο. Η αρχικοποίηση αυτή γίνεται με την συνάρτηση `statistics_initialize(PSSMatrix, readdb_numberOfLetters, readdb_numberOfSequences);` Σ΄ αυτή την συνάρτηση υπολογίζονται οι παράμετροι Karlin-Altschul λ, H, K για στοίχιση με διάκενα και χωρίς διάκενα και στη συνέχεια οι στατιστικές εκτιμήσεις κατωφλίων dropoff για την στοίχιση με διάκενα. Στην επιστροφή στη blastp γίνονται περαιτέρω εκτιμήσεις και διορθώσεις του κατωφλίου για στοίχιση με διάκενα, δηλαδή το πόσο μπορεί να μειωθεί η στοίχιση αριστερά και δεξιά της στοιχισμένης λέξης ώστε να θεωρηθεί το ενδιάμεσο των λέξεων ως διάκενο ή ως επεκταμένη στοιχισμένη λέξη. Στη συνέχεια διαβάζεται ο αριθμός των textures που έχουν σχηματισθεί κατά την χρησιμοποίηση της formatdb. Με είσοδο το αριθμό των textures και του πλάνου χρησιμοποιήσεως 4 πυρήνων GPU καλείται η GPU_blast. Σ’ αυτή την συνάρτηση υπολογίζονται οι δυνατότητες και ο αριθμός των ενεργών καρτών που μπορούν να χρησιμοποιηθούν παράλληλα. Αν δεν υπάρχει ικανός αριθμός nvidia GPUs τότε γίνεται έξοδος από το πρόγραμμα. Σε κάθε άλλη περίπτωση μπορεί να χρησιμοποιηθούν έως 4 GPUs παράλληλα καλώντας την `static CUT_THREADPROC CUDA_blast(TGPUplan *plan)` για κάθε plan που έχει δημιουργηθεί ανά GPU.

4.7 Η συνάρτηση CUDA_blast

Η συνάρτηση CUDA_blast πρόκειται για την συνάρτηση που προετοιμάζει τα δεδομένα για την κάρτα γραφικών και καλεί τον πυρήνα του πολυπαράλληλου υπολογισμού. Τα δεδομένα για τα οποία ανατίθεται μνήμη στη κάρτα γραφικών είναι d_readdb_sequenceData το d_PSSMatrix και το d_output που συνοπτικά αναπαριστούν την μνήμη που απαιτείται για την ανάγνωση του τμήματος της βάσης δεδομένων το PSSMatrix και η έξοδος με τα offsets που φέρουν για τις ενδεχόμενες στοιχίσεις. Επίσης σ' αυτό κομμάτι κώδικα ανατίθενται οι τιμές για τις παραμέτρους εκτέλεσης των πολυπαράλληλων υπολογισμών ήτοι ο αριθμός των blocks και ο αριθμός των threads ανά block. Οι παράμετροι αυτοί παίρνουν τις τιμές 96 και 64 αντίστοιχα. Στην πρώτη έκδοση του κώδικα περνιούνται και σαν παράμετροι στην μνήμη της GPU το ντετερμινιστικό πεπερασμένο αυτόματο για τις επόμενες «τιμές» των λέξεων και τα επόμενα groups των λέξεων. Στη συνέχεια του κώδικα ορίζονται οι επαναλήψεις που πρόκειται να γίνουν ανάλογα με τα textures που έχουν προκύψει από το CUDA_formatdb. Έτσι ανάλογα με το `textureSize` που είναι μεταβλητή που ορίζεται από τις δυνατότητες της κάρτας γραφικών μεταφέρονται διαδοχικά από τον υπολογιστή στην κάρτα γραφικών τα τμήματα της βάσης δεδομένων ως μεταβλητή `d_texDB`. Συνολικά στον πυρήνα περνιούνται οι εξής μεταβλητές:

CUDA_Blast_kernel<<<grid, threads>>>(d_queryLength,	το μήκος του ερωτήματος
d_readdb_sequenceData,	το ερώτημα σε μορφή δομής δεδομένων
d_shortminOffset,	το offset που βρίσκεται η ακολουθία
d_shortpitch,	το offset που βρίσκεται η ακολουθία
statistics_ungappedNominalDropoff,	το ποσό που μπορεί να μειωθεί το ταίριασμα
CPU_hitMatrix,	η περιοχή διεύθυνσης όπου υπάρχει ταίριασμα
readdb_numberOfClusters,	ο αριθμός των ακολουθιών που περιέχονται
d_output,	η δομή δεδομένων εξόδου
d_outputPitch,	το offset που βρίσκεται η στοίχιση
blast_ungappedNominalTrigger,	το σκορ για το οποίο εξακολουθεί να υπάρχει ταίριασμα
d_r1Cutoff,	ο αριθμός που μπορεί να μειωθεί το σκορ λέξης
d_roundNumber,	ο αριθμός του πρώτου cluster
d_nextRoundNumber,	ο αριθμός του επόμενου cluster
textureLoop,	ο αριθμός επανάληψης για το texture
parameters_wordSize,	το μήκος της λέξης
d_texDB);	το τμήμα της βάσης δεδομένων που περνιέται

4.8 Ο πυρήνας CUDA_BLAST_kernel

Ο κώδικας για την αρχική έκδοση φαίνεται παρακάτω:

```
__global__ void
CUDA_Blast_kernel(int d_queryLength,
                                struct CPU_sequenceData* d_readdb_sequenceData,
                                int* d_shortminOffset,
                                size_t d_shortpitch,
                                int d_statistics_ungappedNominalDropoff,
                                int d_CPU_hitMatrix,
                                unsigned int numSequences,
                                struct GPU_offsetData* d_output,
                                unsigned int d_outputPitch,
                                int d_blast_ungappedNominalTrigger,
                                int d_r1Cutoff,
                                int d_roundNumber,
                                int d_nextRoundNumber,
                                int d_textureLoop,
                                int d_wordSize,
                                unsigned char* d_texDB)
{
        unsigned int tid_total = threadIdx.x + blockIdx.x * blockDim.x;

        unsigned int sequenceCount;
        unsigned char currentWord;

        short queryOffsets,
                currentGroup,
                tempGroup,
                currentBlock,
                CUDA_ungappedExtension,
                CUDA_index = 0;
        int sequenceEnd,
                address,
                subject,
                distance,
                d_CPU_ungappedExtension_subjectEndReached,
                diagonal,
                tempIndex,
                subjectStartOffset,
                subjectEndOffset,
                subjectLength,
                subjectOffset,
                lastHit,
                count,
                PSSmatrixIndex;

        unsigned int upperBorder;

        int* row;
```

Στο παραπάνω τμήμα δηλώνονται οι τοπικές μεταβλητές τύπου register για κάθε thread υπολογισμού. Διαπιστώνεται επειδή ο αριθμός μεταβλητών είναι αρκετά μεγάλος, θα πρέπει οι δυνατότητες υπολογισμού της κάρτας να είναι μεγαλύτερες του 3.0

```
        if(d_queryLength > 3000)
        {
                row = d_shortminOffset + tid_total * d_queryLength +
d_readdb_sequenceData[tid_total].sequence;
        }
        else
        {
                if(d_textureLoop == 1)
                {
                        row = d_shortminOffset + tid_total * d_queryLength +
d_readdb_sequenceData[tid_total].sequence;
                }
                else
                {
                        row = (int*)((char*)d_shortminOffset + tid_total * d_shortpitch);
                }
        }

        PSSmatrixIndex = 32;
        d_output[tid_total * RESULTSIZE].sequenceCount = 0;

        if(d_roundNumber != d_nextRoundNumber)
        {
                upperBorder = d_nextRoundNumber * BLOCKNUM * THREADNUM;
        }
        else
        {
                upperBorder = numSequences;
        }
```

Σ' αυτό το κομμάτι ορίζεται η τοπική μεταβλητή row ο οποίος είναι ο τρέχον δείκτης για κάθε γραμμή που μπορεί να σχηματιστεί από το query ανά πολλαπλάσιο του μήκους του, με τη μεταβλητή tid_total να παίρνει τιμές από 0 – 64*96. Ουσιαστικά πρόκειται να δημιουργηθεί μια δομή τύπου πίνακα για την εύκολη εύρεση αργότερα των εντοπισμών και των αποστάσεων της διαγωνίου και των εντοπισμών μεταξύ τους

```
        for(sequenceCount = tid_total + d_roundNumber * BLOCKNUM * THREADNUM;
sequenceCount < upperBorder; sequenceCount += BLOCKNUM * THREADNUM)
        {
                subjectLength = d_readdb_sequenceData[sequenceCount].sequenceLength;
                address = subject = d_readdb_sequenceData[sequenceCount].sequence;
                CUDA_ungappedExtension = 0;

                if (subjectLength >= d_wordSize)        {
        currentGroup = 0;
                        count = d_wordSize;
```

```
                        while(count > 0)
                        {
                                if (d_texDB[address] < d_wordLookupDFA_numCodes)
                                        currentGroup = _cons_DFAnextGroups[currentGroup] +
d_texDB[address];
                else
                                        currentGroup = _cons_DFAnextGroups[currentGroup];

                address++;
                count--;
                        }

                sequenceEnd = subject + subjectLength;
                while (address < sequenceEnd)
                {
                                        tempGroup = currentGroup;
                                        currentBlock = MAXCURRENTWORD;

                                        if (d_texDB[address] < d_wordLookupDFA_numCodes)
                        {
                                        currentWord = tex2D(tex_DFAnextWords, currentBlock +
d_texDB[address], tempGroup);
                                        currentGroup = _cons_DFAnextGroups[currentGroup] +
d_texDB[address];
                        }
                        else
                        {
                                        currentWord = tex2D(tex_DFAnextWords, currentBlock,
tempGroup);
                                        currentGroup = _cons_DFAnextGroups[currentGroup];
                        }

                        if (currentWord)
                        {
                           subjectOffset = address - subject;
                                        currentBlock = (currentBlock - currentWord * 2)/2;
                                        queryOffsets = tex2D(tex_DFAnextWords_short,
currentBlock, tempGroup);

                                        if(!queryOffsets)
                                        {
```

Εδώ γίνονται parsed τα subject sequences ανά λέξεις μήκους w=3. Για κάθε λέξη γίνεται έρευνα για εντοπισμούς με το πεπερασμένο αυτόματο DFAnextWords. Η μνήμη για τα πεπερασμένα αυτόματα είναι τύπου texture που γίνεται allocated με τις εντολές `cudaMemcpyToArray` και `cudaBindTextureToArray` Η Εύρεση της επόμενης μετάβασης ή σε group ή σε word από την texture μνήμη γίνεται με την tex2D με δείκτη τη μεταβλητή τη διεύθυνση του current block και του current word. Αν βρεθεί ομοιότητα δηλαδή εντοπισμός όμοιων λέξεων ή λέξεων με υψηλό σκορ και ο εντοπισμός έχει συμβεί εντός της διαγωνίου σε απόσταση μικρότερη του κατωφλίου τότε καλείται η επέκταση του εντοπισμού χωρίς διάκενα

```
                                        tempIndex = tex2D(tex_DFAnextWords_short,
currentBlock + 1, tempGroup) * constants_max_int2 + tex2D(tex_DFAnextWords_short, currentBlock
+ 2, tempGroup);
                                        queryOffsets =
tex1Dfetch(tex_additionalQueryPositions, tempIndex);
                                        do
                                        {
                                                // Calculate the diagonal this hit is on
                                                diagonal = subjectOffset - queryOffsets;

                                                // Calculate distance since last hit
                                                lastHit = d_CPU_hitMatrix + diagonal;
                                                distance = address - row[lastHit];

                                                if (distance >= d_parameters_A)
                                                {
                                                        // Too far apart, update furthest
                                                        row[lastHit] = address;
                                                }
                                                else if (distance >= d_parameters_overlap)
                                                {
                                                        CUDA_ungappedExtension
                                = d_CUDA_ungappedExtension_extend_tex(PSSmatrixIndex +
queryOffsets * 32,
                                                address,
                                                row[lastHit],
                                                subject,
                                                d_statistics_ungappedNominalDropoff,
                                                d_CPU_ungappedExtension_subjectEndReached,
                                                d_blast_ungappedNominalTrigger,
                                                d_r1Cutoff,
                                                subjectStartOffset,
                                                subjectEndOffset,
                                                d_texDB);
                                                // Update furthest reached value for
the diagonal
                                                row[lastHit] =
d_CPU_ungappedExtension_subjectEndReached;

                                                // If extension scores high enough
to trigger gapping
                                                if (CUDA_ungappedExtension)
                                                {
                                                        CUDA_index++;
                                                        d_output[tid_total *
RESULTSIZE + CUDA_index].sequenceCount = sequenceCount;
                                                        d_output[tid_total *
RESULTSIZE + CUDA_index].subjectStartOffset = subjectStartOffset;
                                                        d_output[tid_total *
RESULTSIZE + CUDA_index].subjectEndOffset = subjectEndOffset;
                                                        d_output[tid_total *
RESULTSIZE + CUDA_index].diagonal = diagonal;
                                                        d_output[tid_total *
RESULTSIZE].sequenceCount = CUDA_index;
                                                }
```

```
                                                }
                                                queryOffsets =
tex1Dfetch(tex_additionalQueryPositions, tempIndex++);
                                        }
                                        while (queryOffsets);
                                }
                                else
                                {
                                        do
                                        {
                                                // Calculate the diagonal this hit is on
                                                diagonal = subjectOffset - queryOffsets;

                                                // Calculate distance since last hit
                                                lastHit = d_CPU_hitMatrix + diagonal;
                                                distance = address - row[lastHit];

                                                if (distance >= d_parameters_A)
                                                {
                                                        // Too far apart, update furthest
                                                        row[lastHit] = address;
                                                }
                                                else if (distance >= d_parameters_overlap)
                                                {
                                                        CUDA_ungappedExtension
                                                                =
d_CUDA_ungappedExtension_extend_tex(PSSmatrixIndex + queryOffsets * 32,
                                        address,
                                        row[lastHit],
                                        subject,
                                        d_statistics_ungappedNominalDropoff,
                                        d_CPU_ungappedExtension_subjectEndReached,
                                        d_blast_ungappedNominalTrigger,
                                        d_r1Cutoff,
                                        subjectStartOffset,
                                        subjectEndOffset,
                                        d_texDB);
                                                        // Update furthest reached value for
the diagonal
                                                        row[lastHit] =
d_CPU_ungappedExtension_subjectEndReached;

                                                        // If extension scores high enough
to trigger gapping
                                                        if (CUDA_ungappedExtension)
                                                        {
                                                                CUDA_index++;
                                                                d_output[tid_total *
RESULTSIZE + CUDA_index].sequenceCount = sequenceCount;
                                                                d_output[tid_total *
RESULTSIZE + CUDA_index].subjectStartOffset = subjectStartOffset;
                                                                d_output[tid_total *
RESULTSIZE + CUDA_index].subjectEndOffset = subjectEndOffset;
                                                                d_output[tid_total *
RESULTSIZE + CUDA_index].diagonal = diagonal;
```

```
                                                    d_output[tid_total *
RESULTSIZE].sequenceCount = CUDA_index;
                                                }
                                            }
                                            currentBlock = currentBlock + 1;
                                            queryOffsets =
tex2D(tex_DFAnextWords_short, currentBlock, tempGroup);
                                        }
                                        while (queryOffsets);
                                    }
                                }
                                address++;
                            }
                        }
}
}
```

Εάν πάλι υπάρχει κάποιος εντοπισμός και δεν υπάρχει κάποιο offset γίνονται διαδοχικά για κάθε thread, παρόμοιοι υπολογισμοί, όπως υπολογισμός διαγωνίου, υπολογισμός απόστασης υπολογισμός σκορ επέκτασης χωρίς διάκενα και εφόσον υπάρχει τέτοιο, γίνεται υπολογισμός της δομής δεδομένων εξόδου και τελικά αύξηση των μετρητών `currentBlock` και `address`.

4.9 Επιστροφή από το πυρήνα CUDA_BLAST_kernel

Εφόσον γίνει ο υπολογισμός των στοιχίσεων από τον πυρήνα η συνάρτηση επιστρέφει στη blastp όπου γίνεται αντιγραφή των αποτελεσμάτων της πολυπαράλληλης και πολυπύρηνης επεξεργασίας για κάθε αποτέλεσμα κάθε πυρήνα που έτρεξε. Εφόσον πραγματοποιηθεί αυτό το στάδιο γίνεται κλήση της runSW η οποία πρόκειται για το στάδιο ολικής στοίχισης των τοπικών αποτελεσμάτων με τον αλγόριθμο Smith Waterman. Περιληπτικά αυτή η συνάρτηση ετοιμάζει το στάδιο πολυπαράλληλης επεξεργασίας για τον αλγόριθμο Smith Waterman, πρώτα ορίζοντας ένα plan για κάθε διαθέσιμη GPU και στη συνέχεια ετοιμάζοντας τις μεταβλητές με αντιγραφή των απαραίτητων μεταβλητών από το host στα διαθέσιμα devices με μορφή shared texture arrays και τελικά το πέρασμα στη `sub_based_sw(short* output, short* startIndex, short* seqProcessed, short* segcellNumber)`. Η συνάρτηση αυτή πρόκειται για μια παράλληλη εκδοχή του αλγορίθμου Smith Waterman και περιγράφεται πολύ καλά στη [54]. Εφόσον γίνει μια ολική στοίχιση, αυτή περνιέται σαν output πίνακας που περιέχει τον αύξων αριθμό του hit από τον αλγόριθμο S-W. Μετά το πέρας της εκτέλεσης του αλγορίθμου τα αποτελέσματα αντιγράφονται για κάθε plan σε ένα resultArray και ελευθερώνονται οι μεταβλητές που έχουν εγγραφεί στη μνήμη.

4.10 Εκτύπωση αποτελεσμάτων και έξοδος

Εφόσον γίνει σωστά η οργάνωση των αποτελεσμάτων, δημιουργείται αρχείο στο οποίο εγγράφονται τα αποτελέσματα. Τα αποτελέσματα θα περιέχουν τις δύο ακολουθίες που έχουν στοιχηθεί ολικά και τους ενδιάμεσους χαρακτήρες ομοιότητας ή κενού καθώς και περιγραφές των ακολουθιών που εξήχθησαν από τη βάση δεδομένων. Επίσης δημιουργείται εκ νέου ο θέσεων-ειδικός πίνακας με τους εντοπισμούς που βρέθηκαν και τα στατιστικά στοιχεία που απαιτούνται όπως το κατώφλι για την εύρεση στοιχίσεων με διάκενα. Τελικά από τον θέσεων-

ειδικό πίνακα γίνεται η εύρεση των τελικών μορφών στοιχίσεων με τις

```
alignments_findGoodAlignments(PSSMatrix);
```

```
alignments_findFinalAlignments(PSSMatrix);
```

```
alignments_getFinalAlignmentDescriptions();
```

```
alignments_getTracebacks(PSSMatrix);
```

οι οποίες πραγματοποιούν την στοίχιση με διάκενα βάσει των στατιστικών που αναφέρθησαν. Ουσιαστικά η πρώτη από αυτές βρίσκει το αρχικό σκορ στοιχίσεων με διάκενα προκειμένου να βρεθούν κάποια ικανοποιητικό σκορ, η δεύτερη βρίσκει τις τελικές στοιχίσεις με διάκενα που είναι πάνω από το κατώφλι, η τρίτη επιστρέφει και καταχωρεί τις περιγραφές από τις ακολουθίες και η τελευταία βρίσκει τα ίχνη οπισθοδρόμησης των εντοπισμών από τις ακολουθίες που βρέθηκαν. Τελικά εκτυπώνονται οι στοιχίσεις που πρώτα έχουν υπολογιστεί, εφόσον υπάρχουν και απελευθερώνονται και οι τελευταίες καταχωρήσεις στη μνήμη από δομές που χρησιμοποιήθηκαν κατά τη διάρκεια εκτέλεσης του προγράμματος.

Κεφάλαιο 5ο

5 Συμπεράσματα

Στην παρούσα εργασία έγινε μια περιγραφή του προβλήματος της στοίχισης ακολουθιών πρωτεϊνών μέσω του αλγορίθμου BLAST. Επίσης έγινε μια επεξήγηση του κώδικα και της υπάρχουσας υλοποίησης [52] που χρησιμοποιεί την GPU για πολυπαράλληλη και συνεπώς ταχύτερη επεξεργασία του αλγορίθμου. Επειδή ο υπάρχον κώδικας χρησιμοποιεί για την εύρεση ταιριάσματος λέξεων τον μηχανισμό πεπερασμένων αυτομάτων, δεν επεκτάθηκε η περιγραφή σ' αυτό το πεδίο, αλλά έγινε μια περιγραφή για το πώς θα μπορούσε να γίνει η υλοποίηση με πίνακα ανιχνεύσεως (lookup table). Εδώ θα πρέπει να ακολουθήσει μια περιγραφή των βημάτων που συστήνονται να γίνουν, ώστε να λειτουργεί ο κώδικας πολυπαράλληλα, αλλά με χρήση lookup table αντί των πεπερασμένων αυτομάτων.

5.1 κωδικοποίηση

Καταρχάς θα πρέπει να γίνει κωδικοποίηση κάθε γράμματος του αλφάβητου για αριθμό γραμμάτων ίσο με 32. Αυτό δεν χαλάει τον αριθμό των bits που θα χρησιμοποιηθούν για κάθε λέξη, που θα είναι 5, αλλά θα γίνεται αλλαγή στο index για την εύρεση λέξεων στο lookup table. Κάνοντας αυτό το βήμα, θα υπάρχει συμβατότητα με την κωδικοποίηση του ncbi και του κώδικα που χρησιμοποιείται εδώ. Η αλλαγή θα γίνει στο αρχείο blast_aalookup.c και στην αντίστοιχη επικεφαλίδα για τις μεταβλητές BLASTAA_SIZE και αυτών που ορίζονται μέσω αυτής.

5.2 Μια προσέγγιση για την υλοποίηση του αλγορίθμου

Δεύτερο στοιχείο που θα πρέπει να δοθεί προσοχή θα είναι η μεταφορά μετά την ολοκλήρωση δημιουργίας του lookup table στην μνήμη της GPU, όπως γίνεται με τις δομές DFA: Εδώ υπάρχει μια σημαντική διαφορά, ότι η δομή του lookup table είναι εντελώς διαφορετική από τη δομή των πεπερασμένων αυτομάτων, με τις δεύτερες να έχουν την μορφή πίνακα δεικτών. Επίσης έχουν τελείως διαφορετικό μέγεθος με τις πρώτες να έχουν μέγεθος τουλάχιστο 4*28540 bytes. Σ' αυτό το μέγεθος προσθέτονται και οι καταχωρήσεις για κάθε πιθανό κωδικόνιο που σχηματίζεται συν τα γειτονικά κωδικόνια που δημιουργούνται. Για παράδειγμα σε αριθμούς αυτές οι ποσότητες είναι 167 για ακριβή ταιριάσματα και 4943 γειτονικά ταιριάσματα για ένα ερώτημα μήκους 169 αμινοξέων και κατώφλι ταιριάσματος με σκορ >= 13. Το πρόβλημα του μεγέθους συνιστά πρόβλημα για την μεταφορά της δομής του lookup table σε γρήγορη περιοχή μνήμης της GPU. Γι αυτό το λόγο συνιστάται [5] η μετατροπή του lookup table σε πίνακα με όλες τις πιθανές καταχωρήσεις. Δηλαδή για το παραπάνω παράδειγμα ένα πίνακα 5110x4x4 καταχωρήσεων ίσο δηλαδή με 40880 bytes, λίγο μικρότερο από το μέγιστο επιτρεπτό όριο της shared μνήμης της GPU, καθώς και χρήση δεύτερου πίνακα με αποθηκευμένα όλα τα offsets των καταχωρήσεων. Επιπλέον θα χρειαστεί και δεύτερη δομή

τύπου πίνακα αποθηκευμένη σε περιοχή μνήμης γρήγορης προσπέλασης, δομής που θα περιέχει τους δείκτες για τις καταχωρήσεις σε μορφή 15bit offset

Εφόσον δεν υπάρχουν πίνακες υπερχείλισης (overflow arrays) η δομή του lookup table θα έχει τη μορφή Nx4 πίνακα, οπότε μπορεί να περαστεί στην texture μνήμη της κάρτας και να χρησιμοποιηθεί ως tex2D array. Η αποθήκευση στην μνήμη της κάρτας μπορεί να γίνει με τις εντολές:

```
CUDA_SAFE_CALL(cudaMallocArray(&cu_array_lookupt,&tex_lookupt.channelDesc,
number_of_matches*4*4) ));

CUDA_SAFE_CALL(cudaMemcpyToArray(cu_array_lookupt,0,0,lookupt,
number_of_matches*8, cudaMemcpyHostToDevice));

    // set texture parameters

    tex_lookupt.addressMode[0] = cudaAddressModeClamp;

    tex_lookupt.addressMode[1] = cudaAddressModeClamp;

    tex_lookupt.filterMode = cudaFilterModePoint;

    tex_lookupt.normalized = false;        // access without normalized
texture coordinates

    // Bind the array to the texture

CUDA_SAFE_CALL(cudaBindTextureToArray(tex_lookupt,cu_array_lookupt,tex_loo
kupt.channelDesc));
```

Έχοντας πρώτα ορίσει την κατάλληλη μεταβλητή `texture<>tex_lookupt` στον πυρήνα του προγράμματος πολυπαράλληλων υπολογισμών. Όμοια διαδικασία μπορεί να γίνει και με τον πίνακα υπερχείλισης, δηλαδή τον πίνακα που περιέχει τα offsets από τα παραπάνω από δύο φορές ταιριάσματα της ίδιας λέξης στην ακολουθία ερωτήματος, καθώς και με τον πίνακα των offsets

5.3 Η προτεινόμενη υλοποίηση και η ανάθεση μνήμης

Μια άλλη λύση [5], θα ήταν η αποθήκευση στην global μνήμη ολόκληρου του lookup table μια μέθοδος που προτιμάται από το [5] λόγω της απλότητας στο χειρισμό. Σ' αυτή την περίπτωση δημιουργείται το ncbi lookup table με πίνακα offset με κατάλληλη μετατροπή του index της δομής δεδομένων του lookup table που δημιουργήθηκε από την `BlastAaLookupIndexQuery` σε μορφή 15 bit διεύθυνσης, όπως αναφέρθηκε στο κεφάλαιο 2. Αυτή η μετατροπή γίνεται με την συνάρτηση `Int4 BlastAaLookupFinalize(BlastAaLookupTable * lookup, EBoneType bone_type)`. Αυτή θα δημιουργήσει ένα πίνακα με το δυαδικό δείκτη του lookup table, για όπου υπάρχουν καταχωρίσεις ομοιότητας λέξεων. Η δομή αυτή ονομαζόμενη ως pv είναι ένας πίνακας 1D με τα indexes των καταχωρήσεων. Η μεταφορά ολόκληρου του lookup table στην global μνήμη θα γίνει με καταχώρηση στην global μνήμη, μεγέθους μνήμης αρκετό, ώστε να χωρέσει ολόκληρη η δομή, και μετά μεταφορά με την cudaMemcpy()

5.4 Ο ψευδοκώδικας για την προτεινόμενη υλοποίηση

Στη συνέχεια ο κώδικας που θα ελέγχει για καταχωρήσεις στην GPU θα μοιάζει με τον παρακάτω:

1 τρέξε παράλληλα κάθε λέξη του πίνακα της βάσης δεδομένων μέχρι το μήκος της κάθε ακολουθίας

2 αν η ακολουθία ερωτήματος είναι μικρότερη σε μήκος από κάποια σταθερά

2.1 σύγκρινε κάθε λέξη της βάσης δεδομένων που έχει μεταφερθεί στη μνήμη με την ακολουθία ερωτήματος για κάθε αύξων αριθμό του pointer της διεύθυνσης του πίνακα της βάσης δεδομένων:

2.1.1 αν υπάρχει καταχώρηση στον πίνακα pv πήγαινε στην καταχώρηση του lookup table και κράτησε την διεύθυνση σε query και db.

2.1.2 συνέχισε τη σύγκριση με άλλους δείκτες αυτή τη φορά

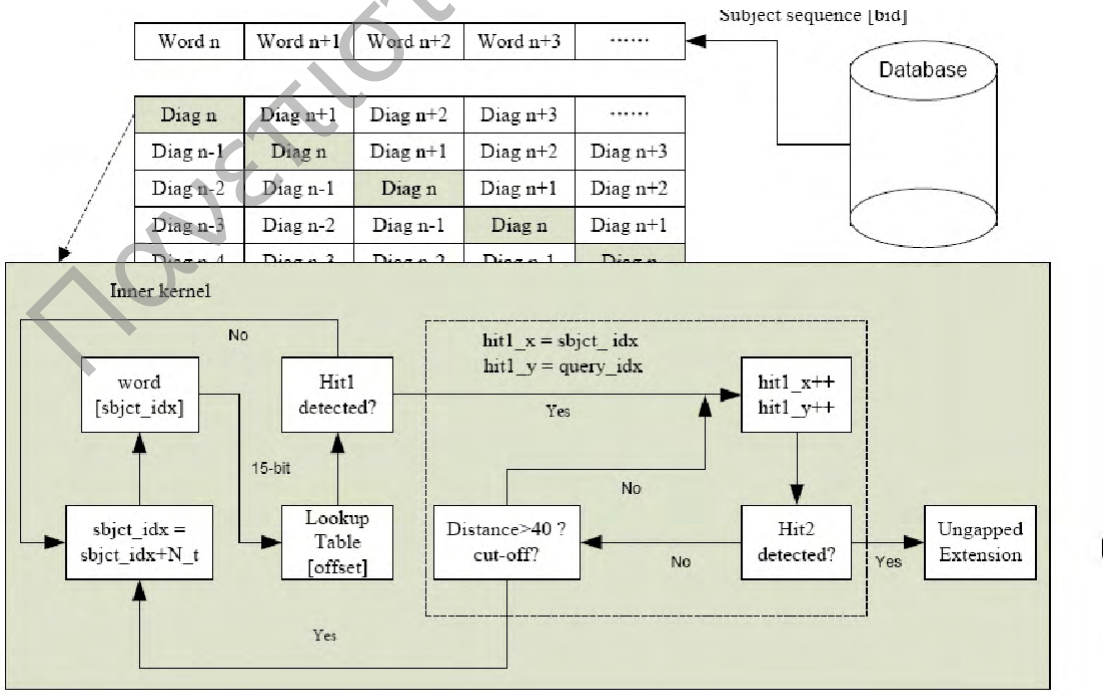
2.1.3 αν υπάρξει και δεύτερο hit

2.1.3.1 έλεγξε είναι μικρότερο από το κατώφλι απόστασης;

2.1.3.2 αν ναι τότε κάλεσε την επέκταση χωρίς διάκενα

2.1.3.3 αν έχει ξεπεραστεί το κατώφλι απόστασης τότε επέστρεψε στην προηγούμενη επανάληψη έως ότου εξαντληθεί το μήκος της ακολουθίας

Οι επαναλήψεις πραγματοποιούνται μέχρι να εξαντληθεί το μήκος της ακολουθίας ερωτήματος για όλες τις ακολουθίες της βάσης δεδομένων. Ουσιαστικά γίνεται παράλληλη αναζήτηση για μια λέξη της ακολουθίας ερωτήματος για κάθε λέξη της βάσης δεδομένων. Ο παράλληλος αλγόριθμος μπορεί να αναπαρασταθεί σύμφωνα με το παρακάτω διάγραμμα:



Τελικά γίνεται, όπως αναφέρθηκε, και η τοπική στοίχιση με τον παράλληλο αλγόριθμο Smith Waterman για όλους τους εντοπισμούς που έχουν βρεθεί, αφού εξαντληθούν οι ακολουθίες της βάσης δεδομένων.

5.5 Πλεονεκτήματα και μειονεκτήματα της προτεινόμενης υλοποίησης

1 Για μεγάλα μήκη ακολουθιών της βάσης δεδομένων, θα πρέπει να γίνονται επαναλήψεις για το δείκτη της λέξης ενώ οι υπόλοιπες αναζητήσεις σε ακολουθίες της βάσης δεδομένων θα έχουν τελειώσει. Αυτό θα έχει ως συνέπεια την καθυστέρηση και την δημιουργία μη ενεργών warps

2 Για διαφορετικά μήκη ακολουθιών θα υπάρχει γενικότερα μια καθυστέρηση για όσες αναζητήσεις τελειώνουν πιο γρήγορα από την ολοκλήρωση των επαναλήψεων. Αυτό σημαίνει μη αποδοτική εκμετάλλευση του επεξεργαστή GPU και δημιουργία μη ενεργών warps

3 Όπως αναφέρθηκε, η αποδοτικότητα σε ταχύτητα προσπέλασης στη μνήμη, θα είναι αρκετά χαμηλή, αφού θα χρησιμοποιείται εξ' ολοκλήρου η global περιοχή της μνήμης της GPU για την χρησιμοποίηση του lookup table

4 Προτέρημα της υλοποίησης είναι όπως αναμένεται, η παράλληλη αναζήτηση ομοιοτήτων λέξεων μεταξύ ακολουθιών ερωτήματος και βάσης δεδομένων, γεγονός που επισπεύδει τον αλγόριθμο blast.

5 Επειδή στην προτεινόμενη υλοποίηση υπολογίζονται σύμφωνα με το ερώτημα και με θέση-ειδικούς πίνακες τα στατιστικά στοιχεία για την εύρεση ομοιοτήτων, έπεται ότι θα αναμένονται πιο ακριβή αποτελέσματα από την [5]

5.6 Αποτελέσματα

Σύμφωνα με το [5] τα αναμενόμενα αποτελέσματα της υλοποίησης του προτεινόμενου αλγορίθμου φαίνονται στην παρακάτω εικόνα για δεδομένες γνωστές ακολουθίες ερωτήματος της βάσης δεδομένων UNIPROT στη βάση δεδομένων SWISSPROT.

sequence ID	*Length*	*E*	*HSPs*	*Time(s)*
1. A4T9V0	64	1000	621	1.2
2. Q2IJ63	128	1000	955	1.7
3. P28484	256	1000	569	2.7
4. Q1JLB7	512	10	405	4.6
5. P08715	1024	10	104	9.7
6. Q8IYD8	2048	10	738	18.1
7. Q06277	4095	10	572	36.7

sequence ID	*Length*	*GPU (s)*	*CPU (s)*	*Speedup*
1. A4T9V0	64	1.2	3.2	2.7
2. Q2IJ63	128	1.7	4.6	2.7
3. P28484	256	2.7	6.4	2.4
4. Q1JLB7	512	4.6	10.4	2.3
5. P08715	1024	9.7	16.5	1.7
6. Q8IYD8	2048	18.1	30	1.7
7. Q06277	4095	36.7	70	1.9

Η υλοποίηση έτρεξε σε host Mac Pro με λειτουργικό Ubuntu 8.10 και κάρτα γραφικών GeForce 8800 GTX σύμφωνα με [5]. Ο αριστερός πίνακας δείχνει τους αναμενόμενους χρόνους των εντοπισμών HSP και τον ολικό χρόνο συμπεριλαμβανομένου και του αλγορίθμου τοπικής

στοίχισης. Εδώ θα πρέπει να επισημανθεί ότι αναμένονται χειρότεροι χρόνοι από ότι στην υλοποίηση [52] με ντετερμινιστικά πεπερασμένα αυτόματα, λόγω του ότι δεν χρησιμοποιούνται περιοχές μνήμης γρήγορης πρόσβασης της μνήμης της GPU όπως στην [52]. Επίσης ο μηχανισμός εύρεσης ομοιοτήτων λέξεων με ντετερμινιστικά πεπερασμένα αυτόματα είναι πιο γρήγορος απ' ότι η αναζήτηση με το lookup table σύμφωνα με [4], [52] και [53]

6 Βιβλιογραφικές Πηγές

1. **Waterman M.S.** Bul Math Biology 46 pp 473-500, 1984
2. **Needleman S.B.& Wunsch C.D.** Journal Mol. Biology 1970, 48, p 442-453
3. **Sellers P.H.** SIAM Journal in Applied Math.1976,26 p 787-793
4. **Altschul S.F. Gish W.** Basic Local Alignment Tool J. Mol. Biology 1990, 215 pp.403-410
5. **Ling C., Benkrid K.** Design and Implementation of a CUDA Compatible GPU Based core for Gapped BLAST Algorithm ICCS 2010
6. **Liu W., Schmidt B.** Accelerating BLASTP on CUDA enabled Graphics Hardware IEEE trans. on Comp. Biology and Bioinformatics Vol 8 No 6 Dec 2011 pp 1678-1684
7. **Ian Korf, Mark Yandell, Joseph Bedell,** An essential guide to Basic Local Alignment Tool, OReilly Publications
8. **Karlin S. Altschul S**. 1990 Proc Natl. Acad. Sci. USA 87, pp 2264-2268
9. **Lipman, D. J. & Pearson, W. R**. (1985). Rapid and sensitive protein similarity searches. Science,227, 1435–1441.
10. **Arratia, R., Gordon, L. & Waterman, M. S.** (1986). An extreme value theory for sequence matching, Ann. Stat. 14, 971–993
11. **Arratia, R., Gordan, L. & Waterman, M. S**. (1990). The Erdos-Renyi law in distribution, for coin tossing and sequence matching. Ann. Stat. 18, 539–570.
12. **Karlin, S., Bucher, P., Brendel, V. & Altschul, S. F.** (1991). Statistical methods and insights for protein and DNA sequences. Annu. Rev. of Biophys. Biophys. Chem. 20, 175–203.
13. **Waterman, M. S.** (1995). Introduction to computational biology. Chapman and Hall, London.
14. **Altschul, S.F. & Gish, W.** (1996) "Local alignment statistics." Meth. Enzymol. 266 460-480.
15. **Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D.J**. (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." Nucleic Acids Res. 25:3389-3402
16. **Smith, T.F., Waterman, M.S. & Burks, C.** (1985) "The statistical distribution of nucleic acid similarities." Nucleic Acids Res. 13:645-656.
17. **Collins, J.F., Coulson, A.F.W. & Lyall, A**. (1988) "The significance of protein sequence similarities." Comput. Appl. Biosci. 4:67-71
18. **Mott, R.** (1992) "Maximum-likelihood estimation of the statistical distribution of Smith-Waterman local sequence similarity scores." Bull. Math. Biol. 54:59-75
19. **Waterman, M.S. & Vingron, M**. (1994) "Rapid and accurate estimates of statistical significance for sequence database searches." Proc. Natl. Acad. Sci. USA 91:4625-4628
20. **Waterman, M.S. & Vingron, M.** (1994) "Sequence comparison significance and Poisson approximation." Stat. Sci. 9:367-381.
21. **Pearson, W.R.** (1998) "Empirical statistical estimates for sequence similarity searches." J. Mol. Biol. 276:71-84.

22. **Arratia, R. & Waterman, M.S.** (1994) "A phase transition for the score in matching random sequences allowing deletions." Ann. Appl. Prob. 4:200-225.
23. **Pearson, W.R. & Lipman, D.J.** (1988) Improved tools for biological sequence comparison." Proc. Natl. Acad. Sci. USA 85:2444-2448.
24. **Smith, T.F. & Waterman, M.S.** (1981) "Identification of common molecular subsequences." J. Mol. Biol. 147:195-197.
25. **Chao,K.-M., Pearson,W.R. and Miller,W.** (1992) Comput. Appl. Biosci., 8, 481–487.
26. **Wilbur,W.J. and Lipman,D.J.** (1983) Proc. Natl. Acad. Sci. USA, 80,pp726–730.
27. **McLachlan,A.D.** (1983) J. Mol. Biol., 169,pp 15–30.
28. **Staden,R**. (1984) Nucleic Acids Res., 12, 505–519.
29. **Schneider,T.S., Stormo,G.D., Gold,L. and Ehrenfeucht,A.** (1986) J. Mol. Biol., 188, 415–431.
30. **Taylor,W.R.** (1986) J. Mol. Biol., 188, 233–258.
31. **Berg,O.G. and von Hippel,P.H.** (1987) J. Mol. Biol., 193, 723–750.
32. **Dodd,I.B. and Egan,J.B.** (1987) J. Mol. Biol., 194, 557–564.
33. **Gribskov,M., McLachlan,A.D. and Eisenberg,D.** (1987) Proc. Natl. Acad. Sci. USA, 84, 4355–4358.
34**. Patthy,L.** (1987) J. Mol. Biol., 198, 567–577.
35. **Stormo,G.D. and Hartzell,G.W**. III (1989) Proc. Natl. Acad. Sci. USA, 86, 1183–1187.
36. **Tatusov,R.L., Altschul,S.F. and Koonin,E.V.** (1994) Proc. Natl. Acad. Sci. USA, 91, 12091–12095.
37. **Yi,T.-M. and Lander,E.S.** (1994) Protein Sci., 3, 1315–1328.
38. **Henikoff,S. and Henikoff,J.G.** (1997) Protein Sci., 6, 698–705.
39. **Bucher,P., Karplus,K., Moeri,N. and Hofmann,K**. (1996) Comput. Chem., 20, 3–23.
40. **Sellers, P.H.** (1980) J. Algorithms, 1, pp 359–373
41. **Lawrence,C.E., Altschul,S.F., Boguski,M.S., Liu,J.S., Neuwald,A.F. and Wootton,J.C.** (1993) Science, 262, 208–214.
42. **Altschul,S.F., Carroll,R.J. and Lipman,D.J.** (1989) J. Mol. Biol., 207, 647–653.
43. **Sibbald,P.R. and Argos,P.** (1990) J. Mol. Biol., 216, 813–818.
44. **Sander,C. and Schneider,R**. (1991) Proteins, 9, 56–68.
45. **Gerstein,M., Sonnhammer,E.L. and Chothia,C.** (1994) J. Mol. Biol., 236, 1067–1078.
46. **Henikoff,S. and Henikoff,J.G.** (1994) J. Mol. Biol., 243, 574–578.
47. **Thompson,J.D., Higgins,D.G. and Gibson,T.J**. (1994) Comput. Appl. Biosci., 10, pp19–29.
48. **Eddy,S.R., Mitchison,G. and Durbin,R.** (1995) J. Comput. Biol., 2, 9–23.
49**. Gotoh,O.** (1995) Comput. Appl. Biosci., 11, 543–551.
50. **Krogh,A. and Mitchison,G. (1995) In Rawlings,C., Clark,D., Altman,R., Hunter,L., Lengauer,T. and Wodak,S.** (eds.), Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology. AAAI Press, Menlo Park, CA, pp. 215–221.
51. **Henikoff,J.G. and Henikoff,S.** (1996) Comput. Appl. Biosci., 12, pp 135–143.
52 https://sites.google.com/site/liuweiguohome/software
53 **Weiguo Liu, Bertil Schmidt, Wolfgang Muller**, Cuda BLASTP, Accelerating BLASTP on Cuda Enabled Graphics Hardware IEEE trans On Bioinformatics Vol 8 nov 2011
54 **Y. Liu, D. Maskell, and B. Schmidt**, "CUDASW++: Optimizing Smith - Waterman Sequence Database Searches for CUDA-Enabled Graphics Processing Units," BMC Research Notes, vol. 2, article no. 73, 2009.